PIONEERS IN METHYLENE BLUE TESTING SINCE 1974

Micro-T
THE MICROFILM

18000 COUNTY ROAD 3 BURNVILLE, MN 55337 USA
TEL 612 836 1987 FAX 612 836 1987 TLR 800 836 1987

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

On file - Sous référence

NOTICE

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

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

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

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

AVIS

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

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

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

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

A Blackboard Architecture to Support Network Fault Diagnosis

by

Sami M. Iqneibi, B. Eng.

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

Department of Systems and Computer Engineering,
Carleton University,
Ottawa, Ontario, Canada

May 27, 1992

© Sami M. Iqneibi 1992

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Veuillez lire - Votre référence

Veuillez lire - Votre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-79797-5

Canada

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

*A Blackboard Architecture to Support
Network Fault Diagnosis*

submitted by Sami M. Iqneibi, B.Eng.
in partial fulfilment of the requirements for
the degree of Master of Engineering

Thesis Supervisor

Thesis Supervisor

Chair, Department of Systems and Computer Engineering

Carleton University
May 27, 1992

Abstract

Many blackboard systems have been built in the last fifteen years to tackle complex problems in various domains. This thesis describes a blackboard framework that was developed to support multiple cooperating diagnostic agents working together to diagnose faults in Wide Area Networks. It shows how the framework was adapted to meet the specific needs of the diagnostic agents involved. The thesis describes the overall system structure and focuses on the details of the internal organization of the framework. It also shows how the framework is used by the diagnostic agents to communicate and interact.

Acknowledgments

Special thanks to my thesis supervisors Dr. Nick Dawes and Dr. Bernie Pagurek for their help and guidance throughout the process of this research. Thanks also to the Telecommunications Research Institute of Ontario which provided financial support for this research. Finally, I would like to thank my family and friends for their help and encouragement.

Contents

Abstract	i
Acknowledgments	ii
1 Introduction	1
1.1 Thesis Objective	2
1.2 Organization	2
2 The BNR Network	3
2.1 Network Structure	3
2.2 Network Monitoring and Management	6
2.2.1 Network Management Protocols	6
2.2.2 BNR Network management	7
2.3 Network Faults and Diagnosis	9
2.3.1 Complexity of Diagnosis Process	9
2.3.2 Types of Faults in the Network	11
3 The Blackboard Problem-Solving Model	13
3.1 Major Components of a Blackboard System	14
3.1.1 The Knowledge Sources	15
3.1.2 The Blackboard Database	15
3.1.3 Control	16
3.2 Pros and Cons	16
3.3 A Blackboard System Example: Hearsay-II	17
3.4 Blackboard Application Systems	19
4 System Design	21
4.1 Pegasus System Requirements	21
4.2 Alternative Designs	22
4.2.1 Direct Message Passing Architecture	23
4.2.2 Message Passing Through a Mediator Architecture	24
4.2.3 The Blackboard Architecture	27

5	The Pegasus System	30
5.1	General	30
5.2	The Pegasus Blackboard	31
5.3	The Dragon Simulator	32
5.4	The Various Diagnostic Methods	33
5.4.1	BANES (Basic Analysis in Nested Evidence Spaces)	33
5.4.2	END (Expert Network Diagnostician)	34
5.4.3	LEAPR (Line Element Anomaly Pattern Recognition)	35
5.5	The Master Knowledge Source	36
6	The Pegasus Blackboard in Detail	38
6.1	The Database	38
6.1.1	Status Messages Area	40
6.1.2	Traffic Messages Area	41
6.1.3	Problems Area	42
6.1.4	Requests and Replies Areas	43
6.1.5	Hypotheses Area	45
6.1.6	KSARs (Knowledge Source Activation Records) Area	45
6.1.7	Areas Management and Usage	46
6.2	The Control Data Structures	51
6.2.1	List of Available Records	53
6.2.2	Mutual Exclusion Semaphore	54
6.2.3	List of Interested KSs	55
6.2.4	List of Recent Changes	56
6.3	Using the Blackboard	56
6.3.1	Getting access to the Blackboard	56
6.3.2	Declaring areas of interest	56
6.3.3	Writing to the Blackboard	57
6.3.4	Monitoring the Blackboard	57
6.3.5	Reading from the Blackboard	58
6.4	A summary of the Blackboard Handling functions	58
6.5	Knowledge Sources Activation	59
6.5.1	The Event Based Activation Method	60
6.5.2	The Master-Slave Activation Method	62
6.5.3	The Hearsay-II Activation Method	63
6.5.4	General	65

7	Research Results	69
7.1	System development	70
7.2	Practical Use	71
7.3	Pegasus Blackboard Performance	73
8	Conclusions	76
8.1	Contributions	76
8.2	Summary of Results	77
8.3	Future Work	77
	References	80
A	Glossary	83

Chapter 1

Introduction

Computer communication networks are very large and complex in nature. The BNR network, for example, covers 4 continents and consists of 3000 elements. The network is difficult to manage not only because those elements are supplied by several manufacturers, but also because the network is administered by numerous organizations. The decentralization of some aspects of the network management sometimes leads to configuration changes which are not reported to all network management centres. All these factors and others make diagnosing the faults in the network a tedious and challenging task. The Pegasus Project at Carleton University, launched in the 1990, provides an innovative approach to the problem of diagnosing computer communication network faults: applying a multiparadigm approach and investigating multiple faults simultaneously. As part of the Pegasus project, my task was to propose an architecture in which the different diagnostic agents can communicate and interact.

1.1 Thesis Objective

This thesis was conducted to achieve the following goals:

1. To conduct a study of the different architectures that can be used to build the multiple paradigm system for network fault diagnosis and to find out the most appropriate one based on the strengths and limitations of each architecture.
2. To build a prototype based on the chosen architecture and to evaluate its performance.

1.2 Organization

The remainder of the thesis is divided into 7 chapters. Chapter 2 describes the BNR Network which was used as a test bed for our system. Chapter 3 describes the blackboard problem-solving model which we followed to build our system. Chapter 4 states the system requirements, enumerates the different design alternatives, and shows why we chose the blackboard architecture. Chapter 5 describes the different modules in the Pegasus System. Chapter 6 gives the details of the Pegasus Blackboard. Chapter 7 describes the Pegasus Blackboard evolution and reports its performance. Finally, the conclusions and future work are presented in Chapter 8.

Chapter 2

The BNR Network

This chapter describes the BNR data communication network which was used as an example of a Wide Area Network (WAN) that can be diagnosed by the Pegasus System. The BNR network is a large network with a wide range of device types. Many kinds of faults occur in the network making it a good choice to carry out our experiments.

The chapter is divided into three sections. The first section describes the network structure. The second section describes the network management protocols common to most WANs and elaborates on the one used on the BNR network. The last section describes the types of faults that can happen in the BNR network and the reasons which make diagnosing such faults a difficult task.

2.1 Network Structure

The BNR network is a packet-switched WAN which provides communication facilities for thousands of workstations which are grouped and connected to several hundred LANs.

Figure 2.1 Workgroups

Figure 2.1 shows the basic components and connections that comprises the BNR network [1]. As seen in the figure, several workstations (up to 11) are connected to the primary hub which in turn is connected to a secondary hub. A number of primary hubs (up to 16) could be connected to a single secondary hub. In BNR terminology, the hierarchical structure of workstations, primary hubs, and the secondary hub is known as a workgroup LAN. The workgroup LAN hierarchy provides connectivity between workstations located under the same secondary hub. From the figure we can notice that

the workgroup concept decreases the number of connections to the LAN Backbone and prevents the local traffic (i.e. traffic between workstations in the same workgroup) from flooding the LAN Backbone.

Workgroup LANs are interconnected by a LAN Backbone through bridges (refer to figure 2.1). This interconnection allows users to communicate with other users and devices not located within the same workgroup LAN. The function of the bridge in this structure is to place the packets received from the secondary hub on the LAN Backbone and to pickup the packets on the LAN Backbone which are destined for workstations below it and deliver them to the secondary hub in the opposite direction.

Since mainframes are important devices in the BNR network, they are directly connected to the LAN Backbone. Furthermore, mainframes are usually connected to more than one LAN Backbone to make them fault tolerant in case one of the interfaces fails.

Finally, LAN Backbones are interconnected together using routers to form Metropolitan Area Networks (MAN) and further interconnected to form a WAN. As illustrated in figure 2.2, a router is usually connected to two LAN Backbones from one side and to two or three other routers through serial lines from the other side. Such connections usually result in a mesh network which provides multiple paths between the different LAN Backbones which makes them robust to device and connection failure. Furthermore, each router in the BNR network (as in most other WANs) is equipped with a routing algorithm which is used to dynamically choose the route that should be taken by each packet received by the router to balance the load on the different lines. This process

Figure 2.2 Routers

increases the network efficiency and provides it with error recovery capabilities but at the same time adds to the complexity of fault diagnosis.

2.2 Network Monitoring and Management

2.2.1 Network Management Protocols

The need for a comprehensive network management protocol was not felt until the

late 1980s when the Internet network growth exploded [2]. Before that, network experts used the Internet Control Message Protocol (ICMP), originally designed to allow hosts and gateways to exchange control messages, to diagnose faults. Moreover, ICMP was used to build many sophisticated packages such as the Packet Internet Groper (PING) and the Packet Filter. With the fast network growth lots of inexperienced operators joined the network management staff and found difficulty in using those tools.

In 1988, the Simple Network Management Protocol (SNMP) was accepted as a network management protocol for the Internet. In the SNMP protocol, network elements are divided into managing and managed systems. Each managed system is abstractly represented by an object database, called Management Information Base (MIB). A management application, which runs on a managing system, monitors and controls the managed system by reading and writing the objects in the database. Each database is associated with a management agent which maintains it and translates the operations on it into real operations on the corresponding managed system.

SNMP uses the proxy approach to manage special devices. Special devices, such as bridges and modems, can't support management agents to interact with management applications because they don't implement the whole TCP/IP protocol suite. To solve this problem, special devices are managed through management agents running on a nearby host (called proxy agents).

2.2.2 BNR Network management

The BNR network is centrally managed from the Network Operations Centre (NOC). At this location, a graphics display program (XNETMON) is used to display the network

General Format:

```
Year/Month/Day  Time  STATE  Device name  State value
```

Example:

```
91/01/01      00:28:33  STATE  12.nRTPc2      15
```

Where State values can be:

- 2 This device is now responding ok.
- 12 This device has not replied to 1-9 successive requests.
- 13 This device has not replied to 20 successive requests.
- 14 This device has not replied to 30 successive requests.
- 15 This device has replied that its communication functions are broken.

Figure 2.3 General format of status messages

status. XNETMON uses as sources of information [3] three text files describing the network topology, and the status messages sent by data collector devices (DCDs) to report changes in the condition of the network elements.

A number of DCDs are distributed around the network. Each one of them is responsible for monitoring a number of network elements. DCDs accomplish their mission by periodically polling those elements (average time between polls is dependent on the element type). DCDs use PING to interrogate mainframes and SNMP to interrogate other devices (e.g. bridges, routers, ... etc.). Any anomaly detected is reported to XNETMON in the form of a status message (refer to figure 2.3). It is worth noting that the polling messages, the corresponding replies, and the status messages are transmitted over the same network being monitored. This transmission is not error-free, adding to

the complexity of the diagnosis process. For example, a broken bridge in the network will make XNETMON display all the workstations below it as broken because they are unreachable.

Many tools are available to the operators to assist them in diagnosing faults. For example, the Remote Diagnostic Access (RDA) Tool, illustrated in figure 2.4, can be used to remotely monitor the traffic exchanged between any two workstations. The RDA tool consists of a matrix switch (connected to a secondary hub) and a PC equipped with analyzer software. An operator can use the PC in the NOC to remotely control the matrix switch and analyzer software through a dial-in modem and analyze the traffic exchanged between two workstations.

2.3 Network Faults and Diagnosis

2.3.1 Complexity of Diagnosis Process

There are several factors that make diagnosing faults in the BNR network a tedious and challenging task:

1. **Network size:** The BNR network is fairly large. It contains thousands of elements and interconnects several hundred LANs.
2. **Amount of status messages:** A single fault in the network might generate a large number of status messages which makes tracing and analyzing them a real challenge to the network operator.
3. **Unreported changes to network topology:** Although the whole network is centrally managed from the Network Operations Centre, LAN administration is not. Changes

RDA tool

Figure 2.4 XNETMON and the Remote Diagnostic Access (RDA) tool

to the network topology may not be reported to the NOC complicating the diagnosis process by distracting the operators attention with unreal faults (e.g. a user phones to

complain that he can't reach some destination which was removed from the network) and forcing them to work with an inexact model of the network.

4. **Using the same network to transmit status messages:** A DCD might mistakenly assume that some devices are down because they are behind a broken network element which makes them unreachable.
5. **Interleaving symptoms of multiple faults:** Multiple faults are difficult to diagnose not only because they rarely happen but also because operators tend to diagnose faults with the assumption that they are single faults (which is true in most cases).
6. **Dynamic routing and error-correcting protocols:** Although dynamic routing and error-correcting protocols are inevitable to support automatic recovery when a device goes down, it makes reproduction of the circumstances in which an error occurred impossible.

2.3.2 Types of Faults in the Network

Many types of faults can occur in the network because of its complex nature and continuous growth. The network is expected to perform complex functions on each packet transmitted through it to make sure that it is delivered correctly in the shortest possible time and, at the same time, to maximize the network utilization and fault tolerance. All these requirements are met by using sophisticated devices which are controlled by huge distributed software programs. Those devices, programs, and the interaction between them is rarely error-free. Furthermore, adding new types of devices (i.e. to expand the network or to increase its capacity) and updated versions of the software (i.e. to add more functionality to the network or to fix some bugs in the earlier version) makes the

problem even worse by introducing compatibility issues. For the above reasons, many types of faults occur in the BNR network, for example:

1. **User can't reach destination:** Operators usually receive several phone calls a day from users who are unable to reach some destination. Many reasons can cause this problem: using the wrong protocol, using a wrong address, or a hardware failure (i.e. connection or device down) on the path between the user's workstation and the destination.
2. **Software incompatibility:** Software upgrades in large networks like the BNR is usually done incrementally. The coexistence of old software and new software in the network sometimes introduces faults because unexpected errors might happen when they interact.
3. **DCD failing:** This type of fault is usually accompanied with the apparent failure of an entire domain of the network. However, this symptom is not unique to this type of fault, it might be also caused by the failure of an essential router in the network.
4. **Intermittent faults:** There are many classes of intermittent faults. For example, one class is caused by a loose connection to some device in the network (e.g. file server). This fault will cause XNETMON to show its status as alternating between OK and NOT OK. Another class of intermittent faults is caused by network congestion which, in many cases, will affect the availability of a certain service.

Chapter 3

The Blackboard Problem-Solving Model

The blackboard model of problem solving has been found to be very useful to tackle ill-defined complex problems. The model's extreme flexibility makes it an excellent research environment to conduct an experimental approach for solving such problems. It has been applied in many domains to perform various tasks, for example: Hearsay-II [4] for speech recognition, HASP/SIAP [5] for ocean surveillance, ACAP [6] for image understanding, and so on. The model prescribes the organization of knowledge and data and the problem-solving behavior within the overall organization.

The following metaphor is usually given to illustrate the basic ideas embedded in the model and its general behavior [7]: imagine a group of human experts working on a problem described on a blackboard in front of them. They cooperate on solving the problem without talking to each other directly. The only way for them to communicate and interact is through the blackboard. Each expert observes the changes made by other experts to the solution state on the blackboard and comes forward to make his own

Figure 3.1 Major components of a blackboard system

changes whenever he can contribute. This process continues until a solution is found.

3.1 Major Components of a Blackboard System

A blackboard system consists of three major components [8] (refer to figure 3.1):

1. **Knowledge Sources (KSs):** The knowledge needed to solve the problem is divided into independent modules, called knowledge sources.
2. **The Blackboard Database:** a global database containing the problem-solving state data (e.g. initial data, intermediate solutions, and final solutions). Knowledge source modularity and independence is preserved by forcing the communications between them to go through the blackboard database.
3. **Control:** The control component dynamically decides the course of the problem-solving process by allowing the most promising KS to execute.

3.1.1 The Knowledge Sources

Each knowledge source captures part of the knowledge needed to solve the problem. The internal representation of that knowledge might not be the same in all KSs. As a matter of fact, one KS might be implemented by rules, another KS by procedures, and yet another by neural-network techniques. KSs don't have to know about the internal representation of each other. However, each one of them should be able to understand the blackboard changes made by the others. Furthermore, it should know the conditions under which it can contribute to the solution.

3.1.2 The Blackboard Database

The blackboard database is a global database containing the problem-solving state data. This database is used by the KSs to communicate with each other and incrementally build a solution. Each KS takes its input from the blackboard in the form of raw data, intermediate solutions or control information, processes them, and deposits its output to the blackboard by either adding new information or modifying old information. A change in blackboard information may trigger other KSs to execute. The process continues until a solution is found.

In most cases, the data on the blackboard was divided into levels and organized as a hierarchy. The nature of the hierarchy and number of levels depended on the problem being solved. In some blackboard systems, another step was taken to structure the blackboard database by organizing the data on it into several hierarchies (known as blackboard panels).

3.1.3 Control

The sequence of KS invocation is not pre-programmed into any blackboard system. It is dynamically determined at run time based on problem-solving state. The control component incrementally builds the solution of the problem by forcing the system to iteratively perform the following steps:

1. A knowledge source is executed. This KS makes some changes to the blackboard.
2. Each KS looks at the new changes and declares the contribution it can make and the estimated cost.
3. Based on those declarations, the control module chooses the most appropriate KS for execution.

3.2 Pros and Cons

The blackboard architecture is not well suited for simple, well-defined, and / or sequential problems because the architecture's major features of dynamic control, flexibility and modularity becomes obsolete for such problems. For example, no one will ever think to use a blackboard architecture for array sorting or word processing. In many cases, researchers initially used a blackboard architecture to understand a problem which they knew little about and later used other more efficient architectures to tackle it.

The advantages of using a blackboard system are numerous:

1. The modularity of the KSs allows the removal, modification, and addition of some KSs without changing the other KSs. This property makes it possible to increase the performance and broaden the capability of the blackboard system easily.

2. Also, the modularity of the KSs allows several developers to work on the system simultaneously. Each KS could be developed and tested separately before being integrated into the whole system.
3. Unlike conventional AI techniques which allow only one knowledge representation for the whole system, the blackboard system permits the system developers to use different forms of knowledge representation for each of the KSs. In some cases, this property allows the system to cover a wider range of problems. In other cases, this allows the system to solve a problem more efficiently and accurately by applying the most appropriate technique for each step of the solution.
4. Separating the knowledge-application strategy from the knowledge sources (by placing it in a control module) allows the developers to experiment with different configurations easily without having to modify the KSs.

3.3 A Blackboard System Example: Hearsay-II

The Hearsay-II is given here as an example to clarify the basic ideas in the blackboard model. The Hearsay-II speech-understanding system [4, 9, 10, 11] was the first blackboard application to be built. Furthermore, the basic blackboard model was constructed by abstracting features in it.

The task of Hearsay-II was to understand speech utterances in the form of queries about a collection of computer science abstracts. Such a task is very complex because of errors in the input data and the imprecise and unknown rules of comprehension.

The blackboard database of Hearsay-II was divided into eight levels of analysis:

parametric (lowest), segmental, phonetic, phonemic, syllabic, lexical, phrasal, and conceptual level (highest). Partial and complete interpretations of the spoken utterance were distributed on those levels in a hierarchical structure of hypotheses (blackboard elements). In this hierarchy, an element on one level is created by combining several more-detailed elements on the adjacent lower level.

All hypotheses on the blackboard database are represented as a uniform structure of attribute-value pairs. The relationships between them were formed using AND/OR links. Using the OR links allowed the system to maintain alternative solutions on the database. Due to the uncertainty of the KSs, the number of alternatives were quite large. The search space was substantially reduced by applying constraints when generating new hypotheses and exploring only the most promising solutions.

Each knowledge source had two major components: a precondition part and an action part. The precondition part searched the blackboard for hypotheses that can be processed by the action part and passed them to it. When executed, the action part processed those hypotheses and made the appropriate changes to the blackboard to integrate its results. Both parts were written as procedures and did not execute until selected by the scheduler.

The Hearsay-II contained two control components to provide a mechanism for activating knowledge sources and selecting the most promising activity for execution: a blackboard monitor and a scheduler. The blackboard monitor kept a record of all changes made to the blackboard and based on their types placed pointers to precondition parts that are interested in them on the scheduling queue. The scheduling queue were also accessed by the precondition parts (when satisfied) to place pointers to the corresponding knowledge sources. In addition to the pointers, the precondition parts provided stimulus

and response frames to describe which hypotheses the action part would process and what changes the action part would produce (when executed). The scheduler iteratively choose the most promising activity, either a triggered precondition or an invoked knowledge source, on the scheduling queue and executed it. The priority of carrying out an activity was determined by the scheduler based on the stimulus/response frames¹ and the solution state.

3.4 Blackboard Application Systems

In the previous fifteen years, many blackboard systems were built to perform various tasks. The HASP/SIAP system [5], for example, interpreted multiple continuous streams of acoustic signals generated by multiple hydrophone arrays. Its task was to track the movement of ships and submarines in a region under surveillance. The knowledge in the KSs were represented as a set of rules. The control component in the system was distributed among a number of control modules which were hierarchically organized. Those modules iteratively decided which event to pursue next and executed the appropriate knowledge source based on its type.

The ACAP system [6] analyzed complex aerial photographs and identified objects in them (e.g. cars, houses, roads, ...). The quality of those photographs depended vastly on the weather condition. Each knowledge source in the system consisted of a single rule. The condition part of the rule served as the precondition of the knowledge source and its action part served as the action part of the knowledge source. The control module was very simple since the system executed all the knowledge sources whose precondition parts

¹ Also known as knowledge source activation records (KSAR).

were satisfied. However, when a region was recognized differently by several knowledge sources, the control module calculated a reliability value for each of their outputs and accepted only one of them.

The system built by Frontini [12] used a blackboard architecture to diagnose faults on the HPNet wide area network and to recommend the remedial actions to correct them. To accomplish its task, the system used the structure of the network and the log events reported to the central managing node. The problems that could happen in the network were well understood to the extent that the events associated with each problem were defined to the system. The task of the system was slightly complicated by the interleaving of events related to independent problems. The problem was relatively simple and appears to not really require the use of a blackboard architecture. As the authors mention, the application did not require any sort of planning or dynamic control and could have been built using a conventional approach. It appears as if the blackboard was used as a pipeline which helped to diagnose several problems at different stages.

To summarize the section, all the systems mentioned above and most other blackboard systems (a number of them are described in [13]) employed only one internal representation for the domain knowledge in the knowledge sources (either procedures or rules). Furthermore, all computations in the knowledge sources were CPU-intensive and the knowledge sources themselves terminated (i.e. returned control to the control component) after making their contributions to the blackboard.

Chapter 4

System Design

4.1 Pegasus System Requirements

Since the problem of diagnosing computer communication networks faults is complex (as shown in subsection 2.3.1) and the number of fault types is relatively large (refer to subsection 2.3.2), the following features are necessary in any proposed system:

1. A group of developers will be working on the project at the same time, so the system must be divisible into independent modules. This way each module can be developed and tested separately and then integrated with the rest of the system.
2. The system should apply more than one diagnostic method to cover a wide range of fault types. Besides the fact that building a system based on one method to solve a complex problem has a very slim chance of success (i.e. many earlier attempts to do so failed), the resulting system is usually quite large and difficult to maintain. The same faults may be diagnosed more efficiently and logically using a multiple paradigm system.
3. The system should have an open architecture (i.e. allow the addition of new

diagnostic methods) since many of the diagnostic methods will be developed later. For example, experimenting with the system might reveal poor performance with regard to diagnosing a certain type of faults when it happens to be that method X is excellent in diagnosing those faults. It would then be very desirable to integrate method X to the system easily.

4. The system should allow experimentation with various configurations (i.e. the way in which the various modules of the system interact) since the exact way in which the various modules will interact with each other is not known yet.
5. Since many of the diagnostic procedures will require operator assistance or the conduct of remote tests, the system should be capable of diagnosing several problems simultaneously to minimize the CPU time wasted while waiting for the result of a remote test or an operator response.
6. The system will monitor and diagnose faults on a live WAN, so it should be fast enough to fulfill the real-time¹ requirement and diagnose faults within minutes of their occurrence. Also, a fast system will accelerate the process of developing and debugging the various modules by testing them against the Dragon simulator [14].

4.2 Alternative Designs

After recognizing and stating the system requirements clearly as in section 4.1, we went on to develop an appropriate system structure to satisfy all the requirements.

We decided to break up our system into several independent modules to allow several developers to work on the project at the same time (i.e. to satisfy requirement # 1).

¹ Not real hard time but reasonable time.

Preferably, each module would be implementing a different type of diagnostic method (i.e. to satisfy requirement # 2). We also decided to implement each module as a separate process to benefit from the multitasking environment of the UNIX platform. This decision speeds up our system substantially by preventing the system from waiting for the result of an external test or an operator response (i.e. satisfies requirement # 5). We avoided using the UNIX light weight process library (LWP) to maintain maximum flexibility in the system (i.e. restrictions such as all light weight processes, i.e. threads, should be created by forking a single process and all threads should share the resources of a single process).

After taking the above decisions, we were left with three architectures to choose from: direct message passing architecture, message passing through a mediator architecture, and the blackboard architecture. A brief description of those architectures is given in the following subsections. Furthermore, by reading the subsections, it will become apparent that the following reasons convinced us to use the blackboard architecture:

1. **Flexibility:** One of the main features of the blackboard architecture is that it allows system developers to experiment with different configurations very easily.
2. **Speed:** System speed is less critical than its flexibility, but still it should present its diagnosis in a reasonable time (i.e. minutes not hours).

4.2.1 Direct Message Passing Architecture

In this architecture, different processes share data and synchronize by sending messages to each other. Each process will have its private message queue on which it receives messages from all other processes (refer to figure 4.1). Therefore, whenever a process wants to share a piece of data with others, it sends a copy to each one of them.

Figure 4.1 Direct Message Passing Architecture

This architecture is preferred by many developers since it is well established, supported on most operating systems, and easy to program because there is no variable sharing between the different processes. On the other hand, this approach does not facilitate experimentation since messages are directed to processes. This means that whenever we want to try a new configuration (i.e. different interaction between the processes) or add a new process, we will have to edit the code of all processes and make the appropriate changes (i.e. requirements # 3 & 4). An inefficiency is introduced by sending multiple copies of the same data to all concerned processes (i.e. requirement # 6).

4.2.2 Message Passing Through a Mediator Architecture

In this architecture, different processes share data and cooperate on solving problems by sending messages to a single well-known process (i.e. mediator) which in turn

Figure 4.2 Message Passing Through a Mediator Architecture

distributes those messages to other processes (refer to figure 4.2). Assuming that we want to allow the exchange of different types of messages between the processes (e.g. one type to describe problems, another type to describe hypotheses, and so on), the architecture could be implemented in two ways: In the first one, the mediator process sends all the messages it receives to all processes in the system and those processes discard the messages with an undesired type. In the second way, the mediator process sends the messages it receives to interested processes only and thus reduces the overhead involved. Of course, this will require the processes to send messages to the mediator process to specify which message types they are interested in receiving. In both ways,

the architecture could be either built on top of the conventional UNIX sockets¹ [15] or the more recent message queues structures [16] (supported on system V).

In the case of using sockets and assuming that we want the processes to selectively receive messages based on their type², every process including the mediator process, will have a number of sockets equal to the number of message types to receive messages on them and a similar number of sockets to send messages. The same results could be achieved using one message queue since the UNIX operating system provides programmers with primitives to selectively receive messages based on their type. In both cases, there are three ways for a process to wait for a message to arrive:

1. Waiting for one type of message to arrive on one of the sockets indefinitely: this could be accomplished by calling a blocking receive primitive. This way works only if the programmer knows that the process will receive this type of message before receiving any other type.
2. Periodically polling all the sockets to receive any message: this way is better than the previous one in that the process will receive any message when it arrives regardless of its type. However, polling is never recommended because of the overhead involved.
3. Assigning a private semaphore for each process: each process will wait for messages on its private semaphore until it is signalled. After this signal, the process can start polling all the sockets to receive the message sent.

One problem remains unsolved: what if a process wants to receive messages in a different order than they were sent in? This problem arose because of the sequential

¹ Similar to the UNIX pipes, but more flexible

² A situation that might arise when a process is waiting for a reply or the result of an external test

nature of the sockets and message queues. It could be solved by implementing an internal buffer in each process.

This architecture is more flexible than the direct message passing architecture since all processes have to know only about the existence of the mediator process. Furthermore, if the mediator process is properly coded with a generic algorithm (i.e. so that it can communicate with any number of processes, just by receiving a message from them to specify their message queue id), adding a new process to the system will require absolutely no change to the code of the mediator or the existing processes.

This flexibility came at the expense of the system performance. The addition of the mediator process not only increased the number of messages that has to be exchanged in order to share data but also increased the overhead required to do so by activating (i.e. context switching) the mediator process to convey the messages from the senders to the receivers.

4.2.3 The Blackboard Architecture

The blackboard problem-solving model was reviewed in Chapter 3. An architecture based on this model was a perfect candidate because the problem of diagnosing faults was complex and, more importantly, ill-defined. We knew very little about the number of modules that we will need to solve the problem, the strengths and weaknesses of each module and how they will fit together in a multiple paradigm system. However, we knew for sure that we needed a very general and flexible framework which will allow us to take an experimental approach for solving the problem.

One major difference between the blackboard architecture and the previous archi-

tures is that data is not directed to any specific process (i.e. knowledge source). A knowledge source in the blackboard architecture takes its initial data from the global database, processes that data, and puts its output back to the global database. In doing so, the knowledge source does not care who provided the initial data or who will benefit from its output and thus provides a very high degree of flexibility in terms of adding, removing, and modifying knowledge sources. Another major difference is that the previous two architectures treat all the modules in the system as peers, whereas in the blackboard architecture one knowledge source or more could be marked as a control KS (or a scheduler) and could be used to control the activities in the system. Therefore, it allows the system developers to experiment with different configurations and interactions between the knowledge sources very easily.

As far as speed is concerned, the blackboard architecture replaces the mediator process by the global database to provide communication between the processes and thus eliminates the overhead of context switching the mediator process. Table 4.1 summarizes the comparison between the three architectures.

The Pegasus blackboard is different from other previously implemented blackboards in many ways. Other blackboards used only one form of knowledge representation to implement the knowledge sources, whereas our system allowed the use of more than one. Our system executes several KSs concurrently and investigates multiple faults at the same time. We couldn't wait for a KS to finish before executing the next KS because some of them never terminated and others were not CPU-intensive.

The implementation details of this architecture are described in chapter 6.

	<i>Direct Message Passing</i>	<i>Message Passing Through a Mediator</i>	<i>Blackboard Architecture</i>
system divided into modules	Yes	Yes	Yes
uses multiple paradigms	Yes	Yes	Yes
has an open architecture	No	Yes (if properly implemented)	Yes
allows experimentation with various configurations	No	Yes (to some extent)	Yes
concurrently diagnoses several problems	Yes	Yes	Yes
relative speed	faster than message passing through a mediator	Slowest among the three architectures	faster than message passing through a mediator
relation between processes in the system	peers	peers	peers + control KSs

Table 4.1 A comparison between the three alternative architectures

Chapter 5

The Pegasus System

5.1 General

The ultimate goal of the Pegasus system is to automate the process of diagnosing WANs. The system is built to cover a wide range of faults by applying a multiparadigm approach [17]. The system is organized as a blackboard architecture for the reasons given in chapter 4. The final product will be installed in the network operations centre where the status and traffic messages are gathered. In addition, all the problems reported to the operators will also be entered to the system.

The system consists of the following main components:

1. The Pegasus blackboard: a framework based on the blackboard problem-solving model which is used by the diagnostic agents to communicate and interact.
2. The Dragon simulator: a software program to simulate the sample network. The simulator is used to test the various diagnostic methods in the Pegasus system.
3. Some knowledge sources:

- a. BANES [18]: this KS generates hypotheses about the network elements' status by analyzing the status messages and relating them to the network configuration.
 - b. END [19]: a rule-based KS used to verify BANES hypotheses and to investigate the problems reported to the operators.
 - c. LEAPR [20]: a KS based on neural networks technology is used to monitor the network to detect performance degradation or any abnormal traffic.
 - d. Phoenix: a model-based KS which uses a modified version of the Dragon simulator as the network model.
 - e. others: KSs using case-based and distributed reasoning methods are also under investigation.
4. A Master knowledge source (or scheduler): a special KS in the Pegasus system which is acquainted with the network status and the activities going on in the system. Its main task is to control the activation of the other KSs in the system to enhance its performance (yet to be developed).

5.2 The Pegasus Blackboard

We have chosen to use the blackboard architecture to facilitate the communication between the different diagnostic methods. In this framework, the different KSs can share information and cooperate through a global database. We also chose to implement the blackboard in the C language as opposed to LISP because it allows us to integrate different languages to the system (i.e. Pascal and COSMIC CLIPS). Each diagnostic method is implemented as a separate process which gives us maximum flexibility in

choosing the knowledge representation for each of them. Also, it allows the system to investigate multiple problems concurrently. As in most other blackboard applications, our blackboard is partitioned into several areas. Efficiency is increased by allowing selective unblocking based on the primitive type of change made to the blackboard. The efficiency is a prime factor in our design since the system should meet the real time requirements of the network.

5.3 The Dragon Simulator

The Dragon Simulator [14] is a Pascal program to simulate packet switched Wide Area Networks such as the BNR network. It was built in the Pegasus group to enable the testing of the Pegasus System. The design of Dragon reflects this fact by supporting most of the external functionality of the simulated network, allowing the introduction of various types of faults and supporting the network management protocols. One main feature of the simulator is its high speed when simulating large networks which was made possible by avoiding the actual transmission of real packets between the network nodes.

The simulator determines the network topology by reading three text files. In those three files, a description of the name, type, and connections to each device in the network. Many device types are supported such as workstations, mainframes, bridges, routers, ethernet, ... etc. In addition, some of the workstations can be designated as data collectors (refer to subsection 2.2.2) to monitor some of the network devices and produce status messages.

Many types of faults and their symptoms are simulated, such as:

- Random corruption of packets.
- Breaking and repairing devices either randomly or using the operator interface.
- Phone calls from users to report a fault.
- abnormal traffic generated by a certain workstation.

The simulation course could be controlled in two ways:

1. Using the operator interface: A user can introduce faults, generate traffic, examine routing tables, ... etc. by selecting one of the items on the simulators main menu.
2. Using the blackboard interface: this interface allows processes in the Pegasus system to query the network devices being simulated and perform tests on them: queries such as what is the state of a certain element or what is its monitoring protocol. Tests such as does a remote login succeed to a certain element or does a reboot fix the fault in a device.

5.4 The Various Diagnostic Methods

5.4.1 BANES (Basic Analysis in Nested Evidence Spaces)

The diagnostic method used in BANES was developed in the Pegasus group. The method uses a model of the system being diagnosed and the symptoms associated with the fault to pinpoint the device(s) causing the problem [18]. In the case of diagnosing faults in the computer communication networks, the model corresponds to the network topology, the location of data collectors and the lists of devices monitored by each data collector. The symptoms correspond to the status messages reported to the Network

Operations Centre. BANES uses these two sources of information to propagate beliefs in an internal structure known as Nested Evidence Space [21].

The main advantage of this technique is that the complexity of computations is linearly related to the number of devices in the network. This feature makes it extremely fast compared to other methods when working on large networks. Unlike human experts, BANES performs extremely well when the network is complex and the number of status messages is large.

Finally, the hypotheses produced by BANES are verified by the expert system END before being displayed to the operator.

5.4.2 END (Expert Network Diagnostician)

The END knowledge source is based on expert systems technology [19]. Rule-based systems are considered one of the most well understood ways of representing knowledge because it usually involves the direct translation of the human experts' knowledge into rules. Several network operators and experts were consulted to extract the diagnostic procedures that they apply when confronted with various types of faults. Then, those procedures were written as rules using the COSMIC CLIPS shell into a prototype. After that, the system went through several refinement cycles.

The first main task of the END knowledge source is to verify hypotheses generated by other diagnostic methods (e.g. BANES and LEAPR). The second main task is to investigate problems reported by users through phone calls.

The final prototype of the END knowledge source is divided into four independent modules. The Veri-element and Veri-line modules verify BANES hypotheses about

elements¹ and lines status respectively. The task of investigating problems reported by network users is given to the General Network Problem Locator module. And the last module, Network Overload Tracer, finds the source of heavy network traffic which causes problems.

Although the diagnostic procedures conducted by END are very accurate, they usually involve a large overhead in the form of external tests (e.g. send an SNMP request to the suspected device or even try to remote login to it).

5.4.3 LEAPR (Line Element Anomaly Pattern Recognition)

One main feature of computer communication networks is their high reliability due to the redundancy of devices and their automatic recovery capabilities. But this same feature might hide gradual network performance degradation until it reaches an unacceptable level felt by network users.

The LEAPR knowledge source was introduced to the Pegasus system to detect network performance degradation in the early stages before it interferes with the users' work [20]. It employs a back propagation neural network implemented in the C language. Furthermore, it takes the traffic statistics on each line of a router as input and produces a hypothesis about the health of that router as an output. A hypothesis produced by LEAPR will usually activate the END knowledge source to further investigate the problem.

The neural network is trained to classify traffic into normal and abnormal by exposing it to examples classified by computer communication networks experts. The main problem in such training is that the network traffic is highly dynamic and its classification

¹ An element being a workstation or a router.

is dependant on the time and the day in which it was taken (e.g. traffic on Sundays is completely different than traffic on Mondays)

5.5 The Master Knowledge Source

The need for a high level knowledge source that has a general idea about the network status and the activities in the Pegasus system is obvious (to date, it has not been developed). The main task of this KS will be to increase the cooperation between the various diagnostic methods. Hopefully, this will result in more comprehensive fault coverage and an increase in the Pegasus system efficiency. The efficiency could be increased by controlling the activation of the KSs. It is hoped that the master KS will choose the most appropriate diagnostic method to diagnose a certain type of faults. Covering more faults could be achieved by orchestrating more than one diagnostic method to incrementally solve the problem.

The master KS will perform other tasks, such as:

1. resolving conflicts between several hypotheses posted by different KSs about the same network element: The master KS may use heuristic rules or conduct additional tests to determine the actual status of the network element.
2. prioritizing problems so that problems involving important network elements are investigated first: The fact that the system can investigate multiple problems concurrently makes the decision of when to start investigating a certain problem a bit complicated. The master KS might decide to investigate lower priority problems to increase the system utilization (i.e. a balanced mixture of CPU-intensive investiga-

tions and investigations requiring external tests usually gives the best results).

Chapter 6

The Pegasus Blackboard in Detail

The major components of the Pegasus Blackboard are shown in figure 6.1, where:

1. The knowledge sources (KSs): a group of processes working together to diagnose the faults in the network.
2. The blackboard database: a global database used by the KSs to communicate and interact.
3. The blackboard handling functions: functions available to the KSs to access the blackboard in an organized fashion.
4. The control data structures: data structures used by the blackboard handling functions to manage the blackboard database and to make the communication between the KSs possible.

6.1 The Database

There are three types of data that can go into the blackboard database:

Figure 6.1 Pegasus' Blackboard Architecture

1. Raw data: this type of data contains the information gathered from the network, such as status messages and traffic data messages.
2. Activities: the data in this type usually requires an action to be done by a certain KS. The activities type is divided into three subtypes:
 - Problems: this subtype is used by the operator to describe problems in the network.
 - requests: this subtype combined with the replies subtype provides a message passing capability that allows KSs to query each other

- Knowledge Source Activation Records (KSARs): this subtype was created to give master KSs some control over the activation of the slave KSs (How that is done is explained later).
3. Knowledge sources output: the output could be a hypothesis to explain an abnormal behavior in the network (i.e. fault) or a reply to a previously submitted request.

We decided to divide the blackboard database into 7 areas, namely: status messages, traffic messages, problems, requests, replies, hypotheses, KSARs (these are described in detail in subsections 6.1.1 – 6.1.6). Each area can take one of the subtypes mentioned above. Dividing the blackboard serves the following purposes:

1. Speeds up the search process: by dividing the blackboard, a KS can search areas of interest only.
2. Enhances the utilization of the memory resource: a record structure in each area can be customized to fit the kind of information that will go in it instead of having one record structure that can accommodate all the kinds.
3. Reduces number of times each KS is activated: the number is substantially reduced by asking the KSs to declare the preliminary changes they are interested in to the blackboard handling functions. Then a KS will be activated only if a change is made to one of his areas of interest.

6.1.1 Status Messages Area

At the present time, the BNR network management relies heavily on the status messages reported by the data collectors to detect faults in the network. Data collectors monitor the status of the network elements by continuously polling them. If they detect

any change in their status, they construct a status message and send it to the Network Operations Centre (NOC). When status messages arrive at the NOC, they are promptly displayed to the network operators. If the status message(s) indicate that there is a fault in the network, the operators initiate a diagnostic procedure to further investigate the problem.

At the current stage of system development, the network, the faults, and the corresponding status messages are simulated inside the Dragon simulator. The simulator continues to simulate the transmission of data packets through the network until a status message arrives at the simulated NOC. At this point, the simulator displays the status message to the screen. Furthermore, the Dragon simulator accesses the blackboard asynchronously and posts a status message record to the status messages area. The Dragon simulator runs in the Pegasus system as a separate process. It could be thought of as a KS which runs indefinitely and needs no events to be triggered.

After the Pegasus system is developed and tested, it will be installed in the NOC. The Dragon simulator will be replaced by the real network. The status messages arriving to the NOC will be handled by an I/O module. The I/O module will receive the status messages sent by the DCDs, convert them to the blackboard status message format and asynchronously post them to the blackboard.

The general structure of the *status_msg_record* is shown in table 6.2 on page 47.

6.1.2 Traffic Messages Area

We realized that we need to collect the routers' traffic and error counts in order to capture any network performance degradation. In the BNR network, those statistics are

already being collected periodically from the routers and stored in the data collectors. Therefore, we can ask the data collectors to send those statistics to the Network Operations Centre (This process might be impractical if the overhead involved is unacceptable). The area is currently implemented on the blackboard but isn't used by the simulator or any KS. KSs requiring this type of statistics now use the requests and replies areas to get what they need from the simulator (e.g. LEAPR).

The general structure of the *traffic_record* is shown in table 6.3 on page 47.

6.1.3 Problems Area

All the problems in the network that need investigation are posted to this area. Most of the problems are detected by users and then reported to the operators in the Network Operations Centre. The area could be used in two ways: First, it could be used by the network operators to instruct the Pegasus system to investigate problems reported by users. For example, a user might call an operator because he couldn't remote login from his workstation to a certain host in the network. In this case, the operator would enter the problem to the Pegasus system by writing a *problem_record* to the blackboard which in result will initiate an investigation to explain it. Second, it could be used by any of the KSs in the system to ask other KSs to investigate problems in the network. For example, the General Network Problem Locator (GNPL) module acts as a dispatcher for the other three modules of END. To continue the previous example about the user's inability to remote login to a host, posting the problem to the blackboard results in activating the GNPL module. After looking at the problem, the GNPL module asks the Veri-line and the Veri-element modules to examine the host, the user's workstation, and the elements

and lines on the path between them to pinpoint the device causing the problem.

The general structure of the *problem_record* is shown in table 6.4 on page 47.

6.1.4 Requests and Replies Areas

The requests and replies areas are complementary to each other so they are best described in one place. There are two types of KSs which use those two areas: the client and the server (let us forget at the moment that a KS might serve as both). The client KSs know that a number of services are provided in the blackboard system through the requests and replies areas interface. They don't care who are providing those services as long as their requests are fulfilled. Looking from the other end, a server KS will perform the requests he receives from the requests area and sends back the appropriate reply(s) to the replies area without knowing which client(s) he served. The server KS marks each reply with a unique id (i.e. the *reply_no* field in the *reply_record* is filled with the corresponding *request_record* index) to distinguish it from other replies so that the client KSs could know for whom the reply is meant. A number of servers can coexist in the system as long as each serves a distinct set of request types. Making the client KSs ask for services by specifying the service number as opposed to specifying the server increases the modularity of the KSs and minimizes the changes required to reconfigure the system.

The two areas are currently used by the KSs to query the simulated network and conduct tests on it. They post their requests to the Dragon simulator and receive the results by reading the corresponding replies. On the other end, the Dragon simulator checks the requests area every second of simulated time to carry out the requested actions (refer to figure 6.2.a). Later, when the Pegasus system is installed in the NOC and the

Figure 6.2 a- Usage of requests and replies areas when the network is simulated.

b- Usage of requests and replies areas when diagnosing the real network.

simulator is replaced by the real network, the requests will be carried out by specialized KSs on the real network. These specialized KSs will use standard network management protocols (e.g. SNMP and ICMP) to query the devices in the network and post the results they receive to the replies area (refer to figure 6.2.b).

The general structure of the *request_record* is shown in table 6.5 on page 48. The fields in the *reply_record* are very similar to the ones in the *request_record*. The general structure is shown in table 6.6 on page 49.

6.1.5 Hypotheses Area

The final results of the KSs are posted to this area. Most of the results will be in the form of a hypothesis about the status of a network element. A hypothesis is produced in an attempt to explain a number of status messages received by the Network Operations Centre or to account for a problem reported to an operator. Sometimes several conflicting hypotheses about the same network element will be posted by different KSs. The task of resolving this confliction is left to a higher level KS to be designed later. The general structure of the *hypothesis_record* is shown in table 6.7 on page 50.

6.1.6 KSARs (Knowledge Source Activation Records) Area

After building a blackboard prototype and testing it for several weeks, we found it necessary to add the KSARs area to the blackboard to give master KSs some control over the activation of the slave KSs. Without this area, it would be difficult to coordinate between the different diagnostic methods. Issues such as which problem should the KSs investigate (e.g. priority should be given to investigating problems involving important devices) and when should such investigation start (e.g. based on the current load of the Pegasus system) would be impossible to resolve. The area is basically used by the master KS(s) to specify to the slave KSs the order in which they should investigate the problems and at what level of priority. The role of the KSARs area will become apparent when the

knowledge source activation methods are explained in section 6.5. The general structure of the *KSAR_record* is shown in table 6.8 on page 50.

6.1.7 Areas Management and Usage

Each area is implemented as an array of records of a fixed size. Random access to this array by the different KSs makes no sense since the whole idea behind the blackboard system is allowing the processes to communicate. In other words, just enabling processes to read and write to an area is not enough. The blackboard system should provide a mechanism by which processes can monitor the blackboard and allow them to locate new changes easily. The Control Data structures, described in the next section, and the blackboard handling functions, described in a later section, were created just to serve this purpose.

We couldn't use dynamic memory allocation to implement the areas because they are accessed by multiple processes and each one of these processes has its own memory mapping tables. Therefore, we had to use shared memory segments.

Field Name	Field Size (in bytes)	Description
node name	16	The name of the element for which the status message has changed.
time	4	The time in seconds when the data collector received the status change.
value	4	The new status of the element.

Table 6.2 General structure of the *status_msg_record*

Field Name	Field Size (in bytes)	Description
node name	16	The name of the router for which the statistics are being reported.
time	4	The time in seconds when the the statistics were collected from the router.
traffic count	12 * 4	An array of packet count for each line attached to the router (up to 12 lines) since the router was last booted.
error count	12 * 4	An array of error count for each line attached to the router (up to 12 lines) since the router was last booted.

Table 6.3 General structure of the *traffic_record*

Field name	Field Size (in bytes)	Description
source node name	16	The name of the node were the user is setting or the node suspected to be faulty.
source node id	4	A unique id number associated with the source node which can be handled more efficiently by the system.

Table 6.4 General structure of the *problem_record* (Continued . . .)

Field name	Field Size (in bytes)	Description
source node type	4	Type of the source node which might be a workstation, router, mainframe, ... etc.
problem type	2	The type of problem being reported such as remote login failure, file transfer failure, ... etc.
destination node name	16	The name of the node from which the user is trying to get service (note that some types of problems does not require this field because it involves one node).
destination node id	4	The id of the destination node (similar in function to <i>source node id</i>).
destination node type	2	type of destination node (similar in function to <i>source node type</i>).
parameter 0	4	We decided to include those fields in the record for future use because when I started building the blackboard, many KSs were not finished yet.
parameter 1	4	
parameter 2	2	
parameter 3	2	
parameter 4	2	
parameter 5	2	

Table 6.4 General structure of the *problem_record*

Field Name	Field Size (in bytes)	Description
function no	2	Each service in the system has a unique number. This field specifies which service is required.
string 0	16	These two strings might be used to pass arguments if the service requires so. Most of the services specify <i>string 0</i> as a node name.
string 1	16	

Table 6.5 General structure of the *request_record* (Continued ...)

Field Name	Field Size (in bytes)	Description
parameter 0	4	Those fields might be used to pass any integer argument required by the service.
parameter 1	4	
parameter 2	2	
parameter 3	2	
parameter 4	2	
parameter 5	2	

Table 6.5 General structure of the *request_record*

Field Name	Field Size (in bytes)	Description
reply no	4	This field distinguishes between the various replies by identifying which request is this reply for. Doing so allows us to have multiple clients and servers sharing the same requests and replies areas.
string 0	16	These two strings might be used to pass results back.
string 1	16	
parameter 0	4	Those six fields might be used to pass numerical results back.
parameter 1	4	
parameter 2	2	
parameter 3	2	
parameter 4	2	
parameter 5	2	

Table 6.6 General structure of the *reply_record*

Field name	Field Size (in bytes)	Description
device name	16	The name of the network element for which the hypothesis is for.
device id	4	A unique id number associated with the device to be used internally by the system.
KS name	4	Is the name of the KS who produced this hypothesis.
KSid	2	The id of the KS who produced this hypothesis. This field and the <i>KS_name</i> refer to the same KS. We decided to include both to give the KSs whom will be processing the hypotheses the option to work with whichever they find more convenient.
hypothesized state	16	The current state of the device as concluded by the KS.
confidence	2	Is a number between 1 and 100 which indicates how confident is the KS about the hypothesis.
parameter 0	4	We decided to include those fields in the record for future use because when I started building the blackboard, many KSs were not finished yet.
parameter 1	4	
parameter 2	2	
parameter 3	2	
parameter 4	2	
parameter 5	2	

Table 6.7 General structure of the *hypothesis_record*

Field Name	Field Size (in bytes)	Description
area	2	A KSAR could refer to any area in the blackboard (all the areas on the blackboard are numbered for ease of reference).
record	4	This field gives the record number which should be processed by the KS specified in the KSid field.

Table 6.8 General structure of the *KSAR_record* (Continued . . .)

<i>Field Name</i>	<i>Field Size (in bytes)</i>	<i>Description</i>
priority	2	The priority at which the KS should process the record (in the UNIX environment a process can change its priority by calling a system function).
KSid	2	The id of the KS for which the <i>KSAR_record</i> is meant.
KS type	2	Sometimes the <i>KSAR_record</i> might be meant to a group of KSs with a type given in this field. In this case the KSid is zero.
KSAR status	2	Indicates the status of the <i>KSAR_record</i> (i.e. whether a KS has started working on it or finished processing it).
parameter 0	4	We decided to include those fields in the record for future use because when I started building the blackboard, many KSs were not finished yet.
parameter 1	2	

Table 6.8 General structure of the *KSAR_record*

6.2 The Control Data Structures

Control data structures are needed in the system to manage and control access to the blackboard database. They also facilitate the communication between the different diagnostic agents. Therefore, the control structures could be divided into two groups according to their function:

1. Control structures to manage the areas on the blackboard database: two of those structures are associated with each area:
 - a. list of available records: is used to keep track of empty records in the area. Whenever a record is needed, this list is accessed and the first record in it is returned.

- b. mutual exclusion semaphore: is used by the blackboard handling functions to ensure the integrity of the control data structures.
2. Control structures to facilitate the communication between the different agents: there are four structures of those — two associated with each area (a and b) and two associated with each KS (c and d):
- a. list of interested KSs: this list is consulted when a change is made to the corresponding area to notify the interested KSs only.
 - b. list of recent changes: is used to keep track of last changes made to the area so that a KS can locate new changes quickly by accessing this list.
 - c. information record: the only way that KSs know about the existence of each other is through this record. It is created when the KS registers on the blackboard and contains various fields such as KS name, type, subtype, ... etc. The information records of all KSs are stored in a single shared memory segment as an array called `KS_ARRAY`.
 - d. Private semaphore: each KS waits for changes to the blackboard on his own private semaphore after declaring the preliminary changes he is interested in to the blackboard handling functions. The semaphore is signaled by the blackboard handling functions when a change of interest to the KS is made to the blackboard to notify him.

Figure 6.3 shows a typical area on the blackboard and the associated control data structures.

Figure 6.3 A typical area on the blackboard and the associated control data structures.

6.2.1 List of Available Records

Since we opted to use arrays of fixed size to implement our areas, we had no choice but to build our own memory management functions to manage those areas. Keeping in mind that all addresses used by processes are virtual addresses and an address used by one process is meaningless to another process, we decided to build the list of available records as a linked list of record indexes instead of actual addresses (refer to figure 6.4).

The list is stored in the same shared memory segment as the array it is managing and is protected by the mutual exclusion semaphore of that area (which is described in the next subsection) when accessed. The list is accessed when a KS needs an empty

Figure 6.4 List of available records

record to write data to the blackboard. If the list is empty, the list of recent changes is accessed to reuse the outdated records.

6.2.2 Mutual Exclusion Semaphore

Since the blackboard is accessed asynchronously by multiple processes, we had to use semaphores to prevent processes from reading or writing to the control data structures while another process is updating them. We decided to use a separate semaphore for each area on the blackboard¹ because the overhead involved in creating and manipulating the

¹ The individual semaphores are grouped into one UNIX semaphore set [16] i.e. MUTEX_SEM

Figure 6.5 A typical list of interested KSs

semaphores is not negligible and at the same time we did not want to lock the whole blackboard while it is accessed by a single process.

6.2.3 List of Interested KSs

A list of interested KSs is associated with each area on the blackboard. It mainly consists of the KS_ids¹ of the KSs interested in that area. All lists of this kind are protected by a single semaphore to impose mutual exclusion. The list actually consists of more than the KS_ids to be directly used in a single UNIX semaphore operation to notify all interested KSs (refer to figure 6.5).

¹ Each Knowledge Source is given a unique number at run time, KS_id, which is used by the other KSs in the system to communicate with it

6.2.4 List of Recent Changes

Each area on the blackboard is associated with a list of recent changes. This list is very similar in structure to the list of available records. However, it links all the records recently posted to that area.

This list is used by the different KSs to find the recent changes made to the corresponding area. The records in this list are ordered according to the time when they were posted. Furthermore, the list continues to grow until the system runs out of empty records (i.e. remember that we are using arrays of fixed size). In this case, the system starts to recycle the records in this list in batches.

6.3 Using the Blackboard

We start-up our system by running a process which creates the shared memory segments and semaphore sets. This process also initializes the control data structures. After this point, any number of processes can use the blackboard to communicate with each other.

6.3.1 Getting access to the Blackboard

Any process wishing to use the blackboard must get access to it first. This is simply done by attaching the shared memory segments to the virtual address space of the process and asking the operating system for the ids of the semaphore sets.

6.3.2 Declaring areas of interest

If a KS wants to monitor the blackboard for certain types of changes, he is required

to declare those types of changes to the blackboard handling functions. The blackboard handling functions will add his id to the list of interested KSs. This way KSs are selectively activated when a change of a certain type is made to the blackboard.

6.3.3 Writing to the Blackboard

Since the blackboard database is part of each process' address space, writing is a very easy operation (i.e. it is no different than writing to a local variable). The two things that we should be concerned of are:

1. Ensuring mutual exclusion: this is done using semaphores and the list of available records to prevent two processes from writing to the same record. Whenever a process needs to write some data to the blackboard, it must access the list of available records and take out one record (this step is protected by semaphores).
2. Notify other processes of the change: when the process finishes writing, it adds the new record to the list of recent changes (also this step is protected by semaphores) and signals the interested processes. Notice that this sequence of steps ensures data consistency by preventing other processes from reading until the whole record has been written.

6.3.4 Monitoring the Blackboard

Each KS in the system waits (if he has nothing else to do) on his private semaphore for new changes to the blackboard. This KS will be signaled (i.e. unblocked) by the blackboard handling functions when another KS makes a change to one of his areas of interest. Then, the KS can search the blackboard to locate the new change by accessing the

lists of recent changes (I should mention here that each KS is responsible for remembering the last change he processed in each area so that he can tell, with the assistance of the blackboard handling functions, whether a new change has been added to the list of recent changes). Once the new change is located, the KS can read and process it.

6.3.5 Reading from the Blackboard

Reading from the blackboard is even simpler than writing to it because we do not have to worry about data consistency. This stems from the fact that the record becomes read-only when added to the list of recent changes. The KS can know where the new records have been posted by monitoring the blackboard (as explained above) and then read them as if they were local variables.

6.4 A summary of the Blackboard Handling functions

A set of C functions to manipulate the control data structures and thus control the activities in the blackboard system are provided to the KSs' programmers to simplify the process of interfacing their programs to the blackboard.

1. **get_access_to_bb ()**: this function will attach the shared memory segments containing the blackboard and control data to the address space of the calling process and give it access to the various semaphore sets.
2. **register_KS ()**: a KS calls this function when he starts running to let other KSs know about his existence and to receive the id of his private semaphore.
3. **check_out ()**: this function is provided to allow upgrading the knowledge sources while the system is running.

4. **interested ()**: the KS declares his interest in a certain area on the blackboard to the blackboard handling functions by calling this function. In effect, his id will be added to the list of KSs to be notified when a change is made to this area (i.e. list of interested KSs).
5. **uninterested ()**: this function is called when a KS loses his interest in a certain area. In effect, his id will be removed from the list of interested KSs. Although this function could be used extensively in conjunction with the **interested ()** function to considerably reduce the number of false signals received by the KS (e.g. A KS might temporarily lose interest in all areas except the reply area while waiting for a reply but he will still be awakened whenever a change is made to one of them), I did not encourage such action because it might lead to errors if not carefully coded.
6. **get_empty_area_name¹()**: a KS can use this function to get an empty record in the *area_name*. The direct access of the list of empty records is prohibited since it might lead to corrupting the list.
7. **post_area_name()**: after filling the empty record returned by **get_empty_area_name**, a KS can post the record by calling this function. In effect, the record number will be added to the list of recent changes and all interested KSs will be notified.
8. **get_next_area_name()**: this function is used by KSs to retrieve the record numbers of the records recently added to the list of recent changes.

6.5 Knowledge Sources Activation

The basic activation method supported on the Pegasus blackboard is the event based

¹ *area_name* is the name of one of the seven areas on the blackboard (e.g. *problem*).

activation method. This method in its simplest form will activate all the KSs when any change is made to the blackboard. Then, each KS, working autonomously, will lookup the new change and decide for himself whether he can contribute to the solution by processing the new change or not (such KS is called self-activating KS). Although this method is very simple, it doesn't provide the means by which a master KS (i.e. a KS with a general overview of the solution state — in our case the network status and the activities in our diagnostic system) can control the activation of the slave KSs.

More sophisticated activation methods could be built using the event based activation method in conjunction with the KSAR area (described earlier in subsection 6.1.6). The event based activation method is described in the next subsection. Then a variety of activation methods that can be built on top of it is described in the following subsections.

6.5.1 The Event Based Activation Method

This method is one of the simplest ways to realize the blackboard model. It activates all the KSs when a change is made to the blackboard to let them decide if they want to process the change or not. Each KS will base his decision only on his knowledge about his own capabilities and the type of the change. No coordination between the different KSs is viable using this method.

We have implemented this method with slight modifications to enhance the performance. By dividing the blackboard into seven areas, introducing the list of interested KSs for each area, and asking the KSs to declare the preliminary changes they are interested in to the blackboard handling functions, we managed to considerably reduce the number of times each KS is activated. With these modifications, the KS is activated only if the

Figure 6.6 The event based activation method (assuming that KS#2 and KS#4 are interested in problems area).

change made to the blackboard falls into one of the types he declared earlier.

An example to illustrate the method is shown in figure 6.6. In this example, both KS#2 and KS#4 are interested in the problems area. When KS#1 writes a new problem to the blackboard only those two are activated. The blackboard system (more specifically,

Figure 6.7 The Master-Slave activation method (assuming that the Master KS decides to run only KS#2).

the blackboard handling functions) knows which KSs to notify by accessing the list of interested KSs.

6.5.2 The Master-Slave Activation Method

In this method, a master KS can control the activation of the slave KSs. All the slave KSs wait for commands from the master KS by monitoring the KSAR area. The master KS knows the capability of each of the slave KSs. Based on this knowledge, a master KS can choose the most appropriate slave KS to diagnose a problem or any other type of change.

An example to illustrate the method is shown in figure 6.7. Both KS#2 and KS#4 wait for a KSAR_record to start executing. The master KS monitors the problems area. As a result, he becomes activated when KS#1 posts a new problem to the blackboard. The master KS looks at the new problem and decides to run only KS#2 to diagnose it. He places a KSAR_record with the KSid equal to the id of KS#2 to let it run.

6.5.3 The Hearsay-II Activation Method

This method is similar to the KS activation method of the Hearsay-II system (where it got its name). Here, the KSs are divided into two parts — the precondition part and the action part. The precondition part looks at the changes made to the blackboard and decides whether the corresponding action part could contribute to the solution by processing those changes. The action part, when executed, actually processes those changes and makes the appropriate changes to the blackboard. In addition, a high level KS (called the scheduler) is needed to schedule the activities in the system.

For this method, we need two KSARs areas (say KSARs area#1 and KSARs area#2). The first area is used by the precondition parts to specify to the scheduler which action parts could be executed. The second area is used by the scheduler to activate the selected action parts. Hence, the first area is monitored by the scheduler and the second area is monitored by the action parts.

The precondition part continuously monitors the areas of interest and when it finds something that could be processed by the action part, it places a KSAR in KSARs area#1. This will result in activating the scheduler which will look at this KSAR and decide whether to execute the action part or not based on the network status and the Pegasus system state. If the scheduler decides to execute the action part, it places another KSAR in the KSARs area#2 asking the corresponding action part to process the change.

An example to illustrate the method is shown in figure 6.8. In this example, we assume that KS#2 is specialized in diagnosing some types of problems. When KS#1 action part writes a new problem to that area, KS#2 precondition is activated. Then, KS#2 precondition places a KSAR in KSARs area#1 to indicate that KS#2 action part

Figure 6.8 The Hearsay-II activation method (assuming that posting a problem results in activating KS#2 action part).

could process the newly added problem. This results in activating the scheduler which

Figure 6.9 A possible configuration for the Pegasus system

looks at the KSAR and decides to execute KS#2 action part. Finally, the scheduler places a KSAR directed to KS#2 action part in KSAR area#2.

6.5.4 General

Although each activation method was described in a separate subsection, more than one activation method could coexist in a system using the Pegasus blackboard framework. Figure 6.9 shows a simplified diagram of a possible configuration for the Pegasus system. Two activation methods are used in this configuration, namely: the event-based activation method and the Master-Slave activation method. The event-based activation method is used to activate KS#1, KS#2, and the Master KS, whereas the Master KS controls the

activation of KS#3 and KS#4 by using the Master-Slave activation method. Note that the actual names of the KSs are not given in the figure since the Master KS has not been developed yet and the final configuration of the system has not been decided yet.

The capabilities of the Pegasus Blackboard goes beyond the needs of the Pegasus system. Its speed allows it to accommodate and support a large number of KSs (i.e. around 100–150). Moreover, the framework's flexibility allows system developers to designate several KSs as master KSs. Those master KSs could be organized in a hierarchical fashion similar to the organization of the control modules in the HASP system, in a flat fashion—i.e. each master KS independently and directly controls a group of slave KSs, or a combination of both (refer to figure 6.10).

Some experiments were conducted to evaluate the system's performance when a large number of KSs are incorporated. It was found that using a master KS results in enhancing the system's performance considerably especially if changes to the blackboard result in activating only a small percentage of the KSs. For example, figure 6.11 shows an experiment with a system consisting of 50 KSs and in which blackboard changes resulted in activating only 20% of the KSs. The system's goal was to transfer 1250 requests from one KS to 20% randomly selected KSs. In figure 6.11.a the system was organized so that all the KSs were self-activating. As a result, when KS #1 placed a request on the blackboard, this resulted in activating all the KSs. The system accomplished its goal in 95 seconds¹ on a SPARC 2 workstation with 32 MBytes of memory. In figure 6.11.b a master KS was introduced to control the activation of the KSs. As a result, when KS #1

¹ The time reported here is actually the time required to execute KS #1 and was measured using the *time* command on UNIX.

- a -

- b -

➡ Notify (Event-based activation method)
 ➡ Activate (Master-Slave activation method)

Figure 6.10 Master KSs organized in: a- a flat fashion.

b- a hierarchical fashion.

- a -

- b -

Figure 6.11 The effect of using a master KS: a- all KSs are self-activating.

b- KSs are activated under the control of a master KS.

placed a request on the blackboard, this resulted in activating only the master KS which decided to activate only 20% of the KSs. This system organization required only 17 seconds to accomplish its goal on the same platform.

Chapter 7

Research Results

This chapter talks about a variety of subjects. It starts with describing earlier prototypes of the Pegasus Blackboard, pinpointing their weaknesses, and showing the remedial actions that were taken in the final prototype to eliminate them.

The second section emphasizes the practical benefits which resulted from the extreme flexibility embedded in the final prototype, namely: the use of the Pegasus Blackboard, in conjunction with the Dragon simulator, as a Real-time System Development Environment.

Finally, the last section illustrates the excellent performance of the Pegasus Blackboard by comparing its performance to that of the message passing through a mediator architecture¹. Moreover, the last section demonstrates the reliability of the Pegasus Blackboard by reporting its prolonged employment for simulation and actual network fault diagnosis.

¹ A detailed discussion about how this architecture could be implemented and the role of the mediator process is given in subsection 4.2.2

7.1 System development

The Pegasus Blackboard underwent several changes before taking its final shape. Those changes were required to improve the system's performance or increase its flexibility to meet the requirements stated in section 4.1. Two earlier prototypes were implemented before the final prototype was coded.

The first prototype was built mainly to get familiar with the UNIX environment, e.g. the shared memory segments and semaphore sets, and see how the different pieces of the blackboard would fit together. The blackboard database of this prototype contained two areas only, i.e. events and problems, and the control structures that managed them could only be accessed by a single process at a time since only one semaphore was used to ensure their integrity. This prototype supported one Knowledge Source activation method, namely the event-based activation method (described earlier in section 6.5). Furthermore, all KSs waited for changes to the blackboard on a single semaphore and all of them were unblocked when a change was made.

The second prototype was developed and implemented shortly after the first one was completed. In the second prototype each KS is assigned a private semaphore on which it can wait for changes to the blackboard. Also, the blackboard was further divided into four areas. This arrangement allowed us to selectively unblock some KSs based on their interests and the type of change. Extensive simulation experiments involving the Pegasus blackboard, the Dragon simulator, BANES, END, and LEAPR revealed a major weakness in the second prototype — the order in which the records of an area were processed was solely based on their location in the area. This problem was resolved in

	<i>First Prototype</i>	<i>Second Prototype</i>	<i>Final Prototype</i>
Support selective unblocking	No	Yes	Yes
Number of areas	2	4	7
Activation method(s)	Only event-based	Only event-based	Event-based + others
Size of lock	The whole blackboard	The whole blackboard	The effected area

Table 7.9 Main differences between the first, second, and final prototypes.

the final prototype by introducing the KSAR area (described earlier in subsection 6.1.6) and supporting activation methods other than the event-based activation method (refer to section 6.5). Also, the number of semaphores to ensure the data consistency of the blackboard control structures were increased to seven (i.e. one semaphore for each area) and thus allowed us to lock the affected area instead of the whole blackboard. Table 7.9 summarizes the differences between the three prototypes.

7.2 Practical Use

Apart from fulfilling the system requirements, the Pegasus blackboard was found very useful as a real-time system development environment. The Pegasus blackboard in conjunction with the Dragon simulator were used to develop and debug all the KSs in the Pegasus system.

By forcing all the communications between the modules of the system to go through the blackboard, we were able to decouple them completely. Also, the blackboard interface to the Dragon simulator allowed the KSs to control the course of simulation, query the

Figure 7.1 A typical configuration to develop KSs

simulated network, and conduct tests on it. As a result, each KS programmer could easily customize his testing environment.

Furthermore, to assist the KSs programmers in their task, they were provided with two useful programs. The first program, called P1, gave them access to the blackboard and allowed them to post any sequence of records to the various areas on the blackboard to check their KSs' behavior. It also allowed them to look at any record on the blackboard by specifying the record type and number. The second program, called Events, allowed them to monitor all the records posted to the blackboard. At run time, the events program could be set to monitor one of the blackboard areas. Therefore, programmers can monitor several areas on the blackboard simultaneously by running multiple instances of the Events program.

A typical configuration to develop a KS is shown in figure 7.1. It consists of the blackboard, the P1 program or the simulator, multiple instances of the Events program, and the KS itself. Later on in the development process, the configuration might be modified to include other KSs to recognize the best interaction between them.

- a -

- b -

Figure 7.2 First experiment: a Writer sending 10,000 Messages to a Reader

7.3 Pegasus Blackboard Performance

This section reports the results of two experiments that were conducted to evaluate the Pegasus blackboard Performance.

The first experiment is shown in figure 7.2. In this experiment, a writer tries to send 10,000 messages through the blackboard to a reader at the other end. This process required 11.72 seconds of CPU time on a SPARC 2 workstation. The same experiment was conducted after replacing the blackboard by a mediator process (refer to 7.2.b) and the time required was 14.91 seconds of CPU time. A simple calculation shows that using the blackboard is 27.2 % faster than using the mediator process.

The second experiment is shown in figure 7.3. In this experiment, a client tries to acquire the services of a server 10,000 times by repetitively sending a request and

- a -

- b -

Figure 7.3 Second experiment: a Client requesting the services of a Server 10,000 times

receiving a reply, once through the blackboard and the other time through a mediator process. When the blackboard was used, the process required 25.65 seconds to complete. When the mediator process was used, the process required 30.48 seconds to complete. This experiment showed that using the blackboard is 18.8 % faster than using a mediator process. It should be pointed out that all the experiments reported were conducted in a multitasking environment on a uni-processor machine and the results would certainly have been different if the experiments were conducted on a multi-processor machine. Using a large shared memory to provide communications between several programs on a multiprocessor system is not recommended because of the high price tag of multi-ported memory chips and the probable congestion problems that will arise.

The Pegasus blackboard proved to be very reliable through its usage for simulation and actual network fault diagnosis. The second prototype was used for simulation for about two months. During that period, approximately 40 million messages were sent through the blackboard with out any real problem. The final prototype conveyed about 100 million messages between the various modules of the Pegasus system, again without any problem. Finally, a copy of the Pegasus blackboard was installed in the BNR's Network Operations Centre in mid December 1991 as part of a diagnostic system. It has been running there since that time without any problem.

Chapter 8

Conclusions

8.1 Contributions

We have developed a blackboard architecture which were successfully used to build a multiple paradigm system to diagnose computer network faults. The design of the system was greatly influenced by the characteristics of the application problem in hand and the nature of the knowledge sources that used it.

Unlike other blackboard systems, the knowledge sources execution could not have been sequential. We arrived at this conclusion at the early stages of the system development because some modules ran continuously without terminating (e.g. Dragon simulator and BANES) and others (e.g. END) were not CPU-intensive modules (i.e. some diagnostic procedures in END conducted external tests on the network while others required human operators intervention). We implemented each knowledge source as a separate process and ran them in a multitasking environment. Implementing the system in this way allowed it to handle several faults concurrently and gave us maximum flexibility in selecting the internal representation of each KS.

The blackboard provided two ways of activating KSs: one way was provided for the self-activating KSs — KSs which could decide for themselves whether they can contribute to the solution state by looking at the blackboard changes. The second way was provided for the KSs which were activated under the Master KS's control.

A prototype of the Pegasus blackboard is currently used by the Pegasus project members to develop and test various diagnostic methods. Also, this prototype was reliable enough to be installed in the Network Operations Centre at BNR as part of a diagnostic system.

8.2 Summary of Results

Using the blackboard architecture proved to be very useful: the modularity of the KSs in the system allowed several developers to work on the project at the same time. Each KS was developed and tested separately from the others. Furthermore, we could defer the decision of how the KSs will cooperate to perform their task until we have a better understanding of the strengths and weaknesses of each one of them.

The design of the system (i.e. selective unblocking and using shared memory segments for interprocess communication) and the use of the C language to implement it are credited for the high speed of the system. This feature allowed the quick development and testing of the various diagnostic methods used in the system.

8.3 Future Work

The work on the Pegasus system and the Pegasus blackboard is far from completion.

In addition to the need for further testing of some KSs and the development of some others, the following work will improve the system capabilities:

1. Writing the Master KS and integrating it to the system: We deliberately postponed this step until we had a better understanding of the capabilities of each KS and the application domain characteristics. At the same time, we anticipated the special needs of this KS to control the others and designed the blackboard architecture accordingly. The major tasks of the Master KS would be resolving conflicts between contradictory hypotheses concluded by the different KSs and determining the appropriate time and KS to investigate a certain fault in the network.
2. Adding pattern-matching capabilities for record retrievals: In the current version of the Pegasus blackboard, no pattern-matching capabilities are provided for the KSs to assist them in retrieving records. A decision was made to postpone considering adding such capabilities until a better understanding of the KSs and their specific needs is formed.
3. Extending the blackboard so that data is stored in dynamic data structures: In other words, the number of fields in each record will be varying. This feature will allow us to use one version of the blackboard in different diagnostic systems. It will also facilitate adding new KSs which require new fields in the records to the system while it is running.
4. Extending the blackboard to keep a record of all the data posted to it: In the current version of the Pegasus blackboard, records in the various areas are continuously reused to post new data. As a result, any given record stays on the blackboard for an hour or two (i.e. depending on how busy the system is and the size of the areas).

The proposed extension will save the records by writing them to disk. Those records could be used to develop and test a diagnostic agent using the case-based reasoning approach.

References

- [1] BNR Information Technology, "BNR Local Area Network Environment: An Introduction for the Network User", Ottawa, Canada, July 1990.
- [2] W. Stallings, Handbook of Computer-Communications Standards Second Edition, Vol. 3, New York: Macmillan, 1989.
- [3] Bell-Northern Research, "MAGNet: Network management tools", Release 3.0, Ottawa, Canada, June 1990.
- [4] L. Erman, F. Hayes-Roth, V. Lesser, D. Reddy, "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty", Computing Surveys, vol. 12, pp. 213-253, 1980.
- [5] H. Nii, E. Feigenbaum, J. Anton, A. Rockmore, "Signal-to-Symbol Transformation: HASP/SIAP Case Study", AI Magazine, vol. 3, pp. 23-35, 1982.
- [6] M. Nagao, T. Matsuyama, H. Mori, "Structural Analysis of Complex Aerial Photographs", Proc. of the 6th IJCAI, vol. 2, pp. 610-616, 1979.
- [7] D. Corkill, "Blackboard Systems", AI Expert Magazine, pp. 41-47, Sep. 1991.
- [8] H. Nii, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", AI Magazine, pp. 38-53, Summer 1986.
- [9] L. Erman, V. Lesser, "A Multi-Level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge", Proc. of the 4th IJCAI, pp. 483-490, 1975.

- [10] F. Hayes-Roth, V. Lesser, "Focus of Attention in the Hearsay-II Speech Understanding System". Proc. of the 5th IJCAI, vol. 1, pp. 27-35, 1977.
- [11] V. Lesser, L. Erman, "A Retrospective View of the Hearsay-II Architecture", Proc. of the 5th IJCAI, pp. 790-800, 1977.
- [12] M. Frontini, J. Griffin, and S. Towers, "A Knowledge-Based System for Fault Localization in Wide Area Networks", Proceedings of the Second International Symposium on Integrated Network Management, Washington D.C., pp. 519-530, Apr. 1991.
- [13] R. Englemore, T. Morgan, (eds), Blackboard Systems, Massachusetts: Addison-Wesley, 1988.
- [14] N. Dawes, "Dragon: A very high speed, very large scale network simulator", Carleton University Technical Report SCE-91-31, August 1991.
- [15] Sun Microsystems Inc., "System Services Overview", 800-3846-10, U.S.A., March 1990.
- [16] Sun Microsystems Inc., "Programming Utilities and Libraries", 800-3847-10, U.S.A., March 1990.
- [17] B. Pagurek, N. Dawes, A. Kaye, "A Multiple Paradigm Diagnostic System For Wide Area Communication Networks", Carleton University Technical Report SCE-91-37, 1991.
- [18] N. Dawes, J. Altoft, B. Pagurek, "Effective Network Diagnosis by Reasoning in Uncertain Nested Evidence Space", Carleton University Technical Report SCE-91-26; submitted to IEEE conference on Artificial Intelligence Applications (Jul 1991).

- [19] Robert Dussault, "A Diagnostic Expert System for Wide Area Networks", Master of Engineering Thesis, Carleton University, March 1992.
- [20] Francois Viens, private conversation.
- [21] J. Altoft, N. Dawes, "RUNES: Dempster-Shafer Reasoning in Uncertain Nested Evidence Spaces", Carleton University Technical Report SCE-91-09; submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence (1