

**A MULTIAGENT FRAMEWORK FOR A DIAGNOSTIC AND
PROGNOSTIC SYSTEM**

A thesis
Presented to
The Academic Faculty

by

Irtaza Barlas

*In Partial Fulfillment
of the requirements for the Degree
Doctor of Philosophy in Electrical Engineering*

Georgia Institute of Technology
November, 2003

Copyright © 2003 by Irtaza Barlas

**A MULTIAGENT FRAMEWORK FOR A DIAGNOSTIC AND
PROGNOSTIC SYSTEM**

Approved:

Dr. George J. Vachtsevanos, Advisor

Dr. Anthony Yezzi

Dr. Linda M. Wills

Dr. Magnus Egerstedt

Date Approved: November 25, 2003

To Zehra
The intelligent part of my life
&
To Meral and Daniyal
The dynamic part of my life

ACKNOWLEDGEMENT

First and foremost I would like to thank my advisor, Dr. George J. Vachtsevanos, who has been a continual source of ideas and support throughout my program. He was instrumental in the basic concepts of the research as well as its details. I would also like to thank the members of my committee – Dr. Anthony Yezzi, Dr. Linda Wills, and Dr. Magnus Egerstedt for their helpful comments. Their valuable suggestions have brought this thesis up to a higher level. I am also indebted to Dr. Itzhak Green for serving as an external committee member. His experience with mechanical aspects of diagnostics and prognostics gave me a better insight into certain issues. I appreciate the help of all the current and past ICSL team members, Sharon Lawrence, Dr. Antonio Ginart, Dr. Freeman Rufus, Nicholas Propes, Dr. Scott Clements, Yuhua Ding, Dr. Seungkoo Lee, Dr. GuangFan Zhang, Yingchuan Zhang, Tamir Hegazy, Taimoor Saleem and all the rest.

Finally, I would like to appreciate my family, specifically my parents and parent-in-laws, my aunts, and my uncle for their prayers. It has been a very long and testing part of life for my wife, kids, sisters and brother. Thank you for your patience and support.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	IV
TABLE OF CONTENTS	V
LIST OF TABLES	IX
LIST OF FIGURES	X
LIST OF FIGURES (CONTD.)	XI
SUMMARY	XII
1 INTRODUCTION	14
1.1 Motivation	14
1.2 Problem Statement & Significance	15
1.3 Background of the Problem	18
1.3.1 Diagnostics & Prognostics	18
1.3.2 Intelligent Agents	20
1.3.3 Multiagent Systems	22
1.3.4 Diagnostic and Prognostic Frameworks	25
1.4 Scope of the Thesis	34
1.5 Assumptions	36
2 DYNAMIC ACTIVE DIAGNOSTIC ARCHITECTURE	38
2.1 Overview	38
2.2 Background	38
2.3 Active Diagnosis	39
2.3.1 Design-time Active Diagnosis	39
2.3.2 Immunity-Based Active Diagnosis	40
2.3.3 Discussion	41

2.4 In Search of an Intelligent Diagnostic System	41
2.5 Dynamic Active Diagnostics: Theoretical Framework	42
2.6 Dynamic Active Diagnostics: Implementation Details	44
2.6.1 Intelligent Agents: A Natural Choice	45
2.6.2 Intelligent Agent Architecture for DAD Framework	46
2.7 Intelligent Diagnostic Agent Architecture	49
2.8 Global View	52
2.8.1 A Typical Scenario	52
2.8.2 Proposed Scenario	52
2.9 Conclusion	53
3 DYNAMIC DIAGNOSTICS	54
3.1 Introduction	54
3.2 Diagnostic Reasoning & Learning: A Historical Note	55
3.3 Why CBR?	56
3.4 Case-Based Reasoning Process	57
3.6 Implementation Approach	60
3.6.1 Knowledge Elicitation & Organization	60
3.6.2 Methodology for Selecting Best Experiences	65
3.6.3 Methodology for Modifying an Old Experience – Adaptation	70
3.6.4 Learning from Experiences	75
3.6.5 Selecting Best Case(s) – Subsequent Remembering Process	77
3.7 Conclusion	78
4 ACTIVE DIAGNOSTICS	80
4.1 Introduction	80
4.2 What is Active Diagnostics	80
4.3 Our Approach	81

4.4 Active Implementation	82
4.5 Dynamic Goal Selection	83
4.5.1 Last Known World-State	85
4.5.2 Abandoning Goals	85
4.5.3 Feature Extraction	85
4.5.3 Search Space and Success Factors	87
4.6 Conclusions	88
5 COORDINATED DIAGNOSTICS	90
5.1 Introduction	90
5.2 Multiagent View	90
5.2.1 Big Picture	91
5.2.2 Coordination/Collaboration Mechanism	92
5.3 Why Coordination Makes Sense	92
5.4 Asking for Help – Coordination	95
5.4.1 Making Use of Assistance	96
5.5 Being Helpful – Collaboration	97
6 PERFORMANCE ASSESSMENT & VALIDATION	100
6.1 Introduction	100
6.2 Performance Measures	101
6.2.1 Accuracy of Early Detection and Validation	101
6.2.2 Precision of Prognosis at Validation	102
6.2.2 Accuracy of Prognosis	102
6.3 Experimental Results	103
6.4 Discussion	106
7 PERFORMANCE CONDITIONS	108
8 CONCLUSIONS	111

APPENDIX A INTELLIGENT AGENTS - AN OVERVIEW	113
A.1 Intelligent Agents - Philosophical Issues	114
A.2 Intelligent Agent Architectures	117
A.2.1 Reactive Agent Architecture	118
A.2.2 Logic Based Agents Architecture	118
A.2.3 BDI Agent Architecture	118
A.2.4 Layered Architecture	119
REFERENCES	120
VITA	124

LIST OF TABLES

Table 3.1 Adaptation process for different failure experiences	73
Table 3.2 Case Selection Rule-Base	79
Table 5.1 Portion of the case selection rule-base	96
Table 6.1 Experimental Results	104

LIST OF FIGURES

Figure 1.1 Top-level view of an Intelligent Agent	8
Figure 1.2 A diagnostic framework with Central Control and Knowledge Base	13
Figure 1.3 Diagnostics and Prognostics Framework for Gas Turbine Engine	14
Figure 1.4 PEDS Framework for Diagnostics and Prognostics	15
Figure 1.5 A diagnostic framework with Distributed Control and Knowledge Base	16
Figure 1.6 Simplified MPROS Architecture	17
Figure 1.7 DSN FDIR Operational Infrastructure	19
Figure 2.1 A UUT and a conventional passive diagnostic system	31
Figure 2.2 A UUT and its active diagnostic system	31
Figure 2.3 Implementation of DAD framework by an IA and related modules	36
Figure 2.4 Schematic diagram of a diagnostic agent	38
Figure 2.5 A typical diagnostic and prognostic framework with related activities	39
Figure 2.6 Global view of multiagent diagnostic and prognostic framework	40
Figure 3.1 Basic case-base reasoning cycle	43
Figure 3.2 CBR Main Processing Blocks [Dubitzky 1997]	45
Figure 3.3 Design methodology of a dynamic diagnostic system	46
Figure 3.4 (a) A general case template used in diagnosis/prognosis; (b) Example of a case designed for a CBR for printer diagnostics [Watson 1997]	48
Figure 3.5 (a) A Case structure for the DAD case library (b) Template of a generic case for diagnosis and prognosis. (c) Example of an experience of a bearing failure	50
Figure 3.6 Remembering Process in CBR [Dubitzky 1997]	52

LIST OF FIGURES (contd.)

Figure 3.7 Example of remembering process	55
Figure 3.8 Adaptation Process in CBR [Dubitzky 1997]	57
Figure 3.9 Failure experiences of different cases	59
Figure 3.10 Example of a simple adaptation process	62
Figure 3.11 Learning Process in CBR [Dubitzky 1997]	63
Figure 3.12 Case Distances for Failure Experiences	64
Figure 4.1 Active Diagnostic Cycle	70
Figure 4.2 Algorithm to select possible goals dynamically	73
Figure 4.3 Example of an active diagnostic cycle	76
Figure 5.1 A community of DAD Agents for cooperative diagnostics	78
Figure 5.2 Modified Active Diagnostic Cycle	85
Figure 5.3 Example of a Coordination/Collaboration among peer DAD agents	86
Figure 6.1 Example of prediction accuracy measure	89
Figure 7.1 Detection Performance as it changes with number of failures	92
Figure 7.2 Prediction Performance as it changes with number of failures	92

SUMMARY

Complex dynamical systems are subjected to failure modes that tend to reduce uptime and present major challenges to maintainers. Initiatives, such as Condition-Based Maintenance (CBM) have addressed these challenges by proposing diagnostic systems with prognostic capabilities. As a result of these developments many architectures have been designed to perform diagnostics and prognostics on centralized and distributed systems. Due to the ad hoc and specific nature of design and implementation of these architectures, they are difficult to port, inefficient to scale, and hard to maintain. Another source of the problem is the variability of the problem domain itself. Hence in large-scale systems, failures do not usually happen in identical manner. Moreover, the statistical information is also either not available or too conservative. A conceptual aspect is that although some of these systems are designed for decentralized domains, the knowledge base and control of the architectures remain centralized.

The shortcomings of the current systems stem from the limitations of their frameworks. The framework is typically designed on the *passive*, *open loop*, *static*, and *isolated* notions of diagnostics. In essence, the framework does not observe its diagnostic results (open-looped), hence can not improve its performance (static). Its *passivity* is attributed to the fact that an external event triggers the diagnostic or prognostic action. There is also no effort in place to team-up the diagnostic systems for a collective learning, hence the implementation is *isolated*.

In this research we extend the current approaches of the design and implementation of diagnostic and prognostic systems by presenting a framework that is

based on a class of Distributed Artificial Intelligence (DAI) systems, namely the Multiagent systems. This framework provides for scalable and robust solution to the problem. This research is novel in its approach and significant since it extends the current architectures by providing such unique features to the framework, as learning, self-organization, and coordination. Since multiagent systems are groups of loosely coupled problem solvers, a distributed framework from the knowledge and control perspective emerges naturally. It also provides fault-tolerance and robustness to the framework.

We show how the Intelligent Agent paradigm is suitable for the conceptual framework of a *dynamic*, *active*, and *coordinating* diagnostic and prognostic system.

We validate the framework by using feature data from bearings.

Major contributions of this research are

- Introduction of new concepts of dynamic, active, and cooperative diagnostics and prognostics.
- Design and implementation of a novel episodic learning mechanism with explicit temporal dimension, namely Dynamic Case-Based Reasoning approach.
- An Intelligent Agent architecture for the implementation of dynamic, active diagnostics.
- A Multiagent framework that introduces a novel cooperative paradigm to diagnostics and prognostics.

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Complex dynamical systems such as aircraft, chemical processes, power plants, etc. are subjected to failure modes that tend to reduce uptime and present major challenges to maintainers. 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) entails maintenance of equipment when needed as compared to scheduled maintenance.

Implementations of CBM practices require on-line real-time monitoring of components/subsystems, diagnostic tools that detect incipient fault conditions and prognostic algorithms that predict accurately and precisely the remaining useful lifetime of failing components. The design of CBM systems has been approached thus far through ad hoc methods or a centralized architecture where data management, decision support etc. occupy a central stage. Such a configuration may be subjected to errors/faults and lacks the ability to learn, adapt, and be flexible. The configurations that claim to be decentralized, still use either a centralized knowledge-base, a global model, and/or a centralized control.

Many popular CBM schemes rely on recent history of the signal patterns and their analysis to create predictions. Some rule-bases may be used to augment the analysis and to incorporate the heuristic information in the predictions. Knowledge engineering practices have been developed over the years to translate human diagnosticians' expertise in a form that can be of use for the software-based expert systems and rule-based systems.

A common feature of the CBM architecture as well as that of the other diagnostic systems is that it is an open-loop design. Therefore, the sole responsibility of these architectures has been limited to such mundane activities as sounding an alarm, or blinking a light on the monitoring station, or merely creating a failure event log. The current software technologies provide a promise for these systems to do more than that.

1.2 PROBLEM STATEMENT & SIGNIFICANCE

A conventional framework can be labeled as a *passive, open loop, and static*, diagnostic and prognostic system. The notion of passivity represents that this system waits for an external event to trigger its diagnostic and prognostic routines. It is an open loop system since it simply reports the results of the diagnostic and prognostic algorithms, without any effort to modify the system or the problem domain. It is static, since its diagnostic capabilities remain constant over its lifetime.

In designing a diagnostic and prognostic framework for medium to large-scale systems a major issue has been the scarcity of failure data. Large-scale systems are not tested by the manufacturers in a manner similar to say, light bulbs. Therefore, unlike light bulbs for example, it is not feasible to destroy thousands of large-scale systems to create a statistical distribution for every failure. This creates a problem for the conventional

diagnostic and prognostic approaches, which rely heavily on some kind of statistical data. Even when the failure data is available, the variability in most of the failures makes it very difficult for diagnostic and prognostic systems to function reliably.

The main problem with this issue is the *static* nature of the framework. In order to understand this problem, let us examine how human diagnosticians handle this issue. A simple observation is that human reasoning is based upon experience (heuristics) but they keep adjusting their reasoning according to the condition at hand. In a worst-case scenario, if a human is unable to solve a problem, he/she can ask for help from other diagnosticians who may have encountered that particular problem. The conventional diagnostic frameworks can not duplicate this variable and cooperative human reasoning very well.

Another aspect of designing diagnostic and prognostic frameworks for large-scale systems is that a failure can cause significant damage resulting in not only hazardous conditions for humans but also potentially massive loss of time and resources. Therefore, it is not only important to diagnose a possible failure but also to take proper actions automatically to do something about the possible failure. These actions can be as simple as sounding an alarm, to as complex as changing the operating mode of the system to bring it to a safe mode. This requires a *closed-loop* behavior that is generally not available in conventional diagnostic and prognostic framework.

Most of the systems are now armed with a host of sensors that monitor every vital aspect of the process. These, so-called “Sensor-Rich” systems, generate massive amounts of data. It is obvious that new computational techniques and implementation frameworks

need to be developed in order to identify and develop scalable processing architectures for gathering and interpreting data from many sources that may be embedded in a system.

This data explosion is not limited to the realm of diagnostics. The exponential increase in the Internet usage has also created a pressing need for the software engineers to devise new techniques and methodologies to handle vast amounts of data.

With the growing complexity of problem domains, research in Artificial Intelligence (AI) has taken a center stage in development of new software technologies to handle problems specifically the one that can not be well defined and hence a routine procedural solution can not be obtained. Reasoning and learning strategies are being successfully developed and deployed for a number of applications.

Current research in the design of diagnostic and prognostic systems is driven by several market factors. The cost of the sensors and computing for example is coming down steadily. This has enabled manufacturers to incorporate more sensors into their systems. While the amount of data generated by these sensors is increasing, the reducing cost of networking, coupled with faster processing has guided the development of new distributed and networked diagnostic frameworks. These technologies are making it possible to design distributed diagnostic and prognostic systems that can be networked to a central location.

Some recent frameworks for diagnostics and prognostics have taken advantage of these trends. Swanson (2001) for example proposed an Intelligent Sensor layer with an Area Reasoner. These Intelligent Sensors are actually powerful computers that generate prognostic results rather than simple raw data. Roemer and Kacprzyński (2000) use neural-networks and data mining techniques for diagnostics and prognostics for gas

turbines. Li Pi Su *et al* (1999) developed a prognostic framework based upon model-based reasoning.

While each of these architectures has been successful in its respective application domain, it is very desirable to develop a robust and scaleable framework that is based upon the new software technologies. However, since current paradigms of diagnostic and prognostic frameworks are not able to address many issues such as the problem of scarcity of statistical data, handling of large amounts of decentralized sensor rich data, immediacy of actions related to a diagnostic or prognostic problem, a software framework alone will not be enough. In summary, the problem to be addressed is two folds: how to extend the diagnostic paradigm that will not be static and open-looped; and how to incorporate smart software technologies into the new diagnostic paradigm.

1.3 BACKGROUND OF THE PROBLEM

1.3.1 Diagnostics & Prognostics

Based on the conceptual similarities, the diagnostic approaches are divided into the following five categories [Debouk, 2000]: (i) fault-tree based methods; (ii) analytical redundancy methods; (iii) expert systems and knowledge-based methods; (iv) model-based reasoning methods; and (v) discrete-event systems based methods. Since focus of this proposal is the framework that uses a diagnostic technique, and not the diagnostics techniques themselves, therefore only an overview of the above-mentioned techniques is presented here. A detailed account can be found in [Pouliezos and Stavrakakis, 1994].

Fault-tree methods [Ulerich and Powers 1988, Viswanadham and Johnson 1988,] are a graphical representation that relates causes to effects in a system. The analytical reasoning methods [Frank, 1990] appear in the control systems literature, and involve the generation of the residual signals, and decision and fault isolation. Comparing predicted values of system variables with the actual observed values generates the residual signals, while the fault isolation is obtained by examining the residuals for likelihood of faults. Expert system methods [Foxvog, 1991 Ghafoor, 1989] are particularly useful for systems that are difficult to model. A set of rules is generated from the heuristic knowledge of experts. Model-based reasoning methods [Mozetic 1992, Reiter 1987] rely solely on the system description and observations of its behavior. Many discrete-event-model based approaches for failure diagnosis are being considered lately. In [Aghasaryan 1998, and Viswanadham 1988], Petri Nets are used to model concurrent alarm conditions in large distributed systems. In [Sampath 1995] the off-line diagnosability of DES is studied. A language-based approach is presented in [Sampath *et al* 1996] and system is modeled as a formal language that accounts for the normal as well as failed modes of operation.

Overall the prognostic approaches can be divided in the following five classes [Roemer *et al* 2001]: (i) experience-based; (ii) evolutionary; (iii) feature progression and AI-based; (iv) state estimator; and (v) physics-based prognostics. The experience-based method [Hadden *et al* 2000] is the least complex and is usually the only choice if there is no model of the subsystem or the component available. Typically failure or inspection data is compiled from legacy systems and a Weibull distribution or other statistical distribution is fitted to the data. The evolutionary prognostic [Engel *et al* 2000] approach relies on gauging the proximity and rate of change of the current component condition

(i.e. features) to known performance faults. The feature progression methods [Thakkar 2001, Khiripet 2001, Swanson 2001] utilize known transitional paths of measured or extracted features as they progress over time. Artificial neural networks (or other AI based techniques) are trained on features that progress through a failure. State estimation techniques [Swanson 2001] such as Kalman filters can be implemented as a prognostic technique. Future feature behavior is predicted by minimizing the error between a model and measurement. A physics based stochastic model [Roemer *et al* 2001] is a technically comprehensive modeling approach. It can be used to evaluate the distribution of remaining useful component life as a function of uncertainties in component strength/stress or condition for a particular fault.

1.3.2 Intelligent Agents

An IA is defined by Wooldridge as “a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.” Wooldridge suggests a *weak notion* of agenthood as against a *strong notion* [Wooldridge 1999]. The weak agenthood enjoys the properties of *autonomy*, *social ability* (that includes Agent-Communication Language ACL), *reactivity*, and *proactivity*. The *strong notion* of agenthood are based on, what is called the *mentalistic* notions, such as *belief*, *knowledge*, *intention*, and *obligation*. Other important notions of agenthood that have appeared in the context of Intelligent Agents are *Mobility* and *Rationality*. Agents with mobility, i.e. *mobile agents*, have attracted attention lately. These agents roam around the web, in order to reduce the network traffic, and perform useful services on behalf of the users. The notion of *rationality* is also considered as the essential characteristic. Thus Russell and Norvig define a *Rational*

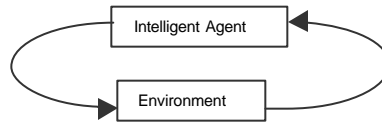


Figure 1.1 Top-level view of an Intelligent Agent

Agent in terms of its percepts and actions. Thus for each possible percept sequence, a rational agent should do whatever is expected to maximize its performance measure [Russell and Norvig 1995]. All researchers consider autonomy as the distinguishing feature for all software programs that claim to be IA. Autonomy for an agent is its ability to fulfill its tasks without the direct intervention of a human, i.e. it has control over its internal states and its behavior.

From the control perspective, an Intelligent Agent receives input from its environment and it acts according to specific goals to alter the environment. Thus, unlike Expert Systems, Intelligent Agents are decision-making systems that are embodied in an environment, as shown in Figure 1.1. This figure also shows how Intelligent Agent paradigm is used as an intelligent control system. The main difference between an intelligent agent and a control system is the fact that a control system is designed explicitly to reach a desired closed-loop behavior [Voos 2000].

Depending upon the design principles and domain requirements, several architectures have been proposed for the Intelligent Agents. Generally, these architectures are divided in the following four classes [Wooldridge 1999]:

1. Logic Based Agents – when decision making is done through logical deduction.

2. Reactive Agents – in which decision making is implemented as a direct mapping from situation to action.
3. Belief-Desire-Intention Agents (BDI) – in which decision making depends upon the manipulation of data structures representing the beliefs, desires, and intentions of the agents.
4. Layered Architectures – in which decision making is realized via various software layers, each of which is more-or-less explicitly reasoning about the environment at different levels of abstraction.

A detailed introduction to these architectures can also be found in [Sycara 1998].

Of these architectures, the BDI approach is the most popular and the most complex one. The simplest architecture is that of the reactive agent. Since, relationship between individual behaviors, environment, and overall behavior is not understandable, therefore it is hard to engineer reactive agents to fulfill specific tasks.

1.3.3 Multiagent Systems

Research in Multiagent systems (MASs) is concerned with the study, behavior and construction of a collection of possibly pre-existing autonomous agents that interact with each other and their environments. Study of such systems goes beyond the study of individual intelligence to consider, in addition, problem solving that has social components [Sycara 1998]. An MAS is defined as a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge.

The characteristics of MASs are that (1) each agent has incomplete information; (2) there is no system global control; (3) data are decentralized; and (4) computation is

asynchronous. In the context of this thesis, we are focusing upon the approaches for design of Multiagent systems. Kinny introduced a methodology that extends the object-oriented design methods with some agent-based concepts [Kinney *et al* 1996]. It is aimed at the construction of a set of models that define an agent system specification. It provides both “external” and “internal” models. While the former presents a system-level view, with the main visible components being the agents themselves, the later is concerned with the internals of agents, their beliefs, desires, and intentions. The methodology works by first identifying the relevant “roles” in the application domain. These are developed into an “agent class hierarchy”. Responsibilities are associated with each role and services identified. Next, “goals” are associated with each service. For each goal, plans are determined that may be used to achieve it. A belief structure of the system is determined next.

Wooldridge [Wooldridge *et al* 2000] presented the Gaia methodology that also supports both the micro-level (agent-structure) and macro-level (agent society and organization structure) of agent development. It is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed so that it can be directly implemented. The analyst moves from abstract to increasingly concrete concepts. It encourages the “organizational design” paradigm for the development of agent-based systems.

The Multiagent Systems Engineering Methodology (MaSE) of Wood and DeLoach [DeLoach 1999], and Zeus Toolkit of Nwana [Nwana *et al* 1999] are two similar initiatives. Their methodology is similar to Gaia with respect to generality and the application domain support. They go further regarding support for automatic code

creation through their respective toolkits. Their goal is to lead the designer from the initial system specification to the implemented system.

The fact the Unified Modeling Language (UML) is a de facto standard for object-oriented modeling inspired research to design extensions of UML for Agent-Oriented domain. Odell *et al* [Odell *et al* 2000] have discussed several ways in which the UML notation might usefully be extended. The extensions include support for expressing concurrent threads of interaction and a notion of “role” that allows modeling of an agent playing many roles.

Sycara [Sycara *et al* 1996] developed the RETSINA (Reusable Task Structure-based Intelligent Network Agents) computational infrastructure. This architecture employed three types of agents. *Interface agents* that interact with the user receiving user specifications and delivering results. *Task agents* help users perform tasks by formulating problem solving plans and carrying out these plans through querying and exchanging information with other software agents. *Information agents* provide intelligent access to a heterogeneous collection of information resources.

The methodologies discussed so far are broadly grouped as those that take their inspiration from object-oriented (OO) development approach. There are obvious advantages to such an approach, the most obvious being that the concepts, notations, and methods associated with object-oriented analysis and design are increasingly familiar to a mass audience of software engineers. However, there are several disadvantages. First, the kind of decomposition that OO methods achieve is at odds with the type of decomposition suitable for agent oriented designs. Agents are more coarse-grained computational objects than are objects. Thus it is challenging to achieve the correct

decomposition of entities into either agents or objects. Another problem is that OO methodologies do not allow us to capture many aspects of agent systems, such as proactivity, negotiation, cooperation, etc.

1.3.4 Diagnostic and Prognostic Frameworks

Implementation of a diagnostic algorithm, possibly with prognostic features, is a challenging task. Issues like, how the diagnostic information is processed, and how it is used for the control of the diagnostic environment, generally form the basis of the design of the framework. Another important question is about the knowledge base of the system and how it is distributed among different levels of control. In almost all the scenarios, implementation is specific to the problem domain and depends upon such factors as available computational resources, physical limitations, and timing constraints.

Since there is no general framework for implementation there is no comparative study on the performance measures of these frameworks. Some of the quantitative performance measures of interest are: *accuracy of detection and validation*, that is how accurate a failure is detected and validated, and does this rate changes with time, and *accuracy and range of prediction*. .

Part of this proposal deals with the definitions of performance measures. Based upon the differences in the control and knowledge distribution, we have grouped the overall diagnostic systems into two categories, i.e. systems that have Central Control and Knowledge base (CCK); as opposed to the systems that have Distributed Control and Knowledge base (DCK).

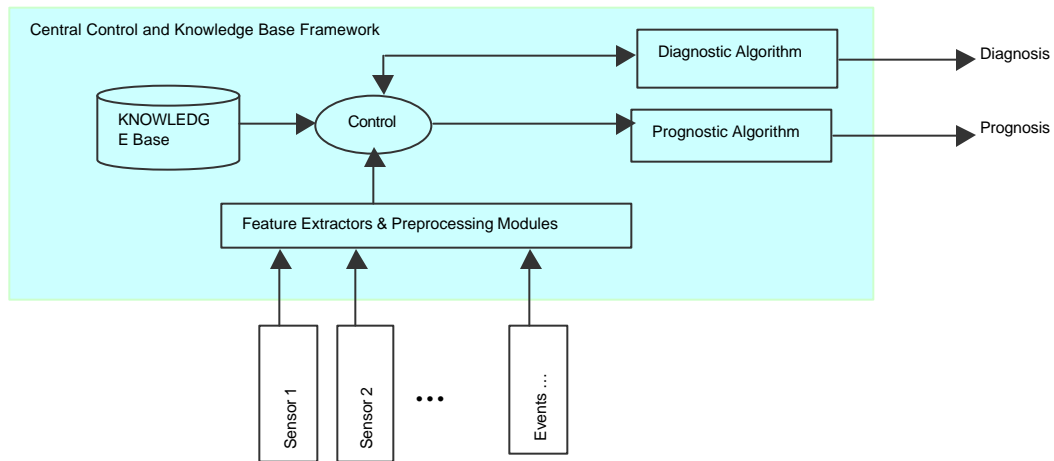


Figure 1.2 A typical diagnostic and prognostic framework with Central Control and Knowledge Base

Centralized Control and Knowledge Base (CCK) Frameworks

Most of the diagnostic frameworks [Roemer *et al* 2000 and Thakkar *et al* 2001] that are designed for sensor-rich systems fall under the CCK category. Figure 1.2 shows a typical scenario. CCK framework takes in sensor data and/or external events and generates a diagnosis. In certain implementations this diagnosis triggers a set of prognostic algorithms that generate remaining-useful-life (RUL) estimates. The control of the framework is centralized in the sense that modules are activated by a central control, using one central knowledge base that contains all the information related to all the sensors, failure conditions, and related parameters. Two frameworks that fall under CCK category are discussed next.

Roemer developed diagnostic and prognostic framework for gas turbine engines. The system works first by feeding the sensor data into the diagnostic algorithms for fault detection, isolation, and classification. Results from multiple diagnostic algorithms undergo a fusion operation to provide a more confident diagnosis. The prognostic module

works in parallel, utilizing the current sensor data as well as the results from the diagnostic algorithms to predict future time-to-failure and/or degraded engine condition. As an extra step the prognostic failure results are used in risk-based analysis to optimize the time for performing maintenance tasks. In this regard this system is a good example of the CBM architecture. This framework is shown in Figure 1.3.

Vachtsevanos [Thakkar and Vachtsevanos 2001] developed a framework called PEDS (Prognostic Enhancements to Diagnostic System), which is another example of the

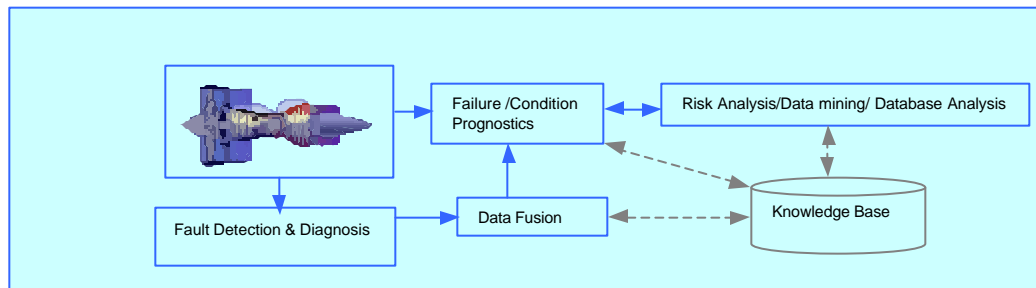


Figure 1.3 Diagnostics and Prognostics Framework for Gas Turbine Engine

CCK architecture. It is a database-oriented design that attempts to be a generic diagnostic and prognostic system. Although the database is not explicitly considered as a knowledge base, it keeps most of the configuration and provides a generic schema. The Case-Based Reasoner is employed to incorporate reasoning to propose a course of action based on the previous cases in the case library. There are some unique features in this system architecture, such as the *causal adjustment* module, and the *usage pattern identification* modules. The risk assessment methodologies are implicitly used in the selection of the failure modes in the database. The control is centralized in the *event dispatch* module. Modules are written in the COM/DCOM software framework. By leveraging the

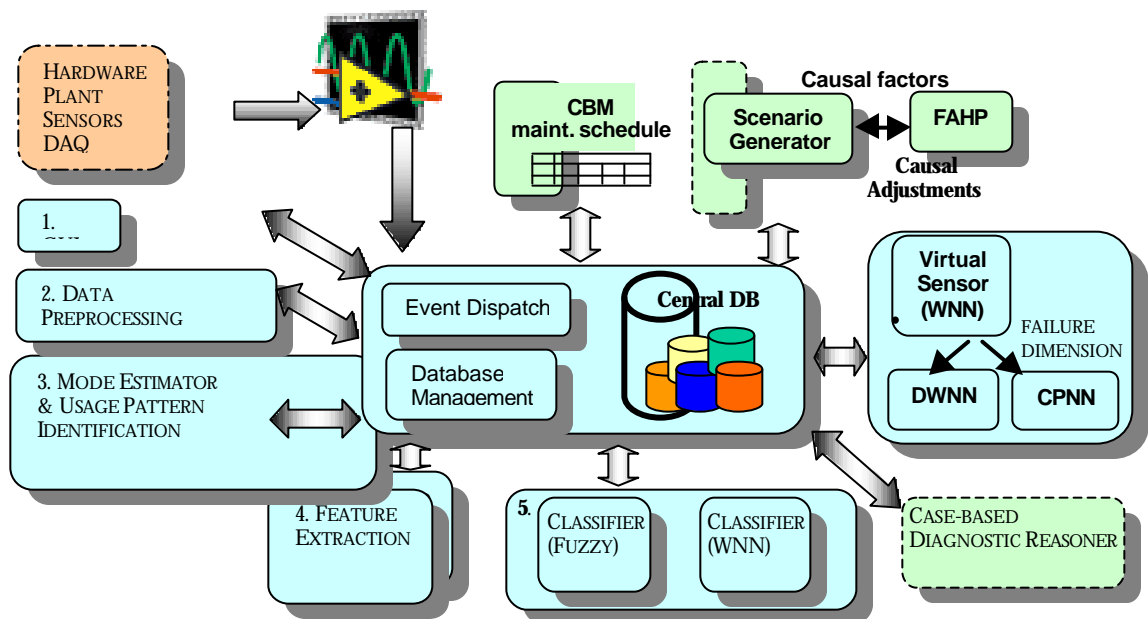


Figure 1.4 PEDS Framework for Diagnostics and Prognostics

component technology of Microsoft the system claims that it can be implemented in a distributed manner.

These architectures show common design strategies employed for the diagnostic and prognostic systems with central control and central knowledge base. Although most of the performance measures of interest are not documented for the two architectures, some can be inferred from the design description. It can be seen that both architectures, and in general all architectures that fall under the category of CCK framework, are flat organization of components. Diagnostics precedes prognostics and some kind of learning, whether by data mining or by case-based reasoning is applied to improve the overall accuracy of the system. The frameworks can not tolerate failure of individual components (reliability measure). Both frameworks handle data uncertainty by using fuzzy rule bases. Uncertainty is handled by the database/risk analysis in the first architecture and by the

CBR in the second architecture (robustness measure). Although learning strategies are employed, they are meant for improvement of results and the system does not organize itself from learning (entropy measure). Both architectures and in general the CCK frameworks are not easily extensible if the scope of the problem grows many folds (scalability measure). They however appear to be reusable for different problem domains (portability measure).

Distributed Control and Knowledge Base (DCK) Frameworks

Frameworks in this category are more modular and are typically designed for systems that are physically distributed (and possibly mobile), such as army vehicles, ships, airplanes, and plants. Control and knowledge base is hierarchical such that control and/or knowledge base is divided in layers. Distributed layers communicate over a network. Figure 1.5 shows components of a typical DCK framework. We will discuss two

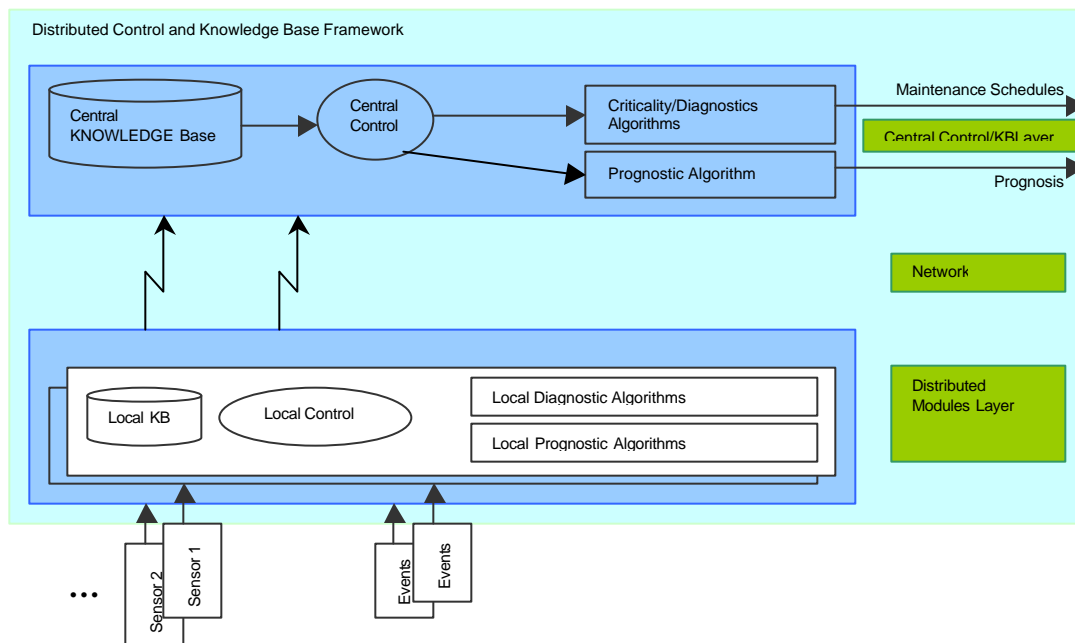


Figure 1.5 A diagnostic framework with Distributed Control and Knowledge Base

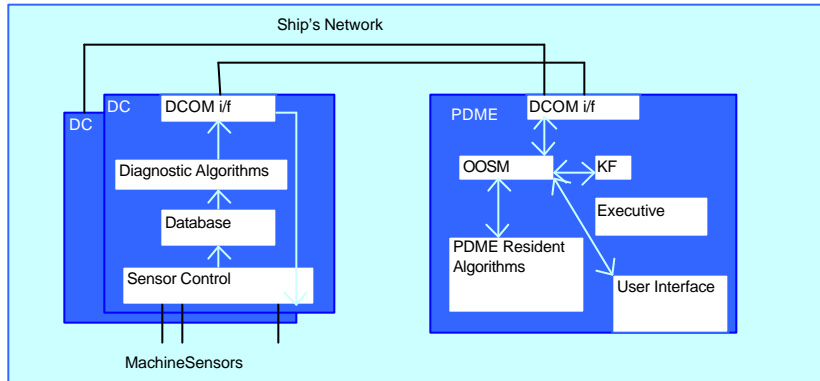


Figure 1.6 Simplified MPROS Architecture

implementations of diagnostic and prognostic schemes that fall under this category. A review of other approaches that fall under this category is also presented.

Honeywell, and its industrial and academic partners are developing a distributed shipboard system to perform diagnostics and prognostics on rotating equipment (e.g. engines, generators, and chilled water systems) for the Office of Naval Research (ONR) [Hadden *et al* 2000].

The CBM system MPROS (Machinery Prognostics/Diagnostics System) consists of sensors, distributed signal processing devices (called Data Concentrators - DCs), and a centrally located subsystem – PDME (Prognostics, Diagnostics, and Monitoring Engine). MPROS, as shown in Figure 1.6 includes several data streams that are integrated as necessary in the DCs (data fusion). A second level of integration (knowledge fusion - KF) occurs in the PDME. At this level, the outputs of different diagnostic and prognostic reasoning mechanisms are fused to yield the best possible analysis. PDME is the logical center of the MPROS system. It is also implemented using Microsoft's Component Object Model (COM) technology and its distributed version - DCOM. PDME is equipped with Object Oriented System (or Ship) Model (OOSM). Although the knowledge is

distributed among the remote and the central site, most of the intelligence is centered at the PDME module. PDME also has several diagnostic and prognostic modules. This suggests transfer of potentially significant amount of data. However, that is necessary since the model-based or rule-based reasoning is centered at PDME and this reasoning may be applied on the data from more than one DCs.

James presented an autonomous diagnostic and prognostic monitoring system for NASA's deep space network (DSN) [James *et al* 2000]. DSN is an international network of antennas that supports interplanetary spacecraft missions. Three deep-space communication facilities, placed approximately 120 degrees apart around the world, permit constant observation of spacecrafts. Sensors in this domain are placed in the spacecraft, and part of the Prognostics and Health Management (PHM) is implemented onboard for real-time estimates. These results are transmitted to the ground station for further FDIR reasoning and the estimation of time-to-criticality and wear, which in turn results in generation of maintenance schedules. An FDI module called BEAM provides fault diagnosis and prognosis at every level of operation using adaptive wavelet analysis, nonlinear information filtering, neuro-fuzzy system identification and stochastic modeling. A combination of Model-Based Reasoning and CBR are used in an expert system called SHINE that provides further fault detection and isolation using the heuristic knowledge. Figure 1.7 shows an operational infrastructure of the DSN FDIR.

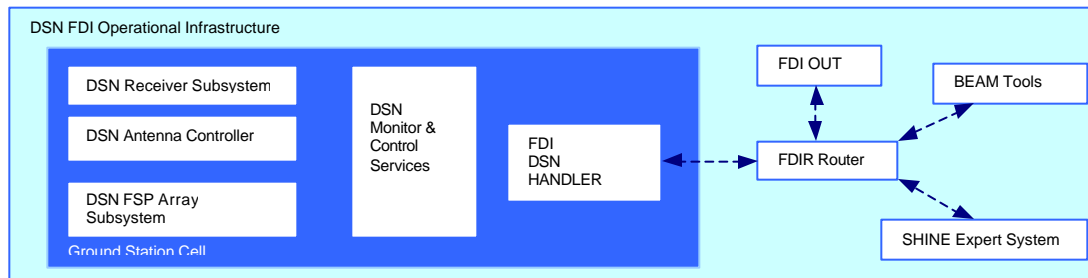


Figure 1.7 DSN FDIR Operational Infrastructure

Vachtsevanos presented a comprehensive framework for the diagnostics and prognostics of army vehicles as part of the Automated Diagnostic Improvement Program (ADIP) [Vachtsevanos 1998]. This framework works at multiple levels. At the first level there are embedded diagnostics in the vehicles that are used for “operational performance monitoring” and “abnormal performance detection”. An interesting feature of this framework is that it is not required that it should be connected to higher levels of control all the time. This is feasible considering the operational environment of the army vehicles. Thus the embedded system connects as per requirement. Operator of the vehicle is responsible for keeping an eye on the embedded diagnostic results. In case of an alarm or a periodic health check, the embedded system is connected to the next level of control and knowledge base. At this, so-called “Maintainer Level”, a much more thorough inspection of the system is performed.

The architectures presented above show typical design strategies employed for the diagnostic and prognostic systems with distributed control and/or knowledge base. In these architectures too, most of the performance measures of interest are not documented, however some can be inferred from the design description. It can be seen that these

architectures (and in general the DCK framework itself) are hierarchical organization of components – the most common scenario being of two layers of hierarchy. Diagnostics precedes prognostics and some kind of learning, such as model-based reasoning and case-based reasoning is applied to improve the overall accuracy of the system. Learning strategies employed are meant for improvement of results. System does not organize itself from learning. These architectures and in general the DCK frameworks are somewhat extensible, however their maintainability may become an issue for larger systems. They also appear to be reusable for different problem domains.

Several distributed diagnostic implementation schemes are found in the literature. Following is a review of some of these approaches.

In [Baroni *et al* 1999], the authors deal with large-scale distributed asynchronous event-driven systems, which they refer to as Active Systems. The active systems are modeled as communicating automata. Based on the observations, many representations of the parts of the active system are generated and eventually merged by using the history of observable events. Diagnostic information is generated on the basis of reconstructed behavior. In [Pencolé 2000], the authors are interested in diagnosing a telecommunication network composed of many sub-systems. They build local diagnosers for these sub-systems. All available local diagnoses are combined, using the history of observable events and the models of the sub-systems, to generate a global diagnosis. The combination is done while minimizing the computation of the overall diagnosis. In [Sengupta 1998], the authors discuss diagnosis problems in distributed systems composed of several spatially separated sites. At every site, there exists a diagnoser that partially observes the system and is in charge of diagnosing faults associated with the site.

Diagnosers are allowed to exchange information. In [Debouk 2000], the author proposes a discrete event model for decentralized information. The system model is divided in local sites and a coordinator. With the assumption that coordinator does not has the global knowledge about the system, three protocols and their variations are presented. Also, an optimized sensor selection solution is presented. In [Roemer *et al* 2001], the authors use the model of the physical plant at two levels of abstraction, a qualitative model and a hybrid model. They accommodate the moderate local computation resources by dividing the discrete diagnostic model into a set of local diagnosers that mimic the topology of the physical plant. The two diagnosers in this scheme communicate only if they represent the subsystems of the physical plant that are physically or informationally connected.

1.4 SCOPE OF THE THESIS

Prognosis is a very difficult task. It has been called the “Achilles heel” of CBM. The research presented in this thesis addresses the problem of prognosis and the problem of design of a diagnostic and prognostic framework by breaking down the problem into two steps. In the first step, the current notions of diagnostics and prognostics are extended. In the second step, a software paradigm is developed that can incorporate the advanced notions of diagnostics and prognostics.

As the first step a notion of “Dynamic Active Diagnostics” is presented. Although the term “active” has been used in conjunction with diagnostics before, it is presented in a different perspective in this thesis. An important part of this notion of diagnostics is that it is dynamic. The “dynamism” comes from learning. This research introduces a novel episodic learning methodology for the framework, namely the Case-Based Reasoning

approach with temporal aspects. The dynamic CBR was motivated by the fact that this learning technique captures a common problem solving approach of humans, whose experiences are not frozen moments in time, rather constitute a sequence of events. Therefore, an interesting *experience* consists of not only a fault signature but also its progression in time. This style of episodic reasoning with explicit temporal constraints is primarily the focus of this research. Since we expect that the faults exhibit variable signatures, these temporal variables are continuously updated to generate new experiences as the system learns from its closed-loop observations. A “post-mortem” of a failed experience is the key learning opportunity in the dynamic CBR for improving its performance by adapting old episodes to reflect new scenarios. These new experiences can comprise a temporal sequence of how a fault progresses, as well as how a false-alarm was registered. This allows the system to learn not only when it was successful in its detection and prediction of fault conditions, but also when it generated false alarms. These notions of dynamic Case-Based Reasoning (CBR) are presented in Chapter 3.

Another aspect of Dynamic Active Diagnostics is the closed-loop behavior. This closed-loop behavior is used internally by the CBR system, in which the system learns from the true-positives and false-positives, with the goal of improving the rate of false-positives and false-negatives. The closed-loop behavior also implies control. There are two approaches possible to handle the practical nature of the problem. The first approach is that of a diagnostician, when the control is limited to the diagnostic and prognostic elements of the system. The second approach is that of a system designer, and this control can be appended to the overall system control architecture. The third aspect of Dynamic Active Diagnostics is the proactive (as opposed to passive) behavior. This behavior is

encapsulated in the Intelligent Agent architecture. These concepts are discussed in Chapter 4.

An important feature of the agent-based architecture is that it can be scaled to work in a community of loosely coupled networked entities. This so-called Multiagent System implements a notion of Cooperative Diagnostics. This enables sharing of diagnostic and prognostic experiences between diagnostic and prognostic agents. These ideas are presented in Chapter 5.

In order to assess the performance of the proposed architecture, measures are defined in Chapter 6 and examples are presented when the components of the framework are tested against real-time data obtained from typical bearing failure modes. Chapter 7 presents a brief overview of the conditions under which the performance of the system can be assessed as being successful or not. Chapter 8 concludes the thesis with possible future research directions.

1.5 ASSUMPTIONS

The following assumptions are made about the presented framework:

1. The failures that are being monitored by the system are *diagnosable*. That is, each failure can be uniquely identified by the observable state of the UUT and/or its controller events.
2. The features used to detect faults are correct and computable in the given time-window. That is, the system does not reason explicitly about the problem related to feature selection, and it assumes that it can get any feature data whenever it asks for it.

3. The faults that are to be prognosed are *not instantaneous*. That is, these faults exhibit a trend and their progression to a failure or a hazard state is slower than the computational time required to detect them and the time to take corrective actions.
4. There are no intermittent types of faults or failures. That is, the fault signatures do not appear randomly. This in itself is a very real and challenging problem, but the current framework does not address it.
5. The sensor data is reliable. The framework assumes that the input data it is getting from the sensors and its features is correct. It makes no effort to validate the data and is not equipped to handle sensor-related problems.
6. The operating mode of the UUT is assumed to be known and is part of each case in the library. Since the cases from the library are only considered if all the features mature for a particular class, it is assumed that there is no ambiguity about the mode.
7. The agent *knows* which other agent to communicate with during the coordination process. The “agent discovery” is itself a challenging problem but is outside the scope of the current research.

CHAPTER 2

DYNAMIC ACTIVE DIAGNOSTIC ARCHITECTURE

2.1 OVERVIEW

The objective of this thesis is to improve upon the current diagnostic and prognostic systems by incorporating new paradigms of software engineering and artificial intelligence. This thesis has two facets to its architecture. On one hand, it extends the conventional notions of a diagnostic framework as Dynamic Active Diagnostics, while on the other, it extends the implementation paradigm by incorporating intelligent software engineering techniques. This chapter presents the conventional architectural practices and their extensions.

2.2 BACKGROUND

The notion of Dynamic Active Diagnostics (DAD) is a major departure from the conventional diagnostic approaches that can be considered passive, open loop, and static. Conventional diagnostic systems are termed *passive* since they never initiate an action or event and only respond to events outside their process space. They are *open loop* since their primary goal is to detect and identify the failure according to a prescribed model, rule-base, or algorithm. These systems do not observe how accurate their diagnosis was. As a result, they remain *static*, as they do not modify their internal states. Since prognostic systems are basically designed to be extensions of their diagnostic counterparts, their architectures suffer from the same limitations.

In the literature, the term *Active Diagnosis* has been used in two contexts. In the first, the Active Diagnosis achieves a *closed loop* diagnosable system. In this thesis, this approach has been termed as “Design-time Active Diagnostics”. In the second context, the Active Diagnosis is achieved by continuously monitoring the state of the system *actively* – rather than waiting for an external event to occur. Since this approach was motivated by research on Immunity Based Agents, we designate it as “Immunity-Based Active Diagnosis”.

2.3 ACTIVE DIAGNOSIS

Active Diagnosis (AD) is a very new area of research with few published results. However, as mentioned above, there are two main thought patterns emerging. In this section these approaches are briefly reviewed.

2.3.1 Design-time Active Diagnosis

This notion of AD was introduced in the context of Discrete-Event Systems [Samapth *et al* 1998]. It extends the prior work of authors on the concept of *diagnosability* of a system [Samapth 1995]. For a DES model of the system that generates a regular prefix-closed language L , failures are the unobservable events (Valve_Stuck_Open etc) as opposed to observable events (such as controller commands Valve_Open, Pump_On, or sensor readings Flow, No_Flow etc.).

A language, and hence the system generating the language, is called diagnosable if failures belonging to a failure set can be uniquely identified from the trace within certain steps of time.

In the AD context, the authors in [Sampath *et al* 1998] have presented the problem of a system that is inherently not diagnosable. The AD is posed as the problem of designing a supervisor/controller such that it dynamically enables/disables certain events/states, so that the language generated by this controlled system L/S is diagnosable.

2.3.2 Immunity-Based Active Diagnosis

This concept of AD is motivated by the *self-identification* process of an immune system [Ishida 1996]. It is based on such technologies as agents, active databases, and active sensing. Each agent corresponds to a sensor and possesses knowledge of the instrumentation system. Also each sensor can identify a particular fault by comparing several sensor values, and can generate a reliability estimate of the identification. System level diagnosis is achieved by mutually activating or suppressing the evaluation of faulty nodes.

Following are some of the features of immunity based agents:

- Recognition is accomplished by distributed agents, which dynamically interact with each other in parallel.
- Agents carry redundant information.
- Agents react based only on their knowledge.
- Agents on the sensor networks can use their dependency information to evaluate other agents using these relations.
- Agents use a reliability measure to detect process level faults by sensor net.

2.3.3 Discussion

In large-scale systems a very important issue is the lack of data regarding the failures and faults in these systems. Large-scale systems are usually tested for faults and failures in conditions that are very different from the actual operational conditions. Moreover, in certain situations, information about all of the possible failures, operating conditions, system loads, and other variables is not available. Therefore, the utility of a Design-time AD system is questionable when not all events and states (observable or not) can be preconceived. The immunity-based AD approach is better, since it is biased towards monitoring for normal conditions, and reports failures in case of any deviation from the normal behavior. However, it is not very clear how the relations between agents will be established for the sensor networks.

Both of these approaches have no learning possibilities and their claim of depicting *dynamic* behavior is questionable.

A practical aspect is the requirement of system resources. In the Design-time AD approach, more effort is put in during design time and hence the *run-time* requirements are minimal. However, the Immunity-based AD approach requires a lot of memory and computing resources.

2.4 IN SEARCH OF AN INTELLIGENT DIAGNOSTIC SYSTEM

This thesis is motivated by the ability of human diagnosticians to make an assessment of the health of a machine and to take reasonable actions based upon observable events and measurements. An experienced diagnostician can also predict failures, often within reasonable margins of errors. The aim of this thesis is to propose a diagnostic framework based upon the human reasoning and learning mechanisms. The

obvious goal is that such a framework will be adaptable to unexpected scenarios and dynamic environments just like its human counterparts.

The research in cognitive science pioneered by Schank and Abelson suggested that the general knowledge about the situations is recorded in the brain as *scripts* [Watson 1997]. For example, auto mechanics faced with an unusual mechanical problem tend to recall other problems that were similar and consider whether their solutions explain the new one. Doctors evaluate the appropriateness of a therapeutic procedure based on their previous experiences [Kolodner 1993].

2.5 DYNAMIC ACTIVE DIAGNOSTICS: THEORETICAL FRAMEWORK

When the problem is obvious, a simple experience-based solution or a modification of it suffices. However, in uncertain situations humans not only rely on their experience but also coordinate and communicate to take advantage of experiences of others. This ability is very useful for human diagnosticians of large-scale systems who face problems that usually manifest themselves in inconsistent manner and also are not very frequent yet can cause significant damage.

With this background, this research is extending the problem of diagnostics to problems of learning, coordinated learning, and self-organization among a community of diagnostic systems. This new approach is termed as Dynamic Active Diagnosis (DAD).

The architecture presented in Figure 2.1 is a schematic diagram of a generic conventional (passive) diagnostic system. The diagnostic system applies the diagnostic algorithms and preset rule-bases on the observable events and system states to detect a problem. The diagnostic system may also isolate the location of the failure. Results are

used to either sound an alarm, display an alert and/or are simply stored in a database and used in prognosis.

The architecture presented in Figure 2.2 is a schematic diagram of the proposed Dynamic Active Diagnostic (DAD) system. The DAD system is decoupled from the rest of the plant and controller. The active diagnostic architecture closes the loop for the diagnostic system. Thus, theoretically, it should be possible for the diagnostic system to observe how well it is diagnosing (or prognosing) and hence improve its performance accordingly. This results in dynamic behavior.

Closing the loop for the diagnostic system opens up several interesting possibilities. Apart from the performance improvement it may also be desirable to reconfigure the system controls in order to bring the system to a safer operating mode in case a failure develops. This is a challenging problem in itself and does not befit the scope of the current thesis.

The *active* feature of the architecture is implied by its ability to coordinate diagnostic/prognostic activities with its peers, as shown by the network cloud in the figure, as well as by the fact that the process exhibits total control over its threads.

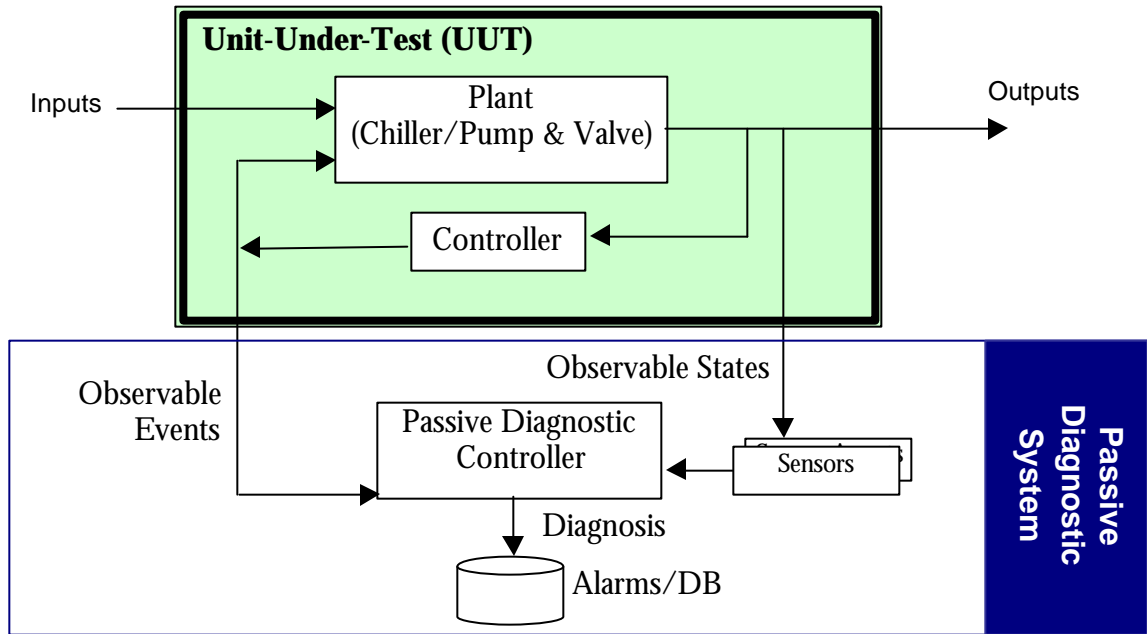


Figure 2.1 A UUT and a conventional passive diagnostic system

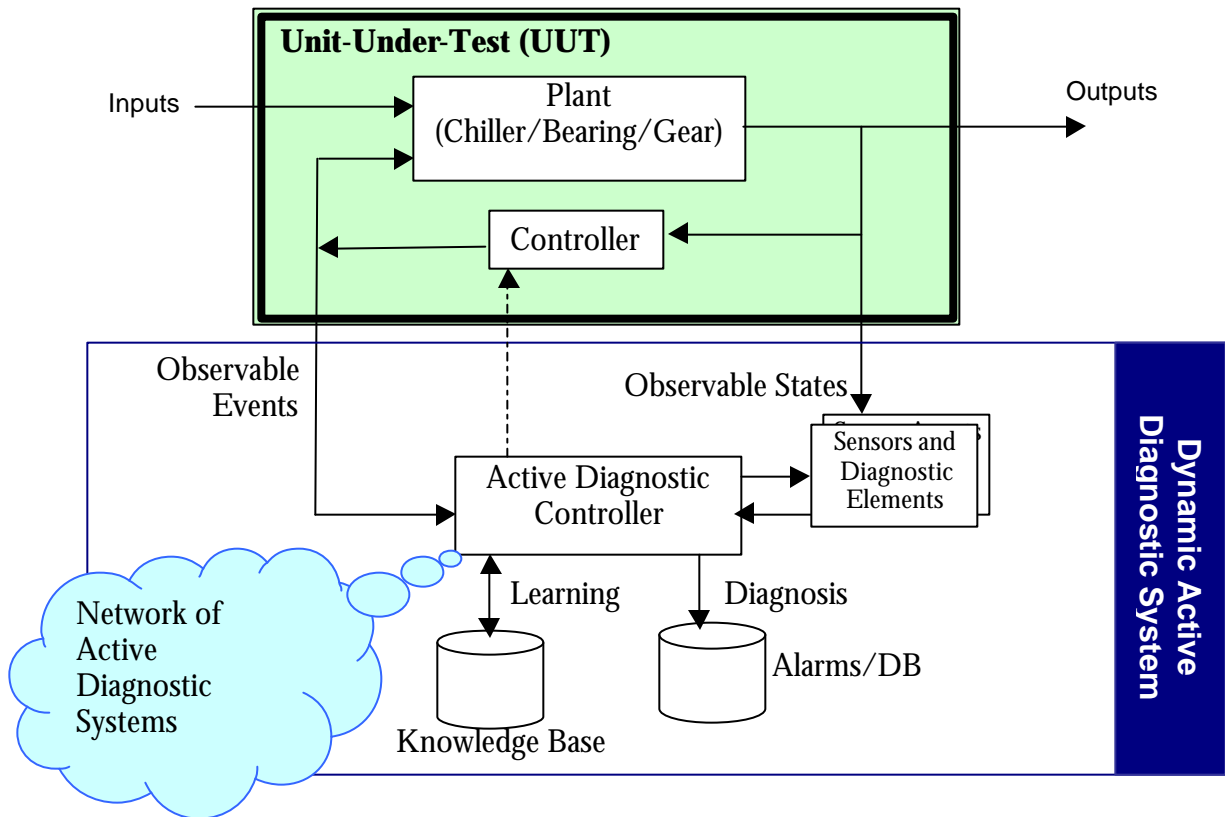


Figure 2.2 A UUT and its active diagnostic system

2.6 DYNAMIC ACTIVE DIAGNOSTICS: IMPLEMENTATION DETAILS

2.6.1 Intelligent Agents: A Natural Choice

Dynamic Active Diagnostics has been characterized as the paradigm that exhibits *active*, *dynamic* and *closed-loop* behavior. It will be shown that all of these attributes are well suited for an Intelligent Agent paradigm for software implementation.

An Intelligent Agent is defined as a software entity that is situated in an environment such that it can perceive as well as act upon it [Wooldridge *et al* 1995]. This forms a closed-loop behavior. In other words, intelligent agents exhibit *closed-loop* behavior by design. In fact, if an agent does not act on its environment, it is considered a regular software entity.

Given that an Intelligent Agent is a closed-loop system, the term *agent* necessitates that it possesses *autonomy* to achieve its goals. This sense of autonomy refers to the ability of an agent to act on its own (without human intervention) given a state of the environment. This autonomous behavior is considered to be a *necessary condition* for agenthood by most researchers [Franklin *et al* 1996]. Autonomy is also coupled with *proactiveness*. Thus, an autonomous agent is by design monitoring its environment and responding to it proactively. Therefore a software component that simply responds to external (environmental) events is not considered an agent. Therefore an Intelligent Agent is by design an *Active* entity.

There is no specific requirement for an agent to exhibit a dynamic behavior. Therefore, a *reactive agent* can have a rule-base that does not change over time. However, in most popular architectures, the Intelligent Agent *learns* from its experience. This learning is possible because the agent is cognizant of the environment as well as

effects of its actions on it. Therefore, an Intelligent Agent can be designed that exhibits *dynamic behavior* in that it learns from its experience over time.

2.6.2 Intelligent Agent Architecture for DAD Framework

A hierarchically organized distributed multiagent system is presented as a framework for implementation of DAD framework. Figure 2.3 shows an agent's block diagram along with related components. A community of these agents is expected to help each other by forming a Multi Agent Diagnostic System (MADS). Different components of the IA based architecture are described next.

Percepts

A feature extractor module takes the raw data from the sensors and applies signal analysis techniques to extract information from the raw data. This information is organized as features in a time-series fashion. The agent has an interface with a sensor/feature extractor module. In agent terminology, this interface is called *percept*. Therefore, an agent *perceives* its environment. The perception includes raw data, features, events, experiences, etc.

Software repository is a collection of software modules that are used by the diagnostic and prognostic agents according to the selection criteria. The agent can improve the diagnosis/prognosis by using the diagnostic and prognostic algorithms.

Reasoning & Learning

The architecture for reasoning and learning is combined in the Case-Based Reasoning (CBR) layer of the agent. This is motivated by the human-oriented approach taken for the design. The intuition of CBR is that the situations recur with some

regularity. The steps taken in one situation are likely to be applicable in a similar situation [Kolodner 1993].

In the diagnostic problem domain where the systems do not exhibit failures on a regular basis the appeal for a CBR is its ability to store knowledge in small *experience capsules* that can be shared so that community learning can take place. In this way CBR combines reasoning and learning. Although CBR is not the first method that makes this combination possible, it is unique in making learning a byproduct of reasoning.

The CBR module of the agent consists of several smaller modules that include the Retrieving, Adaptation, Testing, and Learning modules. The reasoning process starts by *retrieving* the cases from the case library that can be considered close to the current situation. The reasoning aspect of CBR is an attempt to find out if any existing experiences are applicable to the current situation at hand. If no case fits very well, it may be possible to modify some close experiences. This *adaptation* mechanism generates new cases that are tested to confirm that they are indeed applicable and are stored in the case library for future use. The CBR implementation for the diagnostic domain is somewhat different from the conventional CBR systems in that the *experience* represents not only the current situation but also *predicts* future progression of these situations. Details of this architecture are given in Chapter 3.

Coordination/Collaboration and Related Activities

In the MADS Architecture each agent can initiate a communication with its peers. Thus each individual agent is *coordinating* with others for a solution to its current problem. The peer agents on the other hand are said to be *collaborating* with this agent to help out in the search for a solution. This way the agents communicate to solve problems

that are not solvable by a single agent. This is the spirit of the distributed AI (DAI) research in Multiagent systems. The CBR architecture makes it possible for the agents to communicate their individual experiences.

Decision Support & Prescription

The actions taken by the agent architecture include but are not limited to the online diagnosis and prognosis that is meant for a maintainer. Since the system is also elaborate in its data collection, it can also help feed a decision support (DSS) layer that can be used by the system designers to modify the features, design, model, algorithms, etc. Some of the basic modules of a DSS are implemented in this research but the decision-support system is not designed but is a challenging research possibility in itself.

Other actions, such as reconfiguring system controls, are also possible however are outside the scope of this thesis.

2.7 INTELLIGENT DIAGNOSTIC AGENT ARCHITECTURE

Intelligent Agent architectures can be divided in four major classes [Wooldridge 1999]. Reactive Agent Architecture is simple to implement. It is also more robust and computationally tractable]. A well-known architecture in this category is *Subsumption Architecture* of Rodney Brooks [Brooks 1986]. Generally, the reactive architecture does

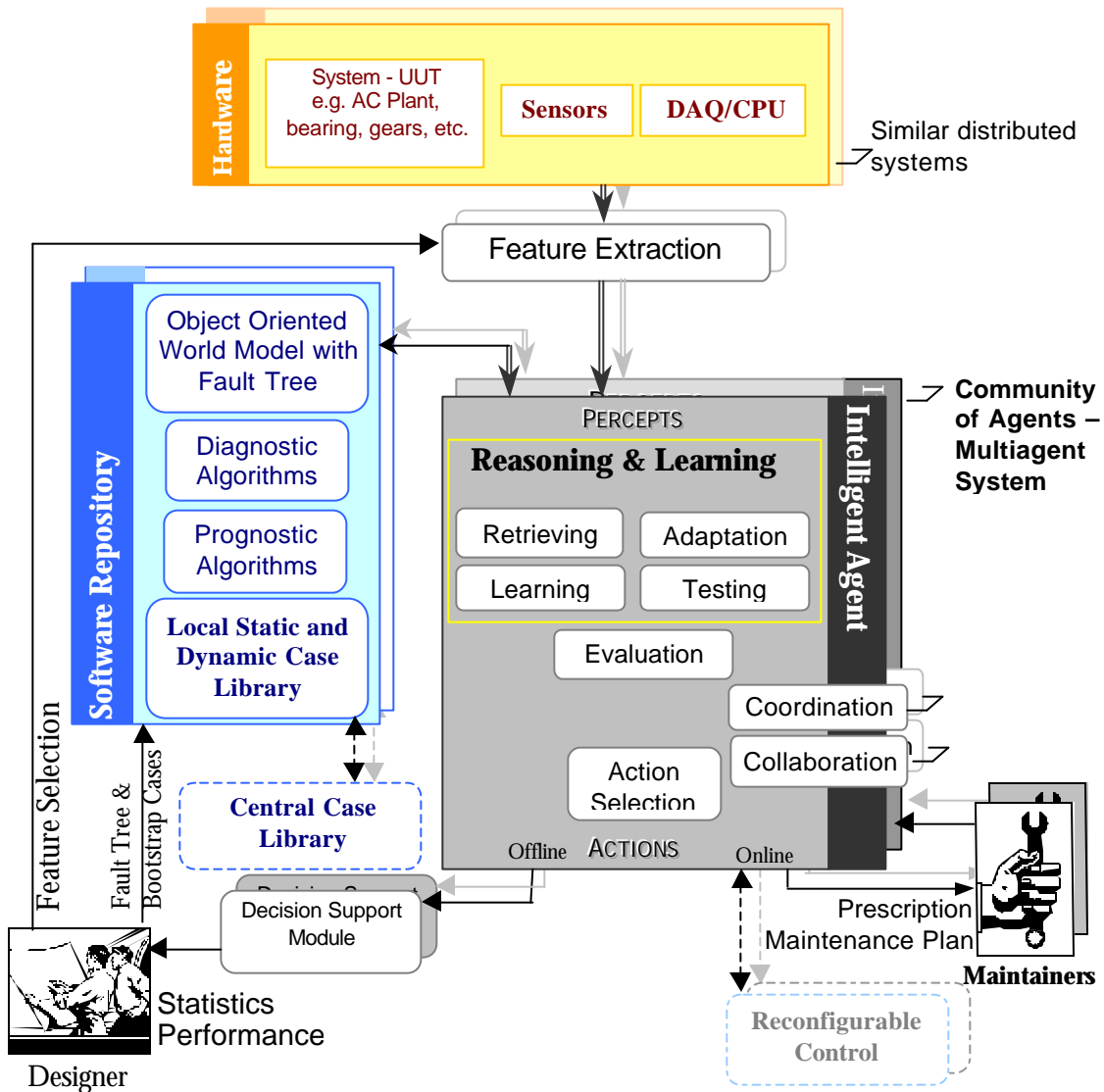


Figure 2.3 Implementation of DAD framework by an IA and related modules

not include any complex symbolic representations and no symbolic reasoning is applied. In many implementations, the behaviors are implemented by simple rules of the form

Situation ® *Action*

This represents a simple map between perceptual input to actions.

Intelligent Agents relying upon a logic-based architecture follow the traditional approach to building artificially intelligent systems, namely symbolic AI. Thus, intelligent behavior is generated by giving that system a *symbolic* representation of its environment and its desired behavior, and syntactically manipulating this representation. Thus agents act as *theorem provers*. METATEM and Concurrent METATEM architectures are good candidates for these reasoning agents [Barringer *et al* 1989].

The BDI architecture has its roots in *practical reasoning* – the process of deciding, moment by moment, which action to perform in the furtherance of our goals. Planning is more goal-oriented behavior and is suitable for the BDI agents. The practical reasoning is concerned about two processes: deciding *what* goals should be achieved, and *how* to achieve these goals. Each BDI agent has a sophisticated reasoning architecture that consists of different modules that operate asynchronously. Starting from the set of *beliefs*, representing the information about the environment, the agent generates options by *options generation function*. A filter function represents the agent's *deliberation* process, which determines the agent's intentions, based on its current beliefs, desires, and intentions. The *action selection function* determines an action to perform based on current intentions [Wooldridge 1999].

In a Layered Architecture, decision making is realized via various software layers, each of which is more-or-less explicitly reasoning about the environment at different levels of abstraction.

For the diagnostic framework an Intelligent Agent module is designed as a hybrid between a horizontal and a vertical layered architecture. A schematic diagram of the

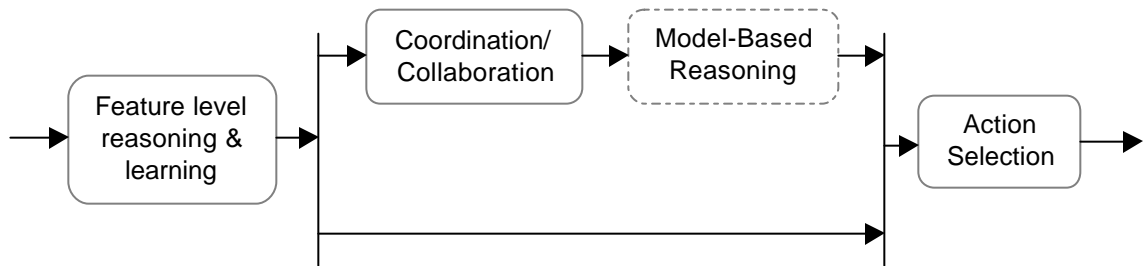


Figure 2.4 Schematic diagram of a diagnostic agent.

architecture is shown in Figure 2.4. This architecture defines a number of “levels of competence” for an autonomous agent. A level of competence is an informal specification of a desired class of behaviors for an agent over all possible environments it will encounter. A higher level of competence implies a more specific desired class of behaviors. Each level of competence includes, as a subset, earlier levels of competence. Thus, level 1 control can examine data from the level 0 system and is also permitted to inject data into the internal interfaces of level 0 suppressing the normal data flow. Layer 0 continues to run unaware of the layer above it, which sometimes interferes with its data paths. The same process is repeated to achieve higher levels of competence.

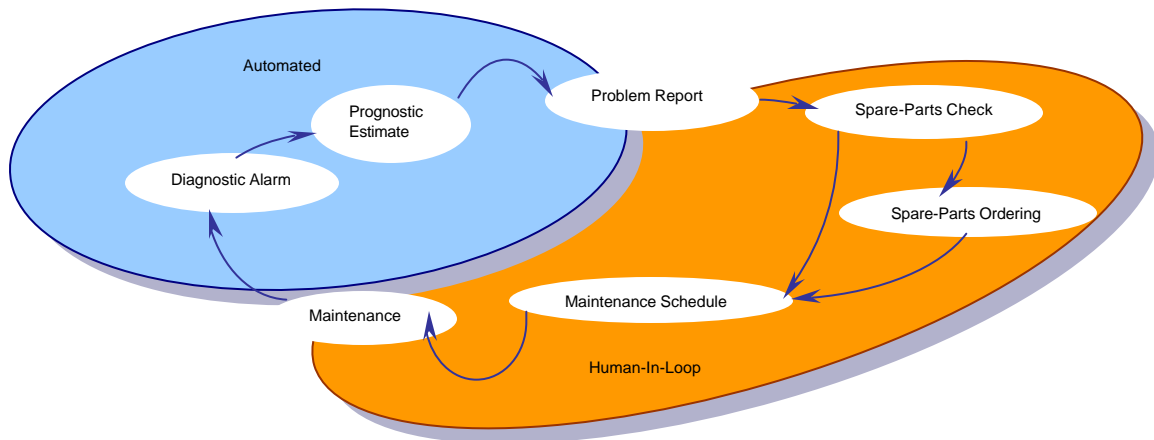


Figure 2.5 A typical diagnostic and prognostic framework with related activities

2.8 GLOBAL VIEW

2.8.1 A Typical Scenario

A typical diagnostic and prognostic framework with “human in the loop” works as shown in the Figure 2.5. A diagnostic alarm produces an RUL estimate. The human in the loop, for example, a technician, reports the problem and schedules maintenance work. A related activity is spare-part inventory check up and ordering of the required spare-parts.

2.8.2 Proposed Scenario

The Multiagent framework can autonomously facilitate decision support systems at multiple levels of hierarchy as well. The decision support systems can interact with inventory control systems, at several levels of domain. A global perspective is presented in Figure 2.6.

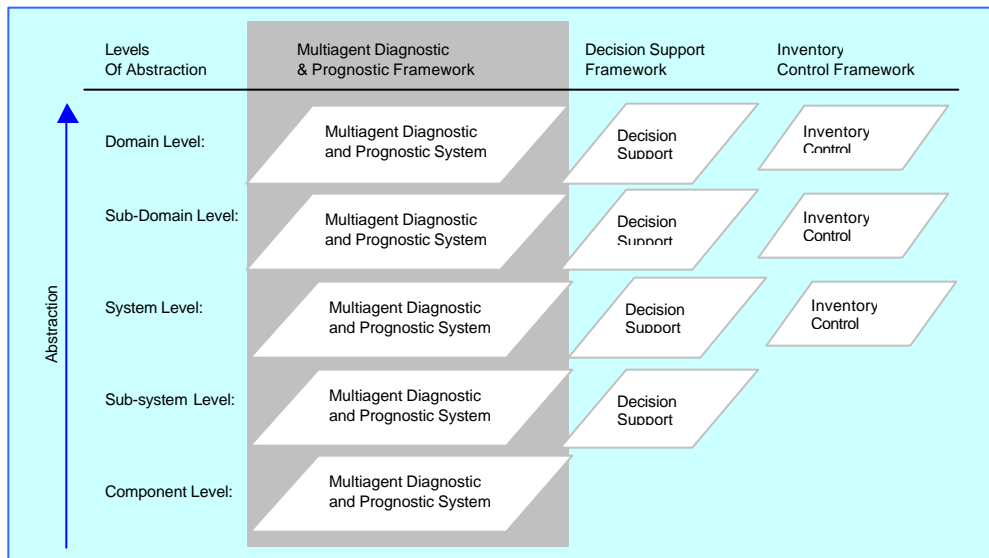


Figure 2.6 Global view of multiagent diagnostic and prognostic framework

2.9 CONCLUSION

We redefine the design of a diagnostic and prognostic system by incorporating AI methodologies of learning, reasoning, and coordination. Theoretically we extend the responsibilities of the current diagnostic system. The new framework should not only diagnose and prognose, but with the collaboration of multiple intelligent agents, can also perform decision-support and inventory control tasks. Obviously these tasks require human intervention at some point. However, as we understand more about the systems and their failure patterns, the framework allows for automation of these regular activities.

In this research we focus on the learning, self-organization, and collaboration of the framework from the diagnostic and prognostic perspective.

CHAPTER 3

DYNAMIC DIAGNOSTICS

3.1 INTRODUCTION

Some of the most challenging problems in designing a diagnostic and prognostic system include determining the right levels of signals and their features at which a diagnosis or prognosis can be generated, and variability of failure mechanism. These problems are compounded for large-scale systems because of the absence of statistical data associated with the critical failures.

For an intelligent diagnostic system, therefore, the challenge is two folds: to figure out if the alarms/thresholds associated with a failure are correct, and if they are not then how to learn the correct values.

In a conventional diagnostic system, alarms and threshold values are obtained from experience or from preliminary testing. Once these values are set in the system's rule-base, the system will not make any effort to modify these values, even if they turn out to be totally incorrect. Such a system will perform only as good as initial selection of these values, and hence is termed as *static diagnostic system*. This system is a sequential machine whose control is designed to be rigid and works well only in domains where the failures happen in a consistent manner and the domain model is well understood.

A diagnostic system that is designed to have a flexible control and will work according to what it learned previously from mistakes as well as correct actions is called

dynamic diagnostic system. This *dynamic* behavior is only possible if the agent has the capability to learn from the history of failures, and can also reason about best possible actions at a given time. Therefore the proposed *dynamic* active diagnostic agent is not only an academic curiosity but also a solution to a challenging real-world problem.

From the CBM perspective the estimation of remaining-useful lifetime (RULT) for a system is critical. In the absence of sufficient data it is also very desirable that an intelligent system can track the progression of the failures and correct itself accordingly and generate predictions that have less uncertainty about them.

In this chapter we present a methodology of implementing dynamic aspects of diagnostics as a learning and reasoning framework, namely Case-Based Reasoning.

3.2 DIAGNOSTIC REASONING & LEARNING: A HISTORICAL NOTE

In diagnostics and prognostics, learning and reasoning architectures have historically been of the rule-based expert system variety. Case-Based Reasoning (CBR) systems have been successfully employed for diagnostics where *experiences* of diagnostics are encoded in case structures. Model-Based Reasoning (MBR) systems are newer and rely on organization of knowledge in models. At signal's level, artificial neural networks (ANN) have been applied for classification tasks as well as for prognostics [Noppadon 2001].

In this thesis we present a novel CBR methodology that combines reasoning and learning to detect incipient faults. The cases track the progressions of these faults to generate not only diagnosis but also a prognosis of the system, as a result of which the agent improves its performance with experience.

3.3 WHY CBR?

CBR systems have outperformed traditional expert systems and model-based reasoning systems [Kolodner 1993]. It has been shown that these systems are simpler to design because their knowledge engineering is simpler to implement.

One of the main criticisms of CBR is that it does not fully explore its solution space. Therefore problems that require optimal solutions are not suited for CBR implementation. Generally this is true for all the heuristic based systems and CBR is not an exception. Another disadvantage is that CBR requires a large amount of memory to hold its cases.

In the current research we chose CBR because of its expressiveness that is very useful for sharing experiences and because the reasoning and learning style of CBR is intuitive and closer to a human diagnostician's episodic approach.

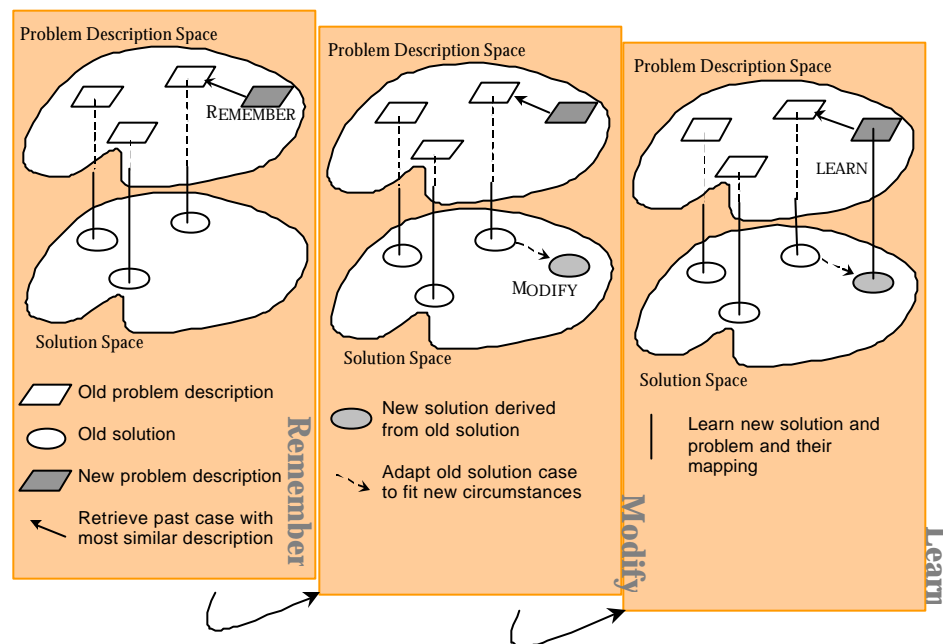


Figure 3.1 Basic case-base reasoning cycle

3.4 CASE-BASED REASONING PROCESS

The CBR process can be described as “remember + modify + learn” cycle. This flow is shown in Figure 3.1.

Figure 3.2 shows a conceptual diagram of the main blocks of a CBR system. The reasoning starts with a new problem query. Best-matched cases are selected from the case library, and are modified to create a new solution. The new solution is tested and is stored into the case library as a new experience.

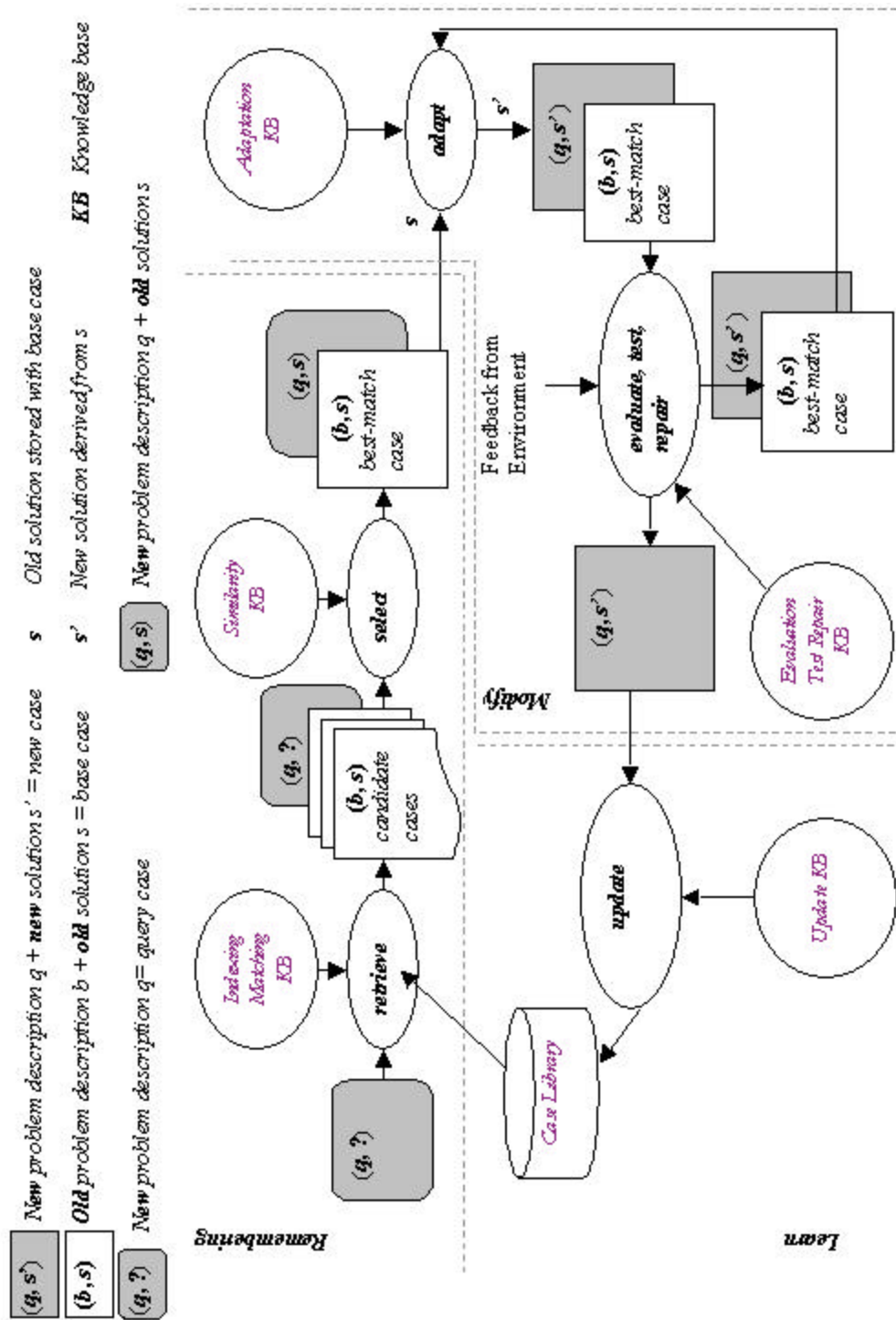


Figure 3.2 CBR Main Processing Blocks [Dubitzky 1997]

To achieve dynamic diagnostic behavior several factors need to be considered. Starting from the knowledge elicitation and organization, to the searching and selection process, to the modification and learning experiences, each is a challenging design activity. Figure 3.3 shows the methodology used in this research. It is organized as a sequence of problems that must be solved to generate a framework that can exhibit dynamic behavior. We will discuss each of these steps in detail in the rest of this chapter.

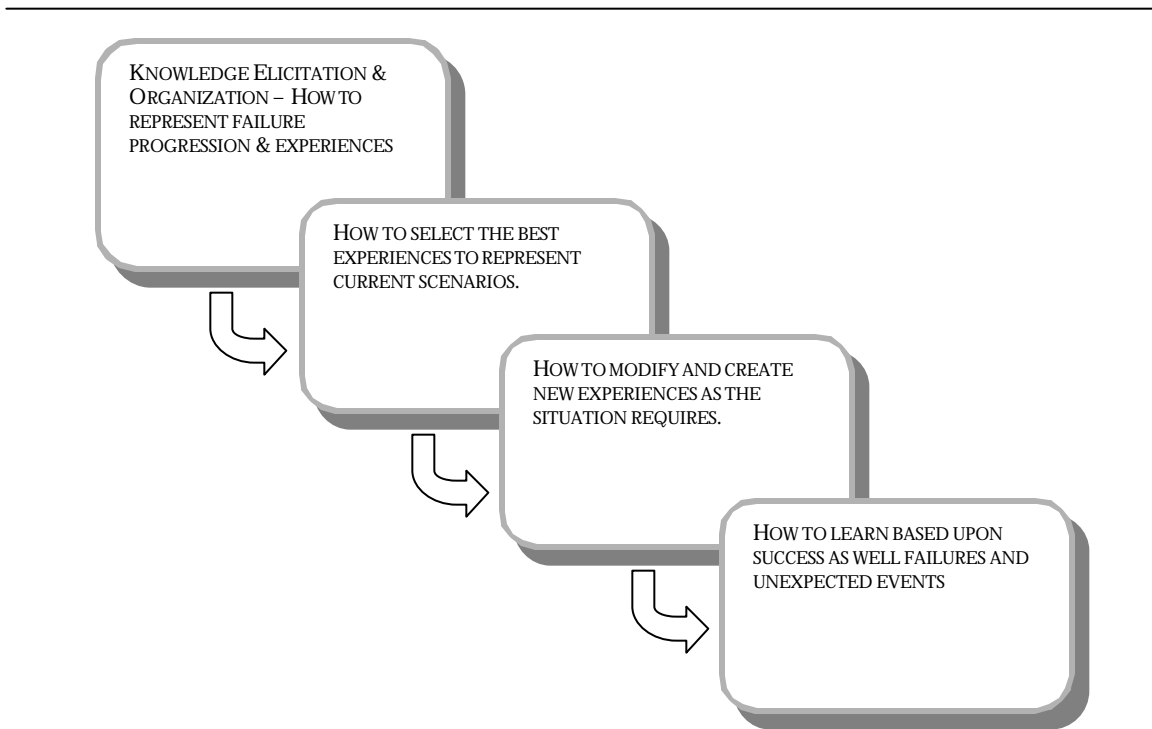


Figure 3.3 Design methodology of a dynamic diagnostic system

3.6 IMPLEMENTATION APPROACH

3.6.1 Knowledge Elicitation & Organization

The experience of a diagnostician can be very complicated and in some cases the knowledge elicitation is impossible. This historically has been a major problem for expert systems. However, since we are interested in finding out about only selected episodes of failures, knowledge acquisition from the diagnosticians and their logs becomes simpler.

Assuming that some decent knowledge is available, the next problem in the methodology is its organization. The goal is to keep the framework simple enough so that its memory and processing requirements are minimal, while at the same time the organization should be expressive so that it can be shared between the systems as well as between the system and the user. This section deals with the issues of representing diagnostic experiences.

Case Structure

A *Case* is a knowledge capsule for CBR style learning. It also represents the basic structure for reasoning. Usually, a case represents a concrete and specific event, object, situation, episode or experience a reasoner has encountered in the past [Dubitzky 1997]. It is described by three constituent parts: *situation/problem description*, *solution*, and *outcome* [Kolodner 1993]. The *situation/problem description* is information related to an event or episode that occurred in the past. The *solution* is the answer to the problem or question specified in the situation/problem description. To ensure that the system generates good solutions the *outcome* component provides a validation mechanism.

<i>(a) Case:</i>	
Title/Description	_____
<i>Problem Description</i>	
Symptom 1:	_____ is _____
Symptom 2:	_____ is _____
...	
Symptom N:	_____ is _____
<i>Solution</i>	
Diagnosis:	_____ Prognosis: _____
Outcome	Evaluation: _____
<i>(b) Case:</i>	
Case Title:	Computer I/O Port is setup incorrectly
Case Description:	Computer doesn't print, self-test is OK
<i>Problem Description:</i>	
Printer prints a self-test	= Yes
Display Message	= 03 IO Prob
Printed Configuration Correct	= True
IO Port works with other devices	= No
<i>Action:</i>	
See software application notes or computer manuals to set up I/O Port correctly.	

Figure 3.4 (a) A general case template used in diagnosis/prognosis; (b) Example of a case designed for a CBR for printer diagnostics [Watson 1997]

Figure 3.4 (a) shows a typical case structure used in conventional CBR systems for diagnostics.

Example

Figure 3.4 (b) shows an example of a case taken from Watson's [Watson 1997] implementation of a diagnostic CBR. This CBR was designed to diagnose problems related to printing. Since this CBR was intended for either users of the printers or for the technical support crew, the contents of the case are very descriptive. The title and case description are the gist of the information embedded in the case. The problem description entries specify the issue in more details. As can be seen, this case will be selected by the CBR only if all the problem description entries on the right-hand side match a given situation. This case has a solution entry as an *Action* statement.

Similar to the example given above, the applications of CBR for diagnosis [Derere 2000, Althoff *et al* 1995] and prognosis [Armengol *et al* 2000] have used a *snapshot* vision for defining a case. A typical CBR implementation will have a set of symptoms observed at any given time as the *situation/problem description*, and a set of possible diagnoses as the solution.

Our Approach - Designing to Represent Progression

In this research diagnosis is not a one time snapshot. Since the diagnostic system looks for an early detection of an incipient fault, these conditions need to be monitored closely through the progression of the failure. We use these guidelines for the design of the basic case structure as well as its reasoning and learning activities. The new case structure is shown in Figure 3.5. As can be observed, this is not only a snapshot *in* time, but also is a snapshot *of* time. The former gives us the diagnosis, while the later provides the prognosis. This approach is in some ways similar to the CBR implementations for time-series reasoning, such as [Riordan *et al* 2002] who used temporal dimensions explicitly in their case definition for weather prediction, although they use the case for instantaneous reasoning. The terminology used in our implementation is as follows:

Symptom - Sensor data or its feature. The sensor data and the features of the data.

T_e - *Early detection threshold*. For a possible candidate for an incipient failure.

T_v - *Validation Threshold*. For an imminent failure.

T_h - *Hazard Threshold*. is the maximum level at which a system can be operated.

T_r - *Repair Threshold* to identify when repair has been performed after a failure

t_{EV} - *Expected time between early detection and validation*.

t_{VH} - *Expected time between validation and hazard*.

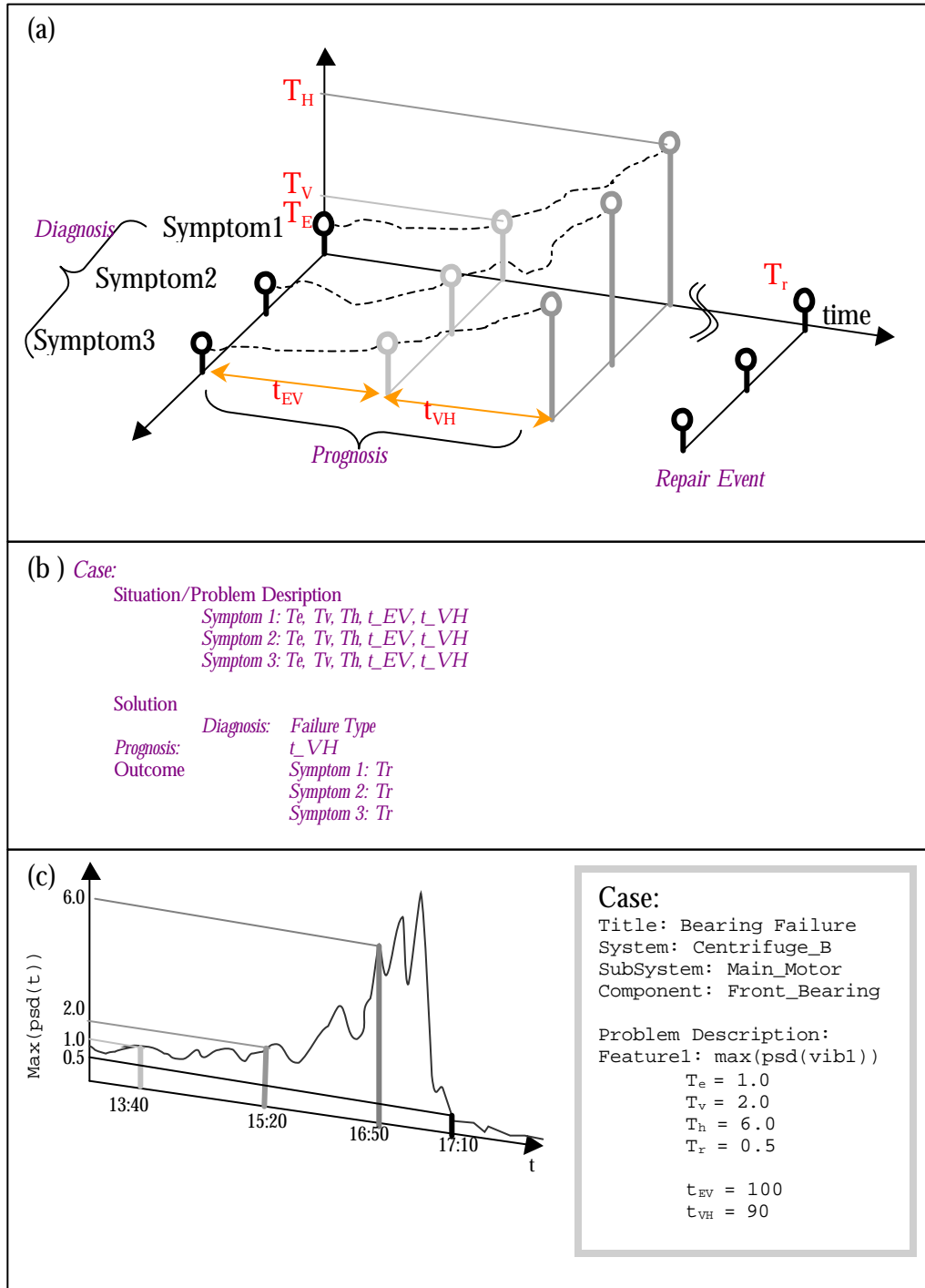


Figure 3.5 (a) A Case structure for the DAD case library (b) Template of a generic case for diagnosis and prognosis. (c) Example of an experience of a bearing failure being translated in a case structure

Example Case

Figure 3.5 (c) shows a case designed for bearing failure diagnosis and prognosis. The bearing was monitored for vibration by an accelerometer. The peak of the power-spectral density (psd) of the vibration signal at a given time is used as a feature. The hazard value of 6.0 is a conservative estimate of the limit of useful life and is established by observation. The bearing failure is a problem of choice for prognostics, since it usually progresses in a way that can be monitored as well as predicted.

As shown in the Figure 3.5(c), the feature initially grows at a lower rate. After validation, the feature grows rapidly. In the case shown the bearing failure results in a big spike. The failure stops the machine and hence the vibration signature is pulled down to almost zero. The monitoring agent needs to record this event so it can reset its reasoning.

Ideally the system should have been stopped before the vibration grew to the hazard value. Therefore in a variation of the above scenario, the vibration will never reach the hazard threshold since the system will be shut down. That will result in the vibration signal dropping below the repair threshold causing the agent to reset its open cases.

At a later instant if the feature grows in the given fashion again, in both the above scenarios, the case will get a 100% success factor, since it represented the new failures religiously.

Please note that the case can be seen as a rough template of this type of failure can progress. The variations in the signal, as it swings up and down, are totally ignored. As long as the signal reaches its milestones in the expected time, the given case gets a 100% success factor score.

3.6.2 Methodology for Selecting Best Experiences

Remembering or *retrieval* involves a sequence of steps that are taken to select the past episodes that are closest to the situation at hand. Figure 3.6 shows the flow of a general retrieval process in CBR.

In the current research we grouped failure experiences in the case-library for easier retrieval. We also add an extra step of labeling a case's state after the best-match cases are obtained.

Cases are defined to be of three types: Normal, Failure, or False Alarm. These

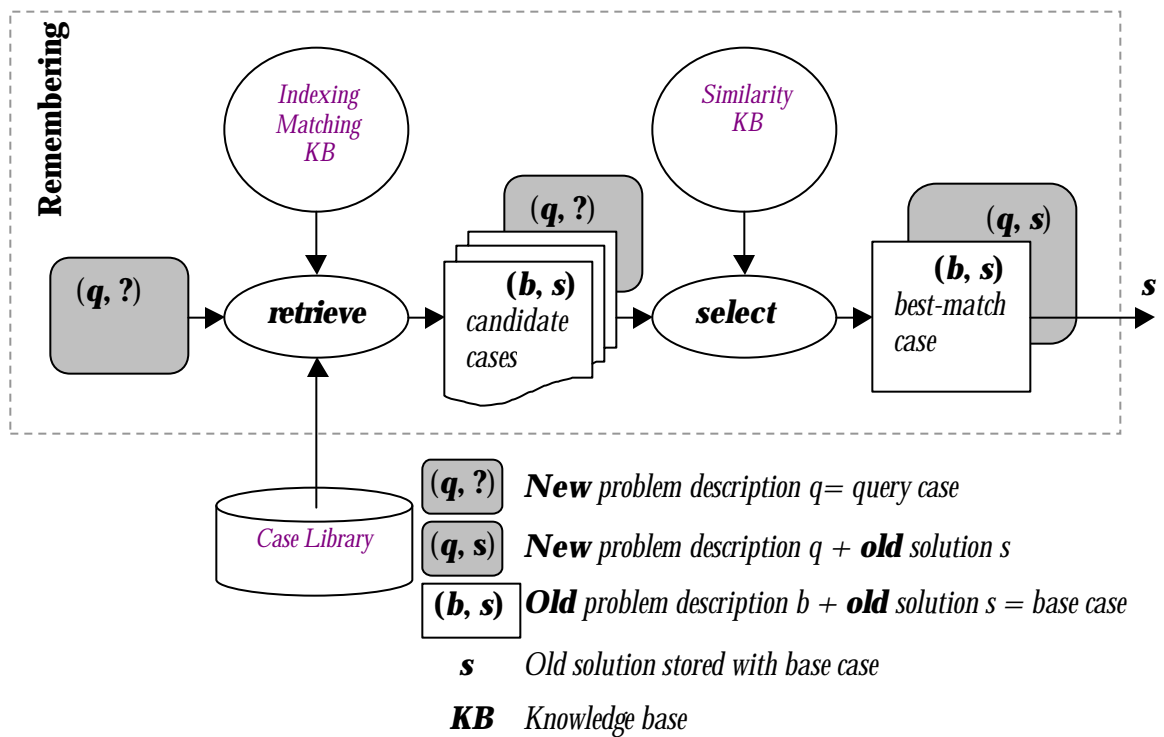


Figure 3.6 Remembering Process in CBR [Dubitzky 1997]

types represent respectively, experiences related to the normal (failure free) operation of the system, behavior of the system under failure conditions, and how false alarms have been generated.

Generating Candidate Cases

For each feature, time-series data is requested from the feature extractor according to the requirements of the failure mode. Each of the feature's current value is tested against each case's progression entries.

Let $f(t) = (f_0(t), f_1(t), \dots, f_m(t))$ represent a set of m features observed at any given time t . Let $T_e^k(j), T_v^k(j),$ and $T_h^k(j)$ represent *early-detection*, *validation*, and *hazard* thresholds, respectively for k -th feature of case j . Let $CS(t)$ be the *case state* at t , which can be either ED , V , or H . The state represents whether the case has crossed the early-detection phase, validation phase, or hazard phase respectively. Then a case j in the case-library will be selected as a *candidate case* if the following condition holds for a given operator \otimes :

$$\begin{aligned}
& [(f_0(t) \otimes T_e^0(j)) \wedge (f_1(t) \otimes T_e^1(j)) \wedge \dots \wedge (f_m(t) \otimes T_e^m(j))] \vee \\
& [(f_0(t) \otimes T_v^0(j)) \wedge (f_1(t) \otimes T_v^1(j)) \wedge \dots \wedge (f_m(t) \otimes T_v^m(j)) \wedge (CS(t) = ED)] \vee \quad (3.1) \\
& [(f_0(t) \otimes T_h^0(j)) \wedge (f_1(t) \otimes T_h^1(j)) \wedge \dots \wedge (f_m(t) \otimes T_h^m(j)) \wedge (CS(t) = V)]
\end{aligned}$$

The operator can be, for example, \geq for features that tend to grow. It can also be a fuzzy similarity operator, in which an aggregation operator will be required. In the current implementation, we have assumed a growing feature scenario and hence the operator is \geq . A generic application will use different operators for different features.

Open and Closed Cases

In our approach a case is *closed* by default and its state is labeled as 'Waiting for ED'. It is *opened* if it meets the first requirement (crosses ED threshold). This case remains open in the subsequent retrieval cycles, as long as it represents current input data faithfully. If it fails to represent current data, the case is marked to have *failed* and its *failure experience* is noted which is used to create a new case if required. This is discussed in the next section.

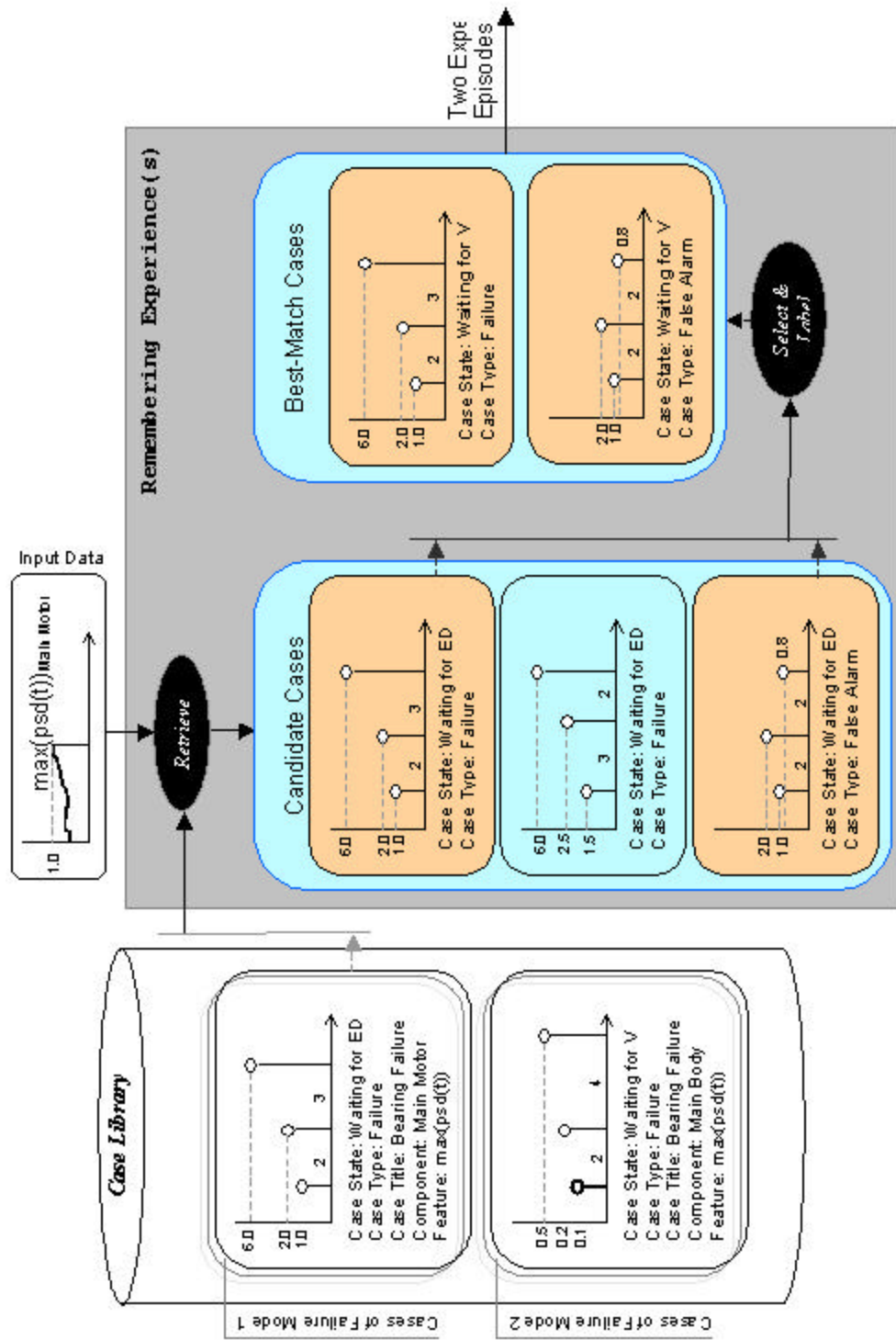


Figure 3.7 Example of remembering process

Example

Figure 3.7 shows an example of *remembering* process. The case-library is populated with closed cases that represent the following two types of failure modes:

Failure Mode 1: Source: Bearing Failure
 Component: Main Motor
 Feature Used: Peak of power-spectral density

Failure Mode 2: Source: Bearing Failure
 Component: Main Body
 Feature Used: Peak of power-spectral density

The current input feature data is obtained from the *Main Motor*. The retrieval process lines up all the candidate cases that belong to the failure mode of the Main Motor.

A closer examination of the candidate cases reveal that two of the three cases represent current data, since the Early Detection threshold is 1.0 in both cases, although one episode represents a failure, while the other shows a false-alarm. The third case is not selected since the ED threshold is higher than current value of the feature data. Case selection process therefore selects case 1 and 3. The *labeling* process changes their state from *Waiting for ED* to *Waiting for V*.

Example above shows that remembering process can result in cases that represent conflicting experiences, as one of the experience episodes is warning of a failure, while the other case is telling the diagnostic agent that this is just a false alarm. At this point both these experiences are valid. We will see later how we select one of these based on our past experience of their usage.

3.6.3 Methodology for Modifying an Old Experience – Adaptation

A recorded episode is never an exact match for the new situation. In our methodology the old solution must be modified to suit the new situation. An experience of *failure* or of a *false alarm* is valuable as it provides a blueprint of how systems fail (or pretend to fail in the case of false alarms). However, as a clever human diagnostician will point out, it is highly likely that the same failure will generate slightly different signature, next time around. The value of an old experience therefore is in providing a ballpark template of failure progression for a prognostic reasoning system. Among the several mechanisms documented for adaptation we employed *Parameter Adjustment - Substitution Method* [Kolodner 1993].

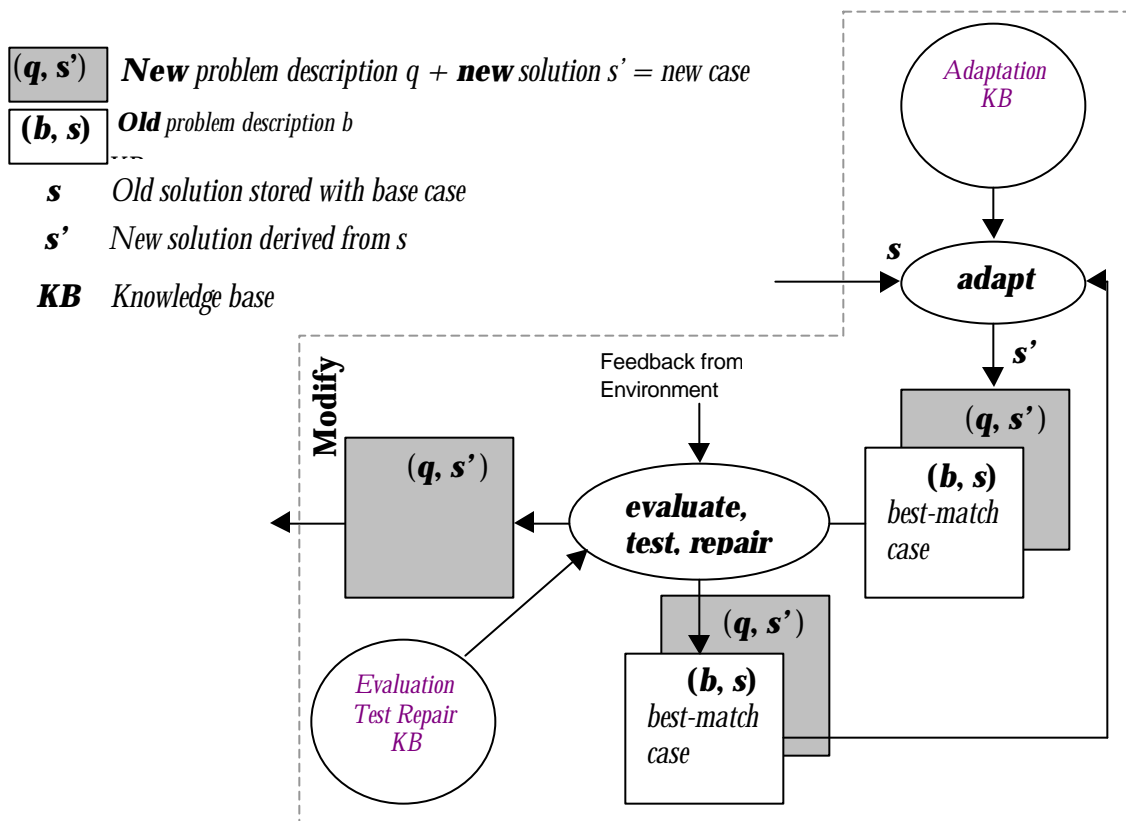


Figure 3.8 Adapt

An open case that fully represents current situation will require no adaptation. In CBR terminology it is called *null adaptation*. For the other types of the cases we divide the adaptation strategy in two groups: *Adaptation Based On Model (ABM)* and *Adaptation Based On Experience (ABE)*.

In *Coordinated Learning* experience, a full case is *transplanted* into the working memory with few modifications. This is an ABM style adaptation and the agent uses a model for modification in order to create new experiences. Details related to this type of learning are given in Chapter 5.

In ABE style adaptation, cases are modified in two ways. In *Failure Adaptation* the agent modifies experiences to reflect different ways a failure or false alarm can happen. In *Failure of Failure Adaptation* the agent learns about how a predicted failure resulted in false alarm or unexpected failure. This type of learning generates cases of the *false alarm* variety.

Figure 3.8 details basic steps taken in the adaptation process. The best case(s) s are obtained from the retrieval process discussed earlier. The output of the adaptation process is a new case with new problem description q and new solution s' . After a new solution is obtained, it is tested and evaluated to ensure that poor solutions are not repeated and to detect potential problems in it. The evaluation process is very simple and pragmatic – just apply the new experience to the current data and see if it faithfully represents current data.

Failure Experiences

All the experience episodes that are retrieved from memory are continuously monitored to find if they are still representing current data. If there is a discrepancy

found, the open case is marked with a *failure experience*, and is closed. There are four types of failure experiences defined: *rapidly developing*, *slowly developing*, *unexpected*, and *false alarms*.

Each of these experiences is prefixed with the temporal milestone the case had successfully passed before getting closed. For example, a case that was opened and validated, but did not generate a failure in the expected time frame is labeled as V_SDF for *validated but slowly developing failure*. Other possibilities are described in Figure 3.9. A *null adaptation* experience is termed as ‘As Expected’.

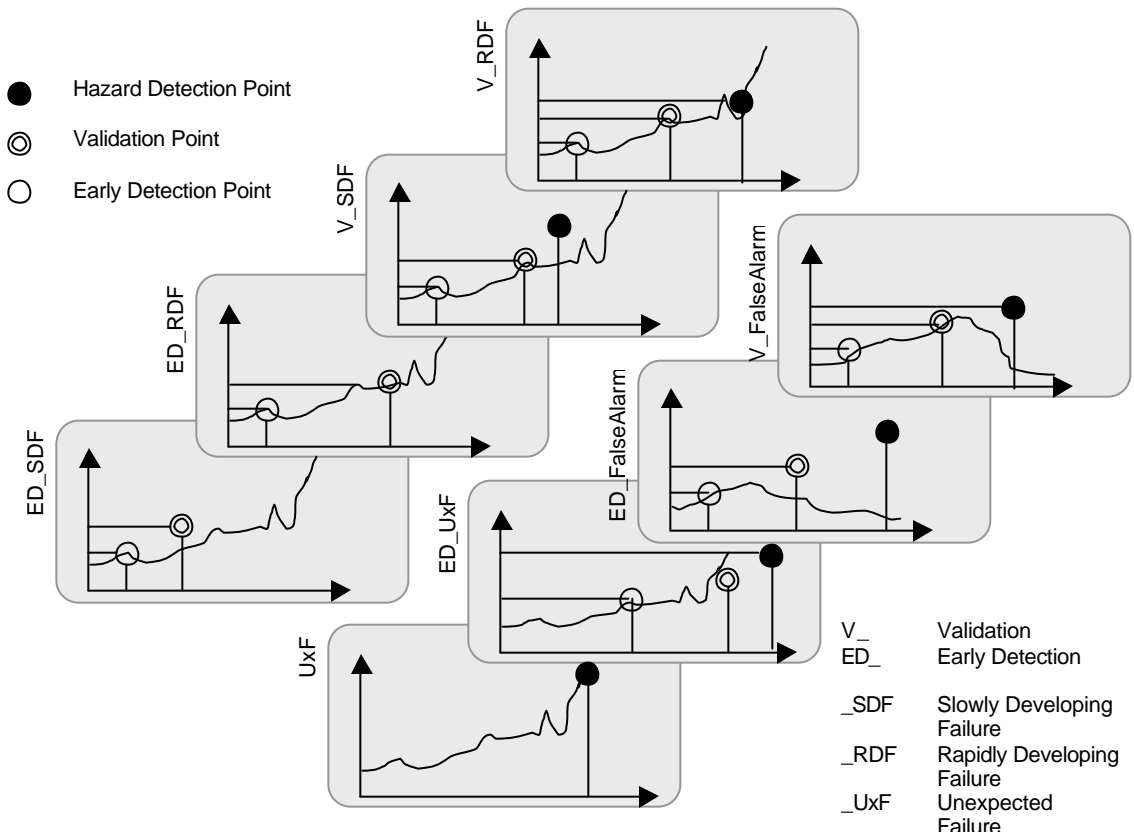


Figure 3.9 Failure experiences of different cases

Adaptation Based on Experience (ABE)

The algorithms for ABE style adaptation are aimed at modifying the best previous experiences to reflect the current data. The modification process may entail changing the temporal variables or the thresholds. These modifications are applied to an existing episode depending upon its *failure experience* as defined above, to generate a new case.

Table 3.1 details the modification process for the failure cases with different failure experiences.

Table 3.1 Adaptation process for different failure experiences

Case Type	Failure Experience	Modification Action	Comments
Failure	V_RDF	Modify t_{VH}	Failure already happened... so now just learn the new pattern
	V_SDF	Mark t_{VH} and learn	If a failure is not allowed to progress, by a repair event, the best the case can represent is that the time between validation and hazard is at least t_{EV} .
	ED_RDF	Modify t_{EV}	Failure is validated in a new time window t_{EV} .
	ED_SDF	Modify t_{EV}	The new t_{EV} will reflect how slowly the given feature will progress.
	UxF	Modify (t_{EV} , t_{VH} , T_e , T_v)	Learn all the variables from the available data in the buffer.
	ED_UxF		Perhaps the validation point was too high, or the time between the ED and V was too large.
	ED_FalseAlarm	Learn New Case	Failure was signaled by ED but is not validated in t_{EV}
	V_FalseAlarm	Learn New Case	Failure was flagged by V but is not validated in t_{VH}

Example

Figure 3.10 shows an example of modification. This example is an extension of the previous example that was shown in Figure 3.7.

The two cases retrieved from the case library now include one of the case that was not selected earlier. This case has a higher ED threshold and hence is *opened* after the feature crossed 1.5 value.

In the first episode, it was expected that the feature would cross the validation threshold in 2 hours. However, as the data crossed the threshold in 1 hour, the old case *failed* to represent the current data. Modifying the old temporal variable generates a new experience episode. This new episode is shown in the *adapted cases*' box.

Although the old case failed to represent the new data, it does not mean the old case should be removed from the library, since the old case represents one way the feature can progress, while the new case represents another way the feature can grow to failure.

Even though we know the failure is progressing rapidly (hence the RDF label), we cannot say much about the total expected failure, except that since the failure is progressing rapidly, a total failure should be expected in less time than was predicted by the old case. This is depicted by the *qualifier* entry in the new case.

The second case is still in infancy, in that it has not crossed the validation threshold. Hence, it is monitored, but is not considered by the agent.

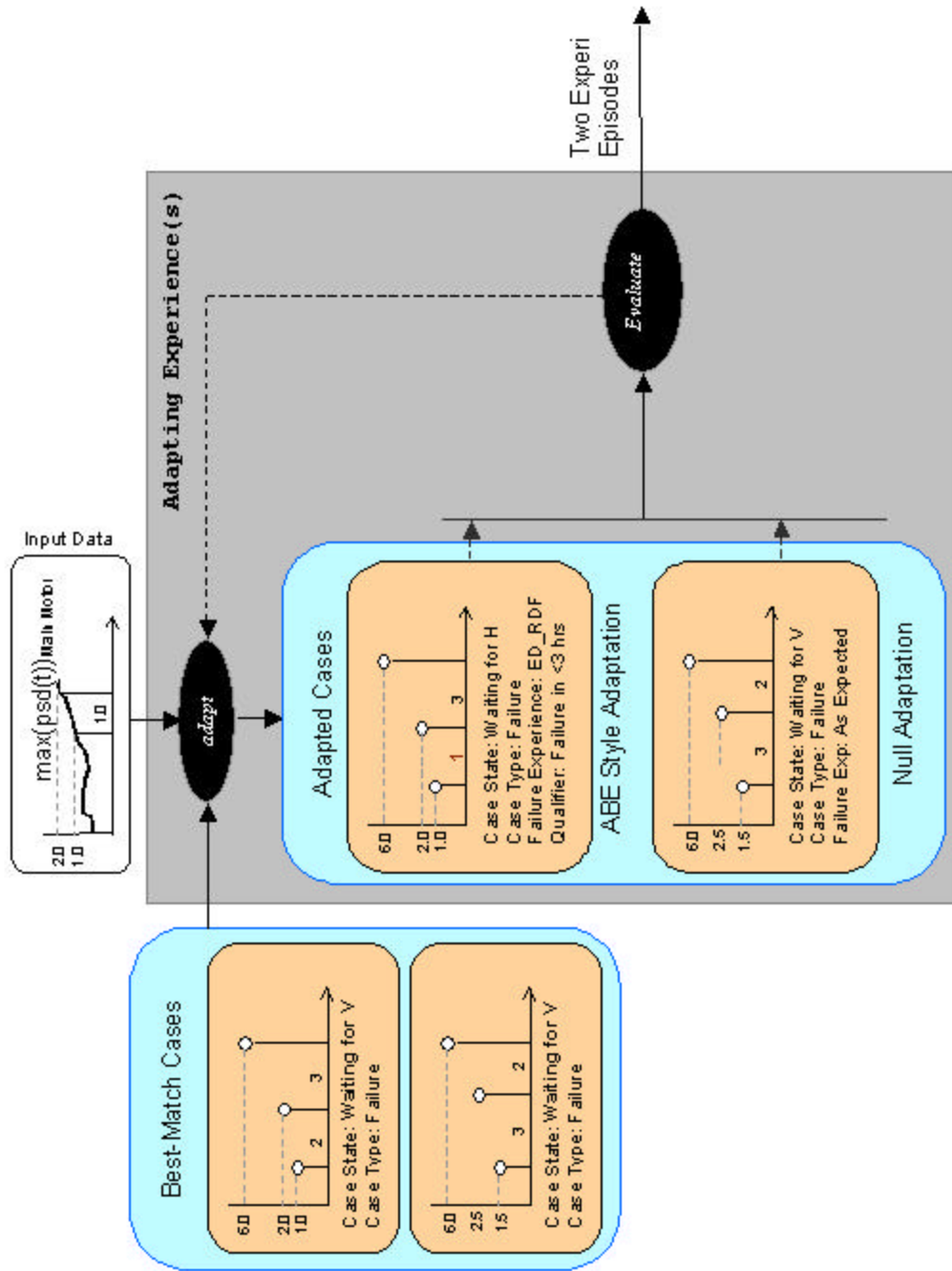


Figure 3.10 Example of a simple adaptation process

3.6.4 Learning from Experiences

CBR combines reasoning with learning, where the later is viewed as process of storing new experiences in the relevant place in memory [Kolodner 1993]. The update process usually includes update of indices so that new experiences can be recalled efficiently. However, in DAD CBR implementation, issues related to indexing are kept simple.

Figure 3.11 shows how the results of the adaptation process are stored in the Case Library. This closes the loop of the overall CBR architecture.

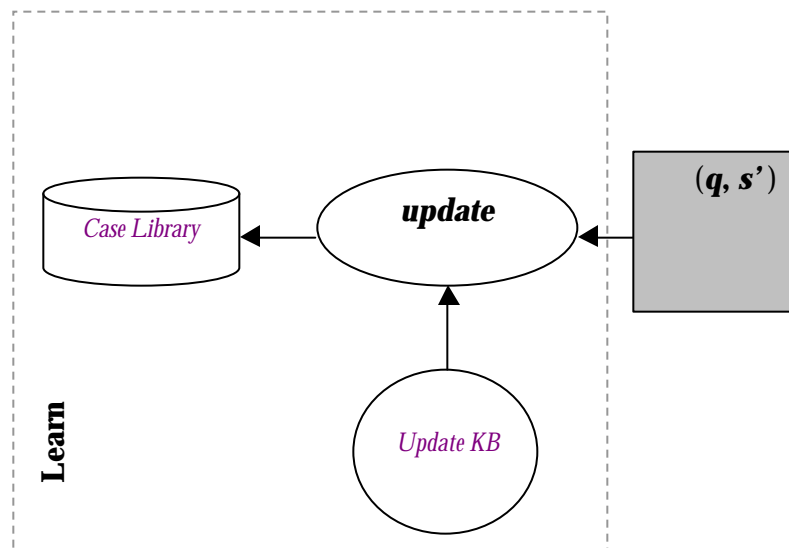


Figure 3.11 Learning Process in CBR [Dubitzky 1997]

3.6.5 Selecting Best Case(s) – Subsequent Remembering Process

When the case library is populated with some *closed* cases and some *opened* cases, the choice of the best-cases becomes more complex. In our methodology, we consider two factors. These include Case Distance (CD), and Success Factor (SF) of a case. The case distance can be considered as a similarity measure and is based on a running *failure experience* of a case.

The failure experience is relative to the temporal values defined in a case. These experiences can be either of a failure type that is consistent in a manner to the failure case (positive CD) or are of unexpected/false-alarm variety (negative CD). Figure 3.12 shows the failure experiences and their respective Case Distances.

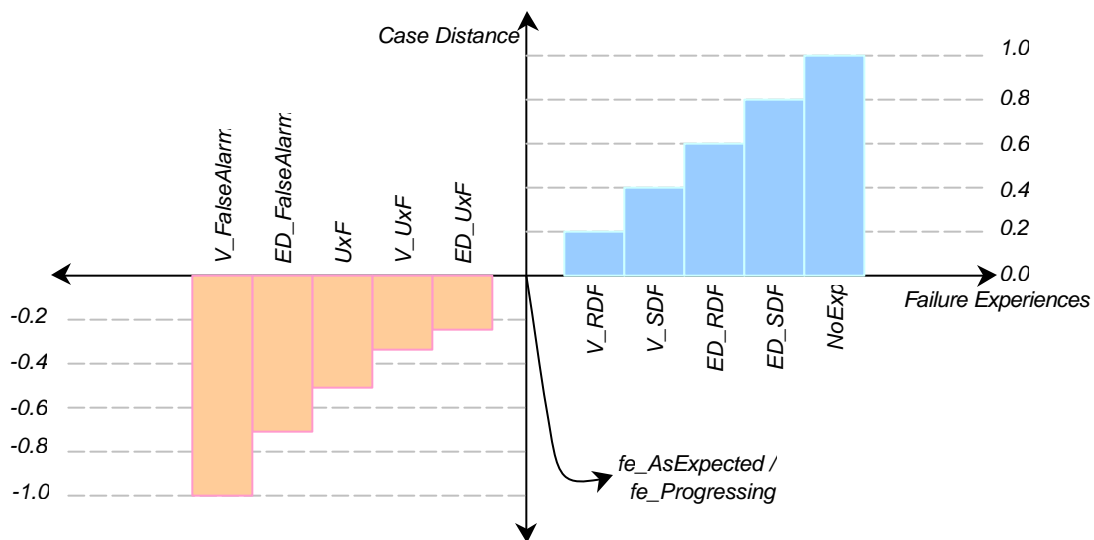


Figure 3.12 Case Distances for Failure Experiences

Success Factors (SF)

Some of our experiences prove to be more valuable because they happen more often. In our methodology, the diagnostic system keeps track of how successful has been one experience over the other. Therefore, if a *false alarm case* has proved to be right at several previous instances as opposed to a *failure case* with the same thresholds, then when the conditions recur, most likely it will be a *false alarm*. This notion is captured by the Success Factors (SFs) that are calculated every time a temporal landmark is reached, such as early detection, validation, etc.

As the agent's experience grows, case library becomes more populated. As a result of which the number of cases that will be retrieved from the library will grow over time. In order to select the best case(s), we created an exhaustive rule-base to address all possible scenarios.

Table 3.2 describes how different combination of cases is treated for best selection and what combination of case states will result in learning (case adaptations).

3.7 CONCLUSION

This chapter presented the methodology for implementing the dynamic diagnostics. The flexibility provided by the learning approach introduced in this chapter enables the DAD agent to learn from its successful as well as unsuccessful reasoning. This chapter only introduced the notion of *coordinated diagnostic* in the context of adaptation. Chapter 5 will discuss the details of this unique adaptation methodology. In the next chapter, we look at how the reasoning and learning capabilities enable the agent to exhibit flexible and active behavior.

Table 3.2 Case Selection Rule-Base

Case Experience	Case Type(s)	Case State	Selected Case(s)	Action
At least one type of case(s) can explain = 'AsExpected'	Normal	-	Select normal case(s)	Remove alarms if set
	False_Alarm	Any	Select false_alarm cases	Remove alarms if set
	Failure	Any case in Hazard	Select case in hazard	Alarm... Wait for Repair
		Any case in Validation	Select case in Validation with shortest time to hazard	Validation Alarm with lowest time
	Early Detection	Select case with highest SF	Follow	
No case is able to explain current data = 'NoExp'	Any	Any	None	Coordination (ask help)
Else	all cases that have failure experiences of SDF/RDF/UxF/False_Alarm variety			
Any case is Normal	Normal	Not possible	None	Not possible
All cases are False Alarms	False_Alarm	Any	Select case with highest SF	Remove alarms if enough confidence
All cases are failures	Failure	Any	Select case with least CD & highest SF	Select the least CD case & Learn new failure progressions
Some cases are False- Alarms and others are Failures	False_Alarm & Failure	Any	Select case with least CD, highest SF	Select the lease CD case & Learn new failure/false-alarm progressions

CHAPTER 4

ACTIVE DIAGNOSTICS

4.1 INTRODUCTION

This chapter introduces the *Active* part of the DAD agent architecture. Specifically we describe our methodology and implementation strategy to exert control over internal states as well as on the environment of a DAD agent. First we introduce what constitutes active diagnostics, and then we describe our approach to implement it.

The current discussion is based upon several concepts that were introduced in the context of the dynamic diagnostics earlier. The processes related to the active diagnostics form the control loop for the dynamic as well as the cooperative part of the DAD agent framework. The cooperative diagnostics is discussed in the next chapter.

4.2 WHAT IS ACTIVE DIAGNOSTICS

In literature the term *Active* is associated with diagnostics in two ways. In Sampath's work [Sampath *et al* 1998] it is used in the context of modifying UUT's control sequence in order to generate a diagnosable language. In Ishida's work on immunity-based sensor network [Ishida 1996], the active aspect emerges from the proactive monitoring of the sensor network by its constituent nodes. In this research we have significantly extended the scope of what is called active diagnostics.

Historically, the diagnostic and prognostic frameworks have been *passive* in design as well as implementation. This *passivity* stems from several factors including,

fixed sequence behavior, constant set of data processing algorithms, no interaction with diagnostic elements and system control, and lack of *proactive* approach to diagnostics.

Active diagnostics therefore can be defined in contrast, as the implementation of diagnostic framework that has at least some of the following *active* attributes

- The framework is dynamic in its behavior, in that it prioritizes its goals according to the situation at hand.
- The framework monitors the results of its actions, and modifies its reasoning to improve its performance.
- The framework selects data processing algorithms based upon the situation at hand.
- The framework controls the diagnostic elements, such as sensors, feature extractor, etc.
- The framework controls the main system (UUT) according to the current diagnosis and prognosis.
- The framework is proactive in that it initiates diagnostic activities, as they constitute its main goals, without an external event.

4.3 OUR APPROACH

In the current research, we have selected a subset of *active* attributes of the framework for implementation. We designed a diagnostic and prognostic system that exhibits the following attributes:

- Dynamic goal selection
- Controlled reasoning

- Proactive behavior of the system.

In the rest of the chapter we discuss the implementation details of these attributes.

4.4 ACTIVE IMPLEMENTATION

Figure 4.1 describes the main thread of an active diagnostic cycle. It starts with an evaluation of the *last known world-state*. For example, if one of the subsystems had a failure progression flag set, the diagnostic system would reschedule its goals to make sure that the important events (in this case a failure) are handled first. If time permits for further actions, then remaining goals are evaluated.

The framework is implemented with a fixed *cycle time* in mind. That is, the system will restart its thread after a prescribed time would pass. This approach can be commonly found in conventional diagnostic systems as well. In agent-based diagnostic system we have chosen to use a fixed cycle time in order to exert an inherent control on the agent activities since a dynamic agent can potentially generate so many cases that it may not be able to process all of its goals in the given time.

In view of these challenges the agent architecture is implemented in a way that it dynamically selects goals according to the criticality of situation. This chapter describes the algorithms implemented for this dynamic behavior. Dashed blocks in the figure represent the activities related to the cooperative diagnostics part of the DAD agent.

Another active control is provided by the selection of which cases to process. This is also related to the problem of available time. Hence, if an agent processes only the cases that have historically been more relevant (or successful) for a failure mode, it will save time to pursue more goals in the given time window.

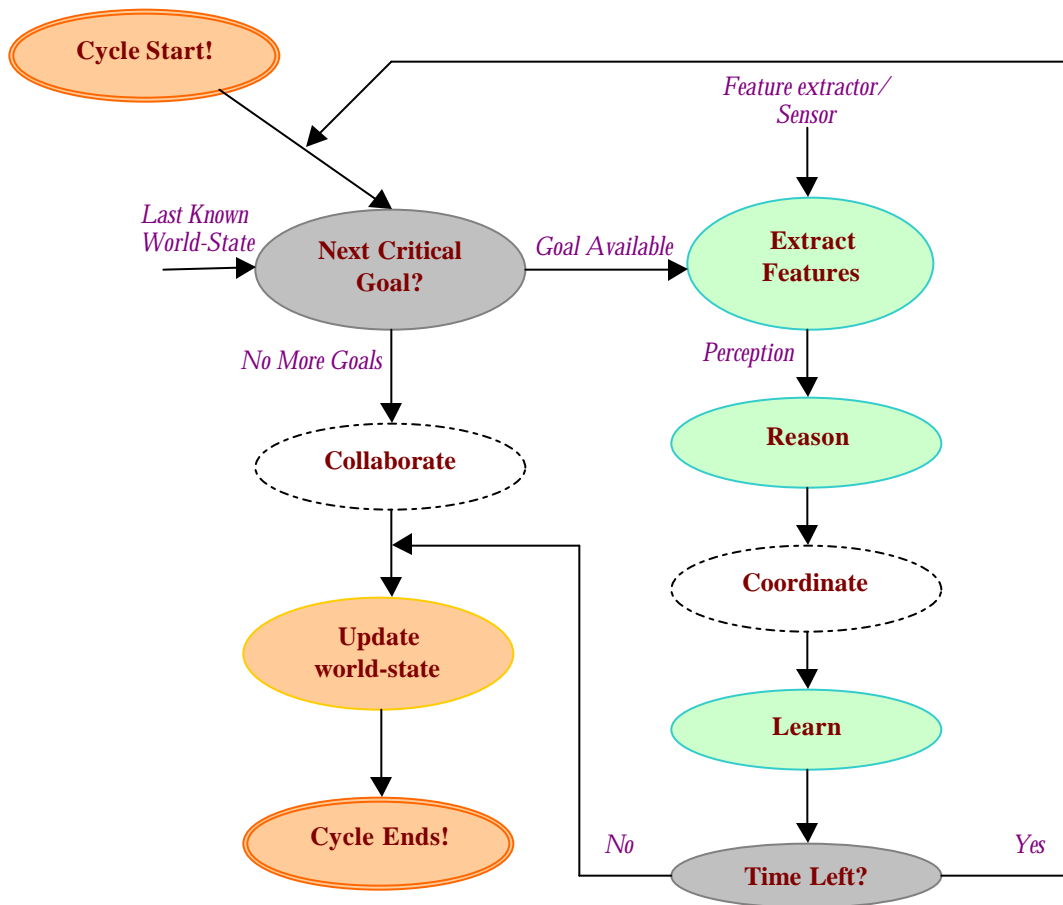


Figure 4.1 Active

diagnostic Cycle

4.5 DYNAMIC GOAL SELECTION

The goals of a DAD agent are to diagnose, as early as possible, critical failure events. The *bootstrapped* cases in the case library represent these goals. The agent on boot up learns about these goals and arranges them in order. The order is achieved by two factors: how long does it take to achieve a goal, and how critical is a goal. These factors together form the *last known world-state* (LKWS) as shown in the Figure 4.1.

The algorithm shown in Figure 4.2 is used to calculate the best next move for the agent to achieve its goals. In order to estimate what is the most critical failure to follow, the agent case-base is assigned an *a priori criticality* factor for each of its goals. This criticality factor can be obtained from a systematic study such as Failure Modes, Effects, and Criticality Analysis (FMECA) of a system. Alternately, it can be obtained via reasoning about the model of the system. In this research we are assuming a known criticality value that is used by the goals. In a FMECA style criticality definition [Andrews and Moss 1993], we have defined following four levels:

1. Catastrophic – complete loss of system,
2. Critical – sever reduction of functional performance resulting in a change in operational state,
3. Major – degradation of item functional output,
4. Minor – no effect on performance.

In our approach we have associated level of criticality with each temporal milestone. Therefore, one criticality factor is associated with early detection of a failure mode, and another one is coupled to the validation phase, and so on.

4.5.1 Last Known World-State

At the beginning of the agent's cycle, the decision about "which goals to follow and in what order", is based upon the values in a special structure that records what was known about the state of the world when the system finished its last cycle. This information is necessary in order to take decisions about which goals to pursue first. The important features of the world that are saved in this structure include the following:

- Assessment of each failure mode (any failure conditions in progress),
- Criticality associated with each failure mode,
- How long did it take to process each failure mode, and
- What was the expanse of the case search space

4.5.2 Abandoning Goals

As can be seen in the goal selection algorithm above, if the given cycle time is not enough for the agent, it will stop pursuing the next goal. Thus the agent will abandon some goals if the cycle time is not selected correctly for the agent, or if there are too many cases related to a particular critical failure mode.

4.5.3 Feature Extraction

Dynamic feature extraction and selection becomes a byproduct of the dynamic goal selection algorithm. This is due to the fact that not all features will be calculated, and hence not all the sensor data will be processed at all times, because it is possible that the agent will not pursue all of the failure modes. This is a major difference between agent-based implementation, in which agent attempts to do its best, and conventional procedural implementation, in which the program iterates through all the steps what.

GetNextCriticalGoalIndex

(goals, last_known_goal_state, time_into_the_cycle, total_cycle_time)

Goals array primarily stores the bootstrapped cases

Last_known_goal_state is an array of structures, of the same size as the goals array.

Time_into_the_cycle is the time lapsed after the current cycle began

Total_cycle_time is a global variable that controls the assigned cycle time for the agent

Function returns index of the goal array that should be followed next.

If no goal is possible, the function returns NULL

- The *pending_flag* is a flag, which if nonzero indicates that this goal is still pending in the current cycle. For each cycle the pending flag is set if the goal has not been considered for processing so far. It is set to zero whenever the goal is scheduled in the current cycle.
- For each goal, the failure condition indicates how advanced is a failure case in representing a failure progression.
- Calculate the criticality array of all the goals.

- For each goal *g* in *goals* array,
 - Calculate criticality as
Criticality(*g*) = (goals(*g*).criticality + last_known_goal_state(*g*).failure_condition)
* goals(*g*).pending_flag
- $goal_idx \leftarrow \operatorname{argmax}_g (\text{Criticality}(g))$
- If Criticality(goal_idx) = 0 then
 - No more goals available
(collaboration thread)
- Else
 - See if time permits to achieve this goal:
 - if last_known_goal_state(goal_idx).last_process_time
<(total_cycle_time – time_into_the_cycle)
 - Return goal_idx
- Else
- Return NULL

Figure 4.2 Algorithm to select possible goals dynamically

4.5.3 Search Space and Success Factors

Dynamic goal selection is in tune with the human diagnostician behavior that would inadvertently focus on the signals coming from the critical components, and may choose not to observe other possible failures if time does not permit.

The situation described above is acceptable in certain scenarios, but may be a problem in others, specifically when failure modes may be related and it would be useful to look at more than one failure at a given time to capture the overall picture.

In order to make the agent pursue all its goals, one way would be to loosen the restriction on agent's cycle time. This may have inadvertent effects on some temporal features, which assume a constant sampling rate. Another way is to improve the performance of the reasoning element by limiting the search space. We are using the latter approach by using a *minimum_success_factor* variable in the case object to control the search space. Therefore, during the search for a best match, when this value is set, all the cases that have proven to be less successful in the past are ignored in time-critical scenarios.

Example

In Figure 4.3 the concepts related to the dynamic goal selection are illustrated with an example that is based on an earlier example presented in Figure 3.7. At the commencement of the active diagnostic cycle the Last Known World-State (LKWS) table helps the agent select its first goal. In the given example, the failure mode 2 was selected first because of the criticality factor and because the ED flag was set. After this goal was achieved, the LKWS table was consulted again. It was found to be possible to achieve the

second goal in the given time frame (14 minutes of the remaining cycle time). Hence the failure mode 1 was processed.

After the completion of both goals, now there was nothing else to do, yet the time window had still not elapsed. Hence the collaboration activity was started. The LKWS table was updated next according to the new conditions observed during the current cycle.

After this step, the active diagnostic cycle was complete and the agent went into dormant state for the rest of the cycle time.

4.6 CONCLUSIONS

In this chapter we introduced the active part of the DAD agent architecture. This part of the agent schedules all its activities including reasoning, learning, and collaboration. The active diagnostic control is a novel implementation methodology and charts out a new design approach for the diagnostic and prognostic architecture. In the next chapter we look at the coordinated diagnostic features of the DAD agent architecture.

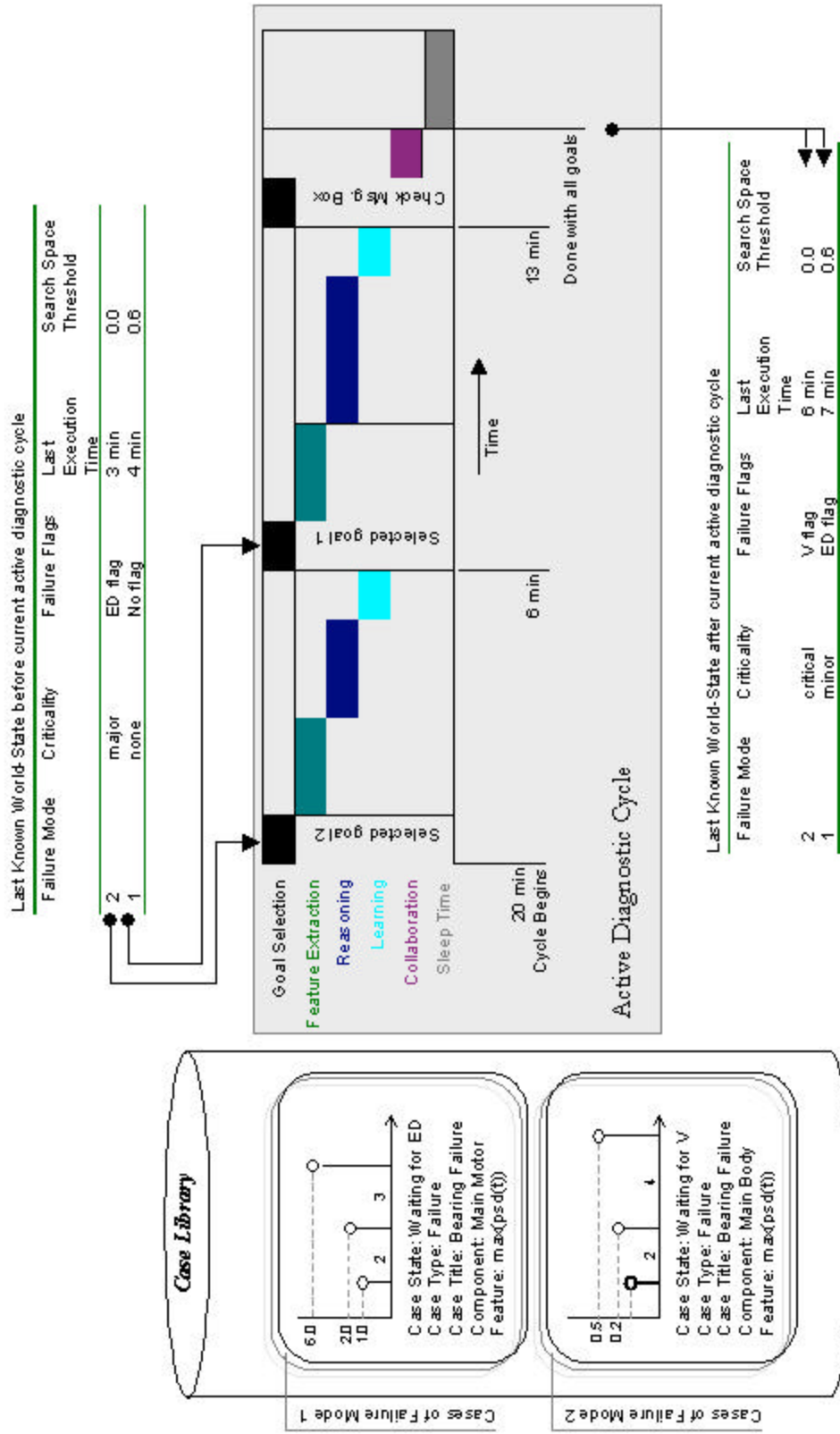


Figure 4.3 Example of an active diagnostic cycle

CHAPTER 5

COORDINATED DIAGNOSTICS

5.1 INTRODUCTION

One of the most powerful attributes of agent-oriented engineering is that the agents communicate among themselves. This communication is done in a message-passing manner forming a loosely coupled community of problem-solving entities, namely a Multiagent System (MAS). This chapter describes why multiagent system makes sense for the diagnostic and prognostic problem. Also it introduces a simple methodology for coordination among peer agents.

5.2 MULTIAGENT VIEW

In multiagent systems, we are interested in the domain-level behaviors that emerge from the coordinated behaviors of individual agents. In the light of Sycara's observations six challenges for a MAS for the diagnostic and prognostic problem can be listed as follows [Sycara 1998]:

- How to decompose diagnostic and prognostic problem and allocate tasks to agents.
- How to coordinate agent control and communication.
- How to make multiple agents act in a coherent manner.

- How to make individual agents reason about other agents and the state of coordination.
- How to reconcile conflicting goals between coordinating agents.
- How to engineer practical MAS for a distributed diagnostic and prognostic platform.

In the current research, we have simplified the design of the MAS by identifying a coarser agent implementation. Therefore, only one agent handles the diagnostics of one system. We also bypass the problem of coherence and of conflicting goals by keeping the goals of agents non-overlapping. Therefore, the agents communicate about the problems that do not exist in the other agent's realm by default.

5.2.1 Big Picture

Figure 5.1 shows how a multiagent system is formed. This gives a big picture in the context of distributed cooperative diagnostics. There are DAD agents responsible for

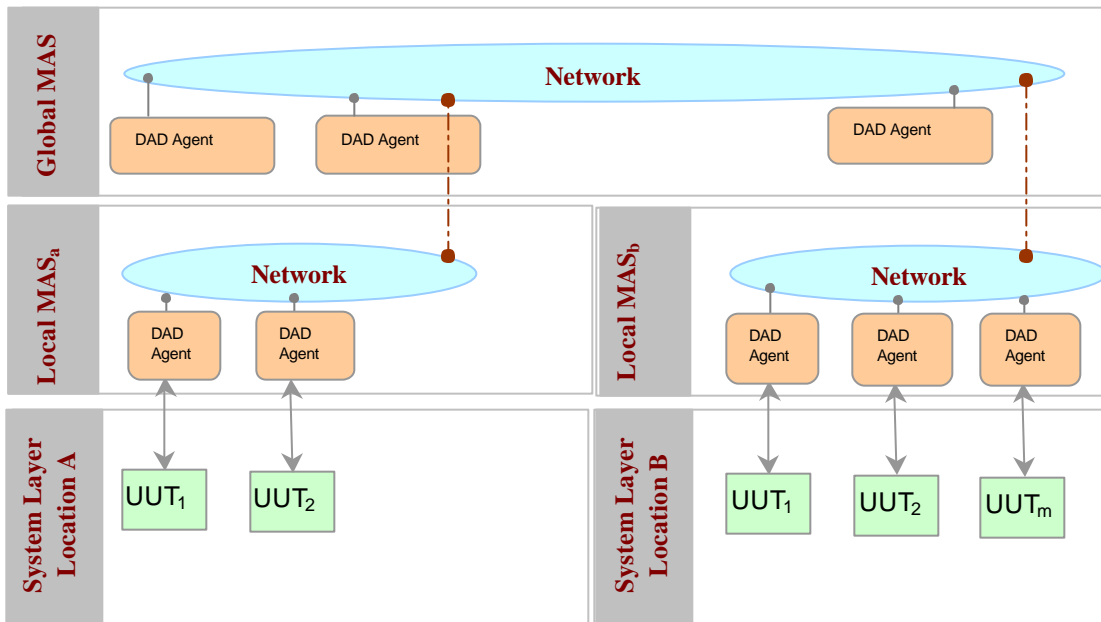


Figure 5.1 A community of DAD Agents for cooperative diagnostics

each system at each location A and B. They form a local MAS where they attempt to learn from each other. At the second level of hierarchy, these agents collaborate at global level to solve the problem.

5.2.2 Coordination/Collaboration Mechanism

We present a simple coordination and collaboration mechanism. The idea of coordination is *asking for help*, and collaboration is *helping other agents*. Both of these activities are required to form a coordinated diagnosis. A coordination request initiates collaboration activities at the peer agents.

Coordination is a last resort activity for an agent. Agents coordinate when they can not find any episode in their experience KB about the current problem. Coordination at this point is simply posting a help message in another agent's mailbox.

As was shown in the agent's active diagnostic cycle, if the agent is too busy solving its own problems, then it will simply be not available for collaboration. If on the other hand the agent has a time-slice that is not occupied, the agent will try to be helpful. In this research it is called collaboration. Therefore, collaboration is not a compulsory activity for an agent. This is in tune with the spirit of autonomy based agent design.

5.3 WHY COORDINATION MAKES SENSE

Coordination is an extra overhead for the framework and hence must be justified. In the context of human diagnostician, when a system exhibits a pattern that is previously unknown to the person, a smart diagnostician would consult his/her peers and seniors to find out the source of the problem and what action to take. This

coordination/collaboration activity may help the diagnostician learn new experiences regarding failures, operating modes, or false alarms.

In the machine's realm, the problem is two folds: how to communicate the problem to its peers, and what is the likelihood that the peers will be able to help solve the problem.

The first problem of communication is a large issue in itself that involves agent communication languages (ACL), definition of ontology, etc. In this research we have kept the communication very simple by canonizing the protocol.

In order to address the second problem we present a simple probabilistic argument about the benefit of collaboration among large-scale diagnostic systems. This argument is a variation of the famous *birthday problem* in probability¹.

Problem

Let us assume that there are n similar systems by the same manufacturer in operation around the globe. We also assume that each system is monitored by only one agent. This agent is responsible for generating diagnosis and prognosis of d critical failure modes. From the agent's perspective, these are the failures that are prognosable in that they are observable and predictable. For the sake of argument, we also assume that all the failure modes are equally likely. With these assumptions, we ask the question: how many systems should be collaborating with each other before the multiagent system can learn from each other's experience of the d failure modes?

¹ The problem is defined at: <http://mathworld.wolfram.com/BirthdayProblem.html>

Solution

Consider the probability $Q_1(n,d)$ that *no two systems* out of a group of n will have matching failures out of d equally possible failure events. Start with an arbitrary system's failure, then note that the probability that the second system's failure is different is $(d - 1)/d$, that the third system's failure is different from the first two is $[(d - 1)/d][(d - 2)/d]$, and so on, up through the n th system. Explicitly,

$$\begin{aligned} Q_1(n,d) &= \frac{d-1}{d} \frac{d-2}{d} \dots \frac{d-(n-1)}{d} \\ &= (d-1)(d-2)\dots[d-(n-1)]/d^{n-1} \end{aligned}$$

But this can be written in terms of factorials as

$$Q_1(n,d) = \frac{d!}{(d-n)!d^n}$$

so the probability $P_2(n, D)$ that two systems out of a group of n *have had the same failure experience* is therefore

$$P_2(n,d) = 1 - Q_1(n,d) = 1 - \frac{d!}{(d-n)!d^n}$$

In general, let $Q_i(n,d)$ denote the probability that a failure is shared by exactly i (and no more) systems out of a group of n systems. Then the probability that a failure is shared by k or more systems is given by

$$P_k(n,d) = 1 - \sum_{i=1}^{k-1} Q_i(n,d)$$

Example

For a group of systems with 10 failure modes, the distribution of failures is assumed to be uniform throughout the group (this is to say that there are no design

problems in the system and all the failures are equally likely), then the number of systems needed for there to be at least a 50% chance that two share failure experiences is the smallest n such that $P_2(n,10) \sim 0.5$. This is given by $n = 4$, since

$$P_2(4,10) = 1 - \frac{10!}{(10-4)!10^4} = 0.496$$

With 10 systems the probability that two will share a failure is given by

$$P_2(10,10) = 1 - \frac{10!}{(10-10)!10^{10}} = 0.9996$$

Discussion

As can be seen from the example above, it does not take many systems to generate shareable experiences. Although the above example assumed that all failures are equally likely, but if some are more likely than the others, then for those failures it would be more probable that the agent will find a peer agent with a matching experience anyway. Thus sharing these experiences becomes a valuable learning possibility even for a small community of systems.

5.4 ASKING FOR HELP – COORDINATION

Table 5.1 is a section of the Table 3.2 that shows when an agent asks for help. In essence an agent attempts to find a closest match in its experience KB to the current input data. Most of the time the agent is expected to find an episode from the case-library that matches the input data, or it can be modified to match the data. However, considering the variability with which a failure can appear it is also very probable that the agent does not find any episode that would satisfy its current percepts.

Table 5.1 Portion of the case selection rule-base

<i>Case Experience</i>	<i>Case Type(s)</i>	<i>Case State</i>	<i>Selected Case(s)</i>	<i>Action</i>
No case is able to explain current data = 'NoExp'	Any	Any	None	Coordination (ask help)

A DAD agent asks help by posting messages in the mailboxes of its peer agents. The discovery of similar peer agents is itself a challenging issue and is not dealt in the current research. In our implementation, we are assuming that the agent knows about its peers and also knows about how to post messages in their mailboxes.

Contents of the message include:

- Feature control information, like feature name
- Actual feature data, an array of actual feature data
- Time of posting, when is this message actually posted
- Message expiry stamp, an expiry stamp that suggests that peer agent should not worry about the message if too much time passes after the post.

5.4.1 Making Use of Assistance

Let us assume that in response to the help message, at least one peer agent was available to help the coordinating agent. The result of this help will appear as a response in the coordinating agent's mailbox. The format of the responding message is as follows:

- Time of posting
- Reply to: Originating message's time of posting

- An experience episode in the form of a case structure that was found to be the closest by the peer agent
- Similarity Factor

The agent on its next active diagnostics cycle would check the inbox. A modified active diagnostics cycle is shown in Figure 5.2 that shows these activities. On receiving the message, the message is inspected and the experience episode is modified to make it consistent with the collaborating agent's case-library. After this pruning step, the new case is made part of the library and is used just like other episodes.

5.5 BEING HELPFUL – COLLABORATION

All DAD agents are helpful by design. However, like their human counterparts, the agents are intelligent enough that they collaborate only after they are through with their obligations. As was shown in Figure 4.1, collaboration is just like another *goal* for the agent, scheduled as the last goal and only if there is some remaining time in the current active diagnostic cycle.

Collaboration is an aggregate of several activities, which include, checking to see which failure modes correspond, pruning requests to match the case-library structure, finding the most similar case, and returning the result in the coordinating agent's in box.

Figure 5.3 describes an example of coordinated diagnostics.

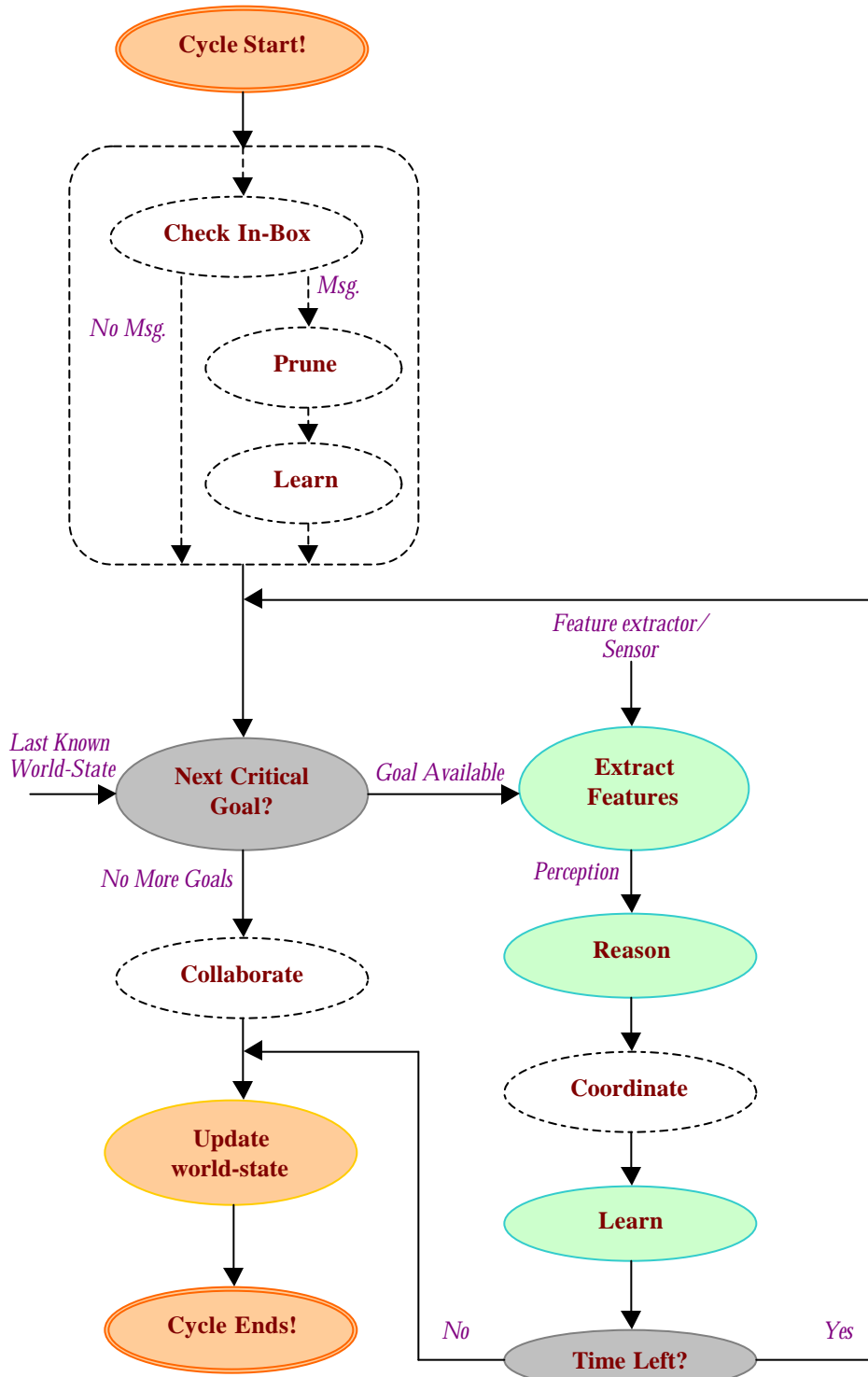


Figure 5.2 Modified Active Diagnostic Cycle

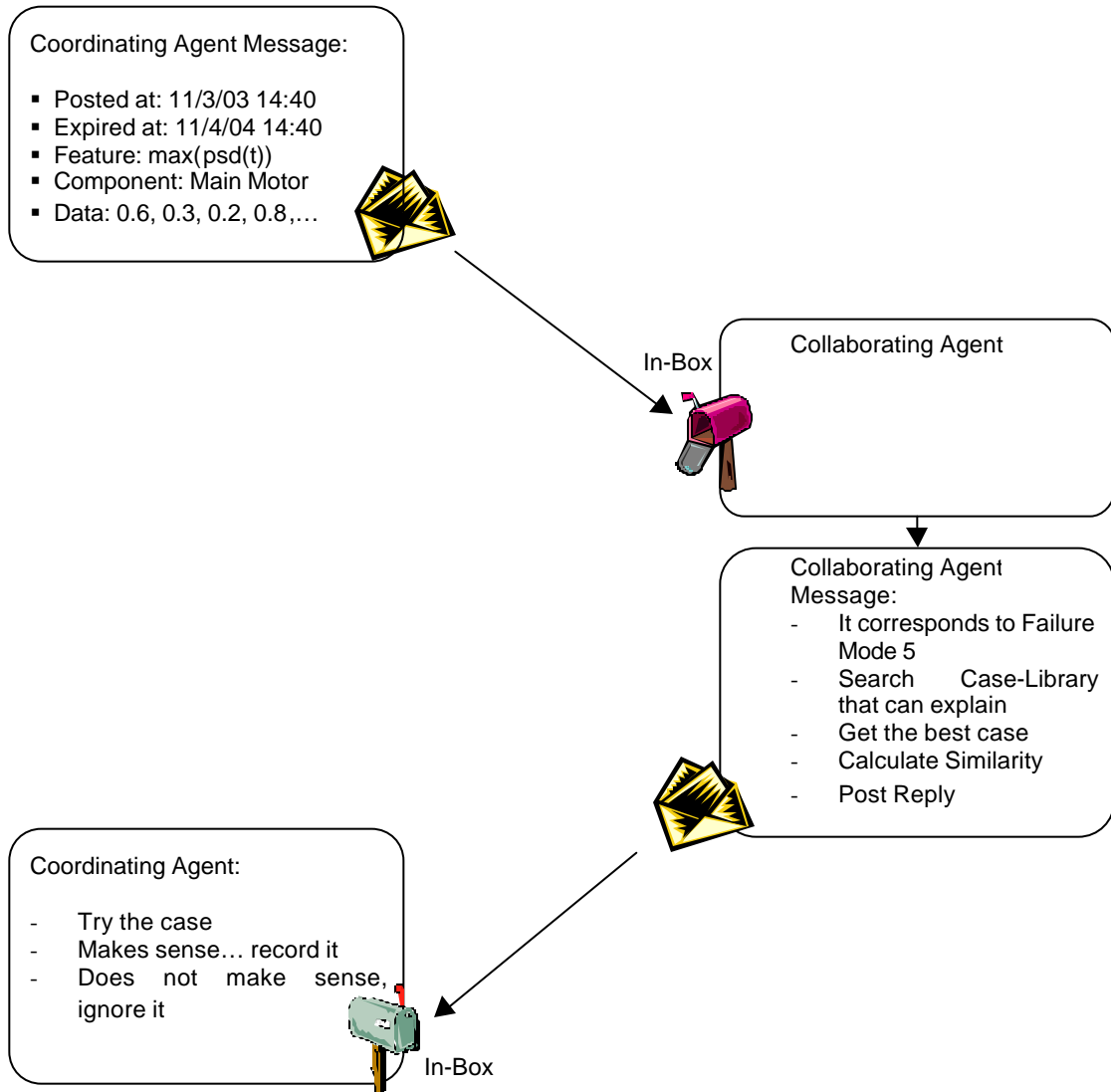


Figure 5.3 Example of a Coordination/Collaboration among peer DAD agents

CHAPTER 6

PERFORMANCE ASSESSMENT & VALIDATION

6.1 INTRODUCTION

The DAD framework introduced in this thesis is unique in its theoretical foundation as well as in its implementation methodology. It is difficult to compare the performance of this framework with conventional diagnostic systems. A major issue is that the framework is not presenting a new diagnostic or prognostic algorithm *per se*, rather it provides an intelligent approach for using the available diagnostic and/or prognostic algorithms. Therefore, it is the performance of the framework that is to be compared and the available literature does not provide any information on that. A related problem is the scarcity of process-data from peer frameworks that can be used for reference. In this scenario the best approach was to study the details of other frameworks and infer some general performance attributes.

In this chapter we present some quantitative measures that are used for validation as well as for the comparison of the DAD framework with a general static and passive framework that is assumed to be using the same diagnostic or prognostic algorithms. In this manner we cancel the variability of the diagnostic and prognostic algorithms in order to compare the frameworks.

6.2 PERFORMANCE MEASURES

We selected four measures to evaluate performance of the individual DAD agent from the detection and prediction perspectives.

6.2.1 Accuracy of Early Detection and Validation

Given a data stream with some N failure progressions, Validation Accuracy (VA) measures how many validation events were correct. Similarly Early Detection Accuracy (EA) measures the percentage of correct ED flags with respect to diagnostics. In a way these are similar to the *true positive* measure commonly found in diagnostic frameworks. However, the presented measures are different since they are used at the early-detection and validation points respectively, rather than at a post-failure moment. The validation milestone is the confirmation point of an impending failure. Therefore, it is very important that the validation is correct. In theory, a framework that implements a learning scheme should improve *accuracy* of its validation over time. A static framework, however, will generate an alarm with the given probability and hence will exhibit a constant accuracy.

An early-detection point is also very important since it flags the beginning of a failure progression. When this flag is raised, the dynamic framework spends more time searching for an appropriate case, and hence may abandon some other goals as explained in chapter 4.

Given a data stream of N failure progressions. E true-positive early-detection flags, and V true-positive validations, the early-detection and validation accuracy is respectively defined as:

$$EA = E/N*100$$

$$VA = V/N*100.$$

6.2.2 Precision of Prognosis at Validation

We use *precision* as the measure to assess the performance of the prognostic algorithm. It is commonly used in the prognostic literature such as [Noppadon 2001] and is defined as a measure of the narrowness of an interval in which the remaining life falls. We use validation event for declaration of the remaining life and hence it is also the point on which the precision estimates are based. Hence if there are N failure experiences being considered at the validation point, the precision is defined to be the range of their t_{vH} .

$$Prec(N) = \text{Maximum } t_{vH} - \text{Minimum } t_{vH}$$

6.2.2 Accuracy of Prognosis

This measures the number of times the actual time-series reached the bounds defined by precision. Therefore if there were N predictions made, and M out of them actually progressed to the predicted failure range, then the accuracy of prognosis is defined as:

$$PA = M/N*100$$

6.3 EXPERIMENTAL RESULTS

For the experiment we used a sequence of sampled bearing failure data obtained from a vibrometer. The main feature used was the power spectral density. The peak *psd* at each sample was combined to create a time-series feature signal. From this data we generated a set of simulated failure templates for the validation. We created some false-alarm templates from this data as well.

We bootstrapped three cases in the case-library, one each for a failure, normal mode, and a false alarm scenario. In our experimental strategy half of the failure templates were randomly selected and presented to the DAD framework in succession as a training phase. This marked the starting point of the test. The framework is then presenting with the remaining half of the failures, called test failures and their performance attributes were noted from this point on.

This procedure was repeated 25 times, starting every time with the original three cases in the case-library. The performance measures were noted for each run and their mean was calculated. Table 6.1 shows the results of these experiments.

In order to compare performance of this framework with that of a framework that does not learn, the bootstrapped cases and the failure templates in the library were selected to ensure that if the system stops its learning and modification cycle, constant attributes of performance are obtained as given by the first row of Table 6.1 below and is highlighted by the ‘static’ points in the Figure 6.1 and Figure 6.2 below.

Table 6.1 Experimental Results

	# Failures	# of Cases	%EA	%VA	%PA	Prec
Expected performance of a static framework →	0	3	80	75	75	25
	0	32	88.1	83.2	86	20
	1	31	88.5	83.4	83.7	24
	2	33	88.4	83.5	85	27
	3	33	88.5	83.6	82	32
	4	36	89.5	83.9	81.5	30
	5	41	88.6	84.2	81.1	27
	6	42	89.9	83.8	84.6	25
	7	34	91.2	83.7	87.7	25
	8	48	90.5	83.8	90.1	26
	9	54	92.1	83.9	90.6	24
	10	45	93.3	84.2	90.9	23
	11	52	93.4	84.3	93.4	20
	12	43	93.8	84.3	91.5	17
	13	57	93.8	84.5	92.5	18
	14	54	93.9	84.7	94.7	15
15	55	93.9	84.9	95.2	14	

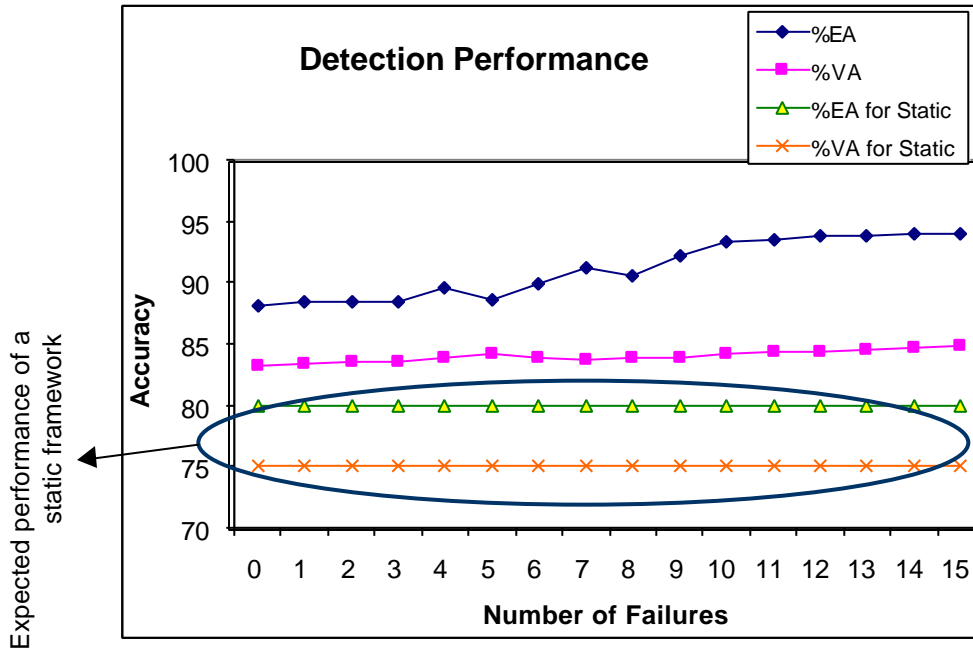


Figure 6.1 Detection Performance as it changes with number of failures

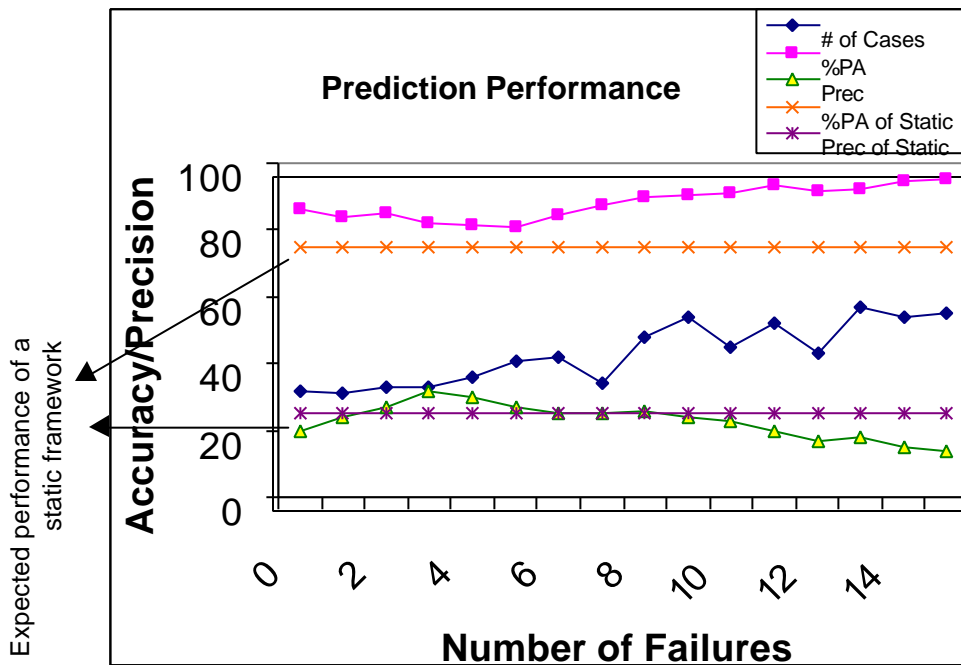


Figure 6.2 Prediction Performance as it changes with number of failures

6.4 DISCUSSION

The choice of the original failure templates was to emphasize the difference between performance of learning and non-learning frameworks. This initial *bias* reflects the practical problem of selection of thresholds and progression variables. Certain problem domains have failure characteristics that do not vary much. In this case if enough failure data is available for the diagnosticians, it is possible to choose failure alarms and progression variables that will give better performance than given above. However, in many other problem domains where such assumptions can not be made and where enough failure data is not available, the choice of the above alarms may be “too good” an assumption for diagnosticians. In our experiments we have made the above assumptions to highlight performance attributes.

As for as the diagnostic measures are concerned we can see that they steadily grow as the system learns about more failures. This steady growth is expected as with more failures the framework had more experience episodes to draw information from. However, as can be noted, the early-detection is improved significantly as compared to the validation accuracy. This can be explained by the fact that the learning of a good validation threshold is a very tricky issue for the framework. The problem is complicated by the presence of false alarms that have similar signatures. The algorithm has a tendency to make more of an educated guess. However, the modification strategy of early-detection seems more robust.

The prediction measures in the experiment are more interesting. As can be observed the accuracy drops with more failure signatures. At the same time the precision values that represent the range of prediction at the validation point increases. This can be

explained by the fact that a framework that modifies experiences dynamically also decreases the precision. However, as more of these episodes are created, the choice of predicted episodes improves and hence the prediction accuracy becomes better, as well as the precision of the prediction improves.

From all these measures, we validated that the system is “doing the right thing”. Therefore the framework dynamically improves its performance with time and as it learns more about failures.

CHAPTER 7

PERFORMANCE CONDITIONS

In the previous chapter, it was shown that the intelligent agent improves its performance over time. This improvement was a result of the fault-progression experiences that this agent was subjected to during its lifetime. An important question can be asked: Under what conditions can the performance variables of the agent start to degrade, even below the expected performance levels of that of a static diagnostic and prognostic system?

There are two aspects of the performance conditions. The first is related to the control of the CBR system itself, i.e. is it possible that in the long run, when the CBR system is populated with a large number of cases, the accuracy of detection and prediction decreases due to the presence of bad or conflicting cases in the memory, and how does the system handle this situation? The second aspect is: Is it possible that a particular set of external events may cause the agent to learn incorrectly, and hence result in a degraded performance?

The first problem is a natural one that arises in dynamic systems that learn on the fly. The design of the system has indirectly addressed these issues by the inherent control provided by the CBR system that assigns a *success factor* to each of the cases in the library. Given that the agent can be adapting several prior experiences at a given moment in time, there arises a possibility that the agent will learn conflicting experiences. Moreover, if we relax the assumption that the feature data is always reliable, there is a

possibility of incorrect or *bad* experience creeping into the case library. How does such a bad learning affect the accuracy of the system in general? Since the CBR system takes into account all the relevant experiences, or the most successful ones, at a given time, few bad experiences will be outweighed during the action selection part and will not result in performance degradation below that of the static framework. This will not be true, however, if this agent learns bad episodes of failures or false alarms, more than it learns the good ones. In the first case, we expect that the bad cases in the library are not representative of the real situation and will eventually go down in their *success factor* with every good episode the agent is exposed to. In the second case the agent will generate a large number of false alarms, or will fail to register some of the real failures. A higher level of reasoning can be used at this point to detect such an anomaly and correct for it. However, this problem highlights the importance of selection of good features to ensure improved performance of the system.

The second condition for performance is based upon several assumptions. For example, it is assumed that a collaborating agent provides the correct information. That condition should be in place before the coordination is useful. A serious effect of the relaxation of this condition is that the case library of the coordinating agent will be populated with incorrect episodes. The sources of this bad information can be the different operating conditions of the collaborating agent, bad communication channel, etc. For example, a message sent by the collaborating agent that predicted a failure in 10 hours might arrive as a predicted failure in 100 hours due to bad communication. This time frame may be too long for the coordinating agent to consider the current incipient fault and it may choose to ignore it, resulting in degraded accuracy.

Similarly another assumption made in this research is that there are no intermittent fault signatures. If this condition is relaxed, the agent will attempt to learn from each intermittent fault. If we assume that these fault signatures are the same, but they appear randomly, the agent will erroneously learn that a particular fault has happened so many times, giving a high *success factor* to the case that was a bad experience in the first place. If we assume that these random faults also exhibit different fault signatures, then the agent will populate the library with unnecessary cases that may incorrectly reflect some other failures, resulting in reduced detection performance.

In short, the performance improvement with experience is not an easy proposition and it must be implemented with care. All the assumptions discussed earlier should be evaluated to make sure that they apply to the domain of application. If some of the assumptions can not be justified than alternate solutions should also be looked into. Otherwise, it would be likely that the intelligent agent architecture will perform as good as the static and passive diagnostic system, if not worse.

CHAPTER 8

CONCLUSIONS

In concluding this research we note that the intelligent frameworks are a promising paradigm to meet the challenges presented by large-scale systems. Although some of the assumptions made in this research may not be realizable in real-world problems, the general approach presented by this thesis is a novel and promising dimension.

The strategy presented in this thesis is just one way of implementing the generic dynamic and active diagnostic system out of many other possibilities. As we learn more about the human reasoning and learning, the DAD approach to designing intelligent diagnostic and prognostic systems can be implemented in better and more cost-effective ways. In the end, we want the software diagnosticians to perform as well as their human counterparts, if not better.

Some of the practical problems that are partially addressed in this research include the modification of behavior after retrofit or redesign of the UUT in the field. This issue in the military is equivalent to the damage of the UUT in combat zone. How these frameworks will adapt to drastic and unexpected changes in the UUT is an interesting future research possibility.

Another future research may be the extension of the ‘active’ part of the framework to the system controls. This is usually not possible for a diagnostician who is given the responsibility after the UUT has been designed and installed. However, if such

an opportunity exists at design and testing time, the DAD framework can be extended to implement a dynamic control loop to exert its own influence on the current and future behavior of the UUT, in order to extend the life of the system.

APPENDIX A

INTELLIGENT AGENTS – AN OVERVIEW

APPENDIX A

INTELLIGENT AGENTS – AN OVERVIEW

A.1 INTELLIGENT AGENTS - PHILOSOPHICAL ISSUES

Since the thesis is centered on this idea, an Intelligent Agent needs to be defined succinctly. The literature is full of different definitions and sometime contradictory attributes for an Intelligent Agent. The following overview covers major streams of thoughts on the subject:

Foner noted in [Foner 1993] that some of the concepts that can be called *computational agents* date back to the late fifties of the last century. However, the growth of Internet and E-commerce created a hyperbole in the study, research and development of software and hardware systems that have been termed as Intelligent Agents. Several points of views exist on the *true* definition and characterization of Intelligent Agents. In their paper Wooldridge quote Carl Hewitt’s remarks [Wooldridge and Jennings 1995]

... The question “What is an agent?” is embarrassing for agent-based computing community in just the same way that the question “What is intelligence?” is embarrassing for the mainstream AI community.

Several researchers have given their insights in to the topic of what should and should not be called *the* Intelligent Agent. Most of them consider “agents” as “autonomous intelligent agents”. For example, Foner and Stan Franklin discuss agenthood without assigning any meaning to the “intelligent” part of it [Frankling and

Graesser 1996]. On the other side of the coin, we find people who believe that any thing that is called an agent is an agent.

Thus we find very generic definitions in the literature, taken directly from the dictionary meaning of agents, as “someone or something that acts on your behalf”. Then there is the more common notion of agents as given by Franklin: An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future. This definition has several essential ingredients of agenthood, like situation in an environment, and the presence of an agenda. However, some of the themes like “senses in the future”, are too vague for this definition to be objective.

For Foner, an agent is necessarily intelligent. His thesis is a reaction to the commercial offerings of agents. He criticizes the excessive anthropomorphizing efforts that make some software user interfaces look more human, while they lack everything that an agent should have. Some of his “crucial notions” for an agent include concepts of *Autonomy* (agent can pursue an agenda independently of its user. That in turn requires aspects of *periodic action*, *spontaneous execution*, and *initiative*), *Personalizability* (agent must be educable. Components of *learning* and *memory* come in this context), *Discourse*: (a two-way feedback, in which both parties make their intentions and abilities known, and mutually agree on something resembling a contract about what is to be done), *Graceful degradation*: (in case of domain mismatch, or bad communications, the contract should be degraded gracefully and if possible most of a task may still be accomplished instead of failing to accomplish any task, and *Cooperation*: (the two parties interact more as peers in agent-oriented systems).

While Foner's "crucial notions" seem a bit more restrictive, they certainly direct us to a useful description of agency. It can be seen that almost all researchers call *autonomy* as an essential requirement for intelligent agents. Autonomy is also sometimes used to define intelligence. For example Reddy's major characterizations of Intelligent Software is that it must be capable of creating for itself an agenda of goals to be satisfied [Reddy 1995]. Vidal et al compare autonomy of the agent to human free will, that enables an agent to choose its own actions [Vidal *et al* 2001]. Autonomy is coupled with social capabilities and design objectives, and an intelligent agent is thus defined by Wooldridge [Wooldridge 1999] as "a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." Wooldridge suggests a *weak notion* of agenthood as against a *strong notion*. The weak agenthood enjoys the properties of *autonomy*, *social ability* (that includes Agent-Communication Language ACL), *reactivity*, and *proactivity*. The *strong notion* of agenthood are based on, what is called the *mentalistic* notions, such as *belief*, *knowledge*, *intention*, and *obligation*.

Other notions of agenthood that have appeared in the context of Intelligent Agents are the following:

- *Mobility*: the ability of an agent to move around in a network, and
- *Rationality*: the assumption that an agent will act to achieve its goals

Out of the above-mentioned notions, agents with mobility, i.e. the *mobile agents* have attracted lots of attention lately. It is interesting to point out that agents from the AI groups tend not to be mobile. The mobile agents are more commercial rather than research products. These agents roam around the web, in order to reduce the network

traffic, and perform useful services on behalf of the users. These agents provide little more than clever string matching while some of the agents have natural language programming capabilities. Thus CompassWare offers an Intelligent News Filter that parses natural language to perform search. Other mobile agents are essentially search engines and their agenthood has been questioned by AI community.

The notion of *rationality* is also considered as the essential characteristic. Thus Russell and Norvig define a *Rational Agent* in terms of its percepts and actions. Thus for each possible percept sequence, a rational agent should do whatever is expected to maximize its performance measure [Russell and Norvig 1995].

Another type of agent that has been considered an Intelligent Agent is based on the ACL such as the KQML (Knowledge Query Modeling Language). Thus for Genesereth these *software agents* "... can be as simple as subroutines; but typically they are larger entities with some sort of persistent control." The software agents are the software components that communicate with their peers by exchanging message in an expressive ACL. These agents are also called Typed-Message Agents or ACL agents [Genesereth and Ketchpel 1994]. The agents must exchange messages in a shared message peer-to-peer protocol (like KQML) to accomplish a task. The motivation behind the protocol is to differentiate these kinds of software agents from other software technologies, like Expert Systems and Object Oriented software.

A.2 INTELLIGENT AGENT ARCHITECTURES

Intelligent Agent architectures can be divided in four classes [Wooldridge 1999] discussed next.

A.2.1 Reactive Agent Architecture

Reactive Agent architecture is simple to implement. It is also more robust and computationally tractable. A well-known architecture in this category is *Subsumption Architecture* of Rodney Brooks [Brooks 1986]. Generally, the reactive architecture does not include any complex symbolic representations and no symbolic reasoning is applied. In many implementations, the behaviors are implemented by simple rules of the form

Situation \rightarrow Action

This represents a simple map between perceptual input to actions.

A.2.2 Logic Based Agents Architecture

Intelligent Agents based upon this architecture follow the traditional approach to building artificially intelligent systems, namely symbolic AI. Thus intelligent behavior is generated by giving that system a *symbolic* representation of its environment and its desired behavior, and syntactically manipulating this representation. Thus agents act as *theorem provers*. METATEM and Concurrent METATEM architectures are good candidates for these reasoning agents [Barringer *et al* 1989].

A.2.3 BDI Agent Architecture

The BDI architecture has its roots in *practical reasoning* – the process of deciding, moment by moment, which action to perform in the furtherance of our goals. Planning is more goal-oriented behavior and is suitable for the BDI agents. The practical reasoning is concerned about two processes: deciding *what* goals should be achieved, and *how* to achieve these goals. Each BDI agent has a sophisticated reasoning architecture that consists of different modules that operate asynchronously. Starting from the set of

beliefs, representing the information about the environment, the agent generates options by *options generation function*. A filter function represents the agent's *deliberation* process, which determines the agent's intentions, based on its current beliefs, desires, and intentions. The *action selection function* determines an action to perform based on current intentions.

A.2.4 Layered Architecture

In Layered Architecture, decision making is realized via various software layers, each of which is more-or-less explicitly reasoning about the environment at different levels of abstraction.

REFERENCES

- Aghasaryan, A., Fabre, E., Benveniste, A., Boubour, R., and Jard, C. (1998). Fault Detection and Diagnosis in Distributed Systems: An Approach by Partially Stochastic Petri Nets, *Journal of Discrete Event Dynamical Systems: Theory and Applications*, pp. 203-231
- Althoff, K-D., Auriol, E., Bergmann, R., Breen, S., Dittrich, S., Johnston, R., Manago, M., Traphöner, R., Wess, S. (1995), Case-Based Reasoning for Decision Support and Diagnostic Problem Solving: The INRECA Approach, Proceedings of the 3rd German Workshop on CBR, LSA-Report 95-02, Germany
- Andrews, J.D., and Moss, T. R., (1993) Reliability and Risk Assessment, Longman Scientific & Technical Publishing, UK
- Armengol, E., Palaudaries, A., and Plaza, E., (2001). Individual Prognosis of Diabetes Long-term Risks: A CBR Approach, *Methods of Information in Medicine*, Vol 40, Schattauer Publishers, Germany
- Baroni, P., Lamperti, G. , Pogliano, P. and Zanella, M. (1999). Diagnosis of Large Active Systems, *Artificial Intelligence*, vol. 110, pp. 135-183
- Barringer, H. , Fisher, M., Gabbay, D. , Gough, G. and Owens, R. (1989). MetateM: A Framework for Programming in Temporal Logic, In *REX Workshop on Stepwise Refinement of Distributed Systems: Model, Formalisms, Correctness*, LNCS Vol. 430, pp. 94-129, Springer – Verlag, Berlin, Germany
- Brooks, R. A. (1986). A Robust Layered System for a Mobile Robot”, *IEEE Journal of Robotics and Automation*, 2(1), pp. 14 – 23
- Debouk, R. I. (2000). Failure Diagnosis of Decentralized Discrete Event Systems, Ph.D. Thesis, University of Michigan, Ann Arbor, MI
- DeLoach, Scott A. (1999). Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems, *Agent-Oriented Information Systems '99 (AOIS'99)*, Seattle WA
- Dubitzky, W. (1998). Knowledge Integration in Case-Based Reasoning: A Concept-Centred Approach, Ph.D. thesis, University of Ulster UK
- Engel, S. J., Gilmartin, B. J., Bongort, K., and Hess, A. (2000). Prognostics, The Real Issues Involved with Predicting Life Remaining,” *IEEE Aerospace Conference Proceedings*, vol. 6, pp. 457 – 469
- Foner, L. N. (1993). What’s An Agent, Anyway? A Sociological Case Study, Agents Memo 93-01, Agents Group, MIT Media Lab, MIT
- Foxvog, D. and Kurki, M. (1991). Survey of Real-time and On-line Diagnostic Expert Systems, *Real Time Systems*, 1991. Proceedings., Euromicro '91 Workshop on, pp 61 – 69

Frank, P. (1990). Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge Based Redundancy, *Automatica*, vol. 26, pp. 459-474

Franklin, S. and Graesser, A. (1996). Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents, *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag

Genesereth, M. R. and Ketchpel, S. P. (1994). Software Agents, *Communications of the Association for Computing Machinery*, pp 48-53

Ghafoor, A. Kershaw, R. S. (1989). A design methodology for expert systems for diagnostic and repair, *Conference Proceedings, Eighth Annual International Phoenix Conference on Computers and Communications*, pp. 550 –554

Hadden, G. D., Bennett, B. H., Vachtsevanos, G., and Dyke, J. V. (2000). Shipboard Machinery Diagnostics and Prognostics/Condition Based Maintenance: A Progress Report, *IEEE Aerospace Conference Proceedings*, vol. 6, pp. 277– 292

Ishida, Y. (1996). Active Diagnosis by Immunity-Based Agent Approach, *Proc. DX 96, Val-Morin, Canada*, pp. 106-114

James, M. L. and Dubon, L. P. (2000). An Autonomous Diagnostic and Prognostic Monitoring System for NASA's Deep Space Network, *IEEE Aerospace Conference Proceedings*, Vol. 2, pp. 403 – 414, 2000.

Khripet, N., (2001) An Architecture for Intelligent Time Series Prediction with Causal Information, Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA

Kinney, D., Georgeff, M., and Rao, A. (1996). A Methodology and Modelling Technique for Systems of BDI Agents, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Lecture Notes in Artificial Intelligence 1038*, Springer

Kolodner, J. (1993). *Case-Based Reasoning*, Morgan Kaufmann Publishers, San Mateo CA

Mozetic, I. (1992). Model-Based Diagnostics: An Overview, *Advanced Topics in Artificial Intelligence*, V. Marik, O. Stepankova, R. Trappl (Eds.), Springer-Verlag, pp. 419- 430

Nwana, H. S. , Ndumu, D. T. , Lee, L. C. , and Collis, J. C. (1999). Zeus: A Toolkit for Building Distributed Multi-Agent Systems, *Applied Artificial Intelligence Journal*, Vol. 13, No. 1, pp. 129 – 186

Odell, J., Parunak, H. V. and Bauer, B. (2000). Representing Agent Interaction Protocols in UML, *Agent Oriented Software Engineering – Proceedings of the First international Workshop (AOSE-2000)*, P. Ciancarini and M. Wooldridge (Eds.), Springer-Verlag, Germany

Pencolé, Y. (2000). Decentralized Diagnoser Approach: Application to Telecommunication Networks, in *Proc. of DX'2000, Eleventh International Workshop on Principles of Diagnosis*, pp. 185-192

Pouliezos, A. and Stavrakakis, G. (1994). Real-time Fault Monitoring of Industrial Processes, Kluwer Academic Publishers, Boston, MA

Reddy, R. (1995). To Dream the Possible Dream, Turing Award Lecture, Communications of the Association for Computing Machinery, Vol. 39, No. 5, pp. 105-112

Reiter, R. (1987). A Theory of Diagnosis From First Principle, Artificial Intelligence, vol. 732, pp. 57 – 95

Riodran, D., and Hansen, B., (2002), A Fuzzy Case-Based System for Weather Prediction, Engineering Intelligent Systems, CRL Publishing, Vol 3, pp. 139 – 146

Roemer, M. J., and Kacprzyński, G. J. (2000). Advanced Diagnostics and Prognostics for Gas Turbine Engine Risk Assessment, IEEE Aerospace Conference Proceedings, vol. 6, pp. 345 – 353

Roemer, M. J., Kacprzyński, G. J., Nwadiogbu, E. O., and Bloor, G. (2001). Development of Diagnostic and Prognostic Technologies for Aerospace Health Management Applications, IEEE Aerospace Conference Proceedings, vol. 6, pp. 3139 – 3147

Russell, S and Norvig, P. (1995). Artificial Intelligence: A Modern Approach, Prentice Hall, New Jersey

Sampath, M. (1995). A Discrete Event Systems Approach to Failure Diagnosis, PhD thesis, University of Michigan, Ann Arbor, MI

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. (1996). Failure Diagnosis Using Discrete-Event Models," IEEE Trans. Contr. Syst. Tech., vol. 4, no. 2, pp. 105-124

Sampath, M., Lafortune, S., and Teneketzis, D. (1998). Active Diagnosis of Discrete-Event Systems, IEEE Trans. Automat. Contr., vol. 43, no. 7, pp. 908-929

Sengupta, R. (1998). Diagnosis and Communication in Distributed Systems, in Proc. of WODES 1998, International Workshop on Discrete Event Systems, pp. 144-151. Published by IEE, London, England

Swanson, D. C. (2001). A General Prognostic Tracking Algorithm for Predictive Maintenance, IEEE Aerospace Conference Proceedings, vol. 6, pp. 2971– 2977

Sycara, K., Decker, K., and Pannu, A. (1996). Distributed Intelligent Agents, IEEE Expert, Vol. 11, Issue 6, pp. 36 – 46

Sycara, K. P. (1998). Multiagent Systems, AI Magazine 19(2), American Association for Artificial Intelligence, pp. 79-92

Thakkar, A and Vachtsevanos, G. (2001) Prognostic Enhancements to Diagnostic Systems” - USN OSD SBIR Phase II N65540-01-C-0015, Phase II Kickoff Meeting,

Ulerich, N. and Powers, G. (1988). On-line Hazard Aversion and Fault Diagnosis in Chemical Processes: The Digraph + Fault-tree Method, IEEE Trans. Reliability Engineering, vol. 37, no. 2, pp. 171-177

Vachtsevanos, G (1998), A Proposal for ADIP," Internal ICSL Report, Georgia Institute of Technology, Atlanta, GA

Vidal, J. M., Buhler, P. A., and Huhns, M. N. (2001). Inside an Agent, IEEE Internet Computing, pp. 82 – 86

Viswanadham, N. and Johnson, T. (1988). Fault Detection and Diagnosis of Automated Manufacturing Systems, in Proc. 27th IEEE Conf. on Decision and Control, pp. 2301-2306

Voos, H. (2000). Intelligent Agents for Supervision and Control: A Perspective, Proceedings of the 15th IEEE International Symposium on Intelligent Control (ISIC 2000), pp. 339 – 344, Greece, July 17 – 19

Watson, I. (1997). Applying Case-Based Reasoning: Techniques for enterprise systems, Morgan Kaufmann, San Francisco CA.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent Agents: Theory and Practice, Knowledge Engineering Review, 10(2)

Wooldridge, M. (1999). Intelligent Agents, Multiagent Systems, G. Weiss (ed.), The MIT Press, pp. 3 – 51

Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design, Autonomous Agents and Multi-Agent Systems, Vol. 3, No. 3, pp. 285 – 312

Wooldridge, M. and Ciancarini, P. (2001) Agent-Oriented Software Engineering: The State of the Art, Lecture Notes in AI Vol. 1957, in P. Ciancarini and M. Wooldridge, (Edt), Springer-Verlag

VITA

Irtaza Barlas was born on July 11, 1967 in Karachi, Pakistan. He holds a B.Sc Engineering degree in Electrical Engineering from the University of Engineering and Technology Lahore, Pakistan, and an M.S in Electrical and Electronic Engineering from the California State University, Sacramento. He was involved in the data communication industry of Pakistan for several years. He is also a full time faculty member at the National University of Sciences and Technology in Pakistan, where he is affiliated with the Department of Computer Engineering.

He is a lifetime member of Eta Kappa Nu the honor society in Electrical Engineering. He was the founding advisor of the Advanced Computing Machinery's chapter in the College of Electrical and Mechanical Engineering in Pakistan.

During his study at Georgia Tech, he has been involved in designing and implementing software architectures and solutions for various supported projects. He also worked closely with the industry at several Coop positions during his course of study. Irtaza's research interests are in diagnostics, prognostics, intelligent agents, reasoning and learning frameworks, and computer vision.

Upon the completion of his studies, Irtaza intends to work in the industry as well as keep his affiliations with academia.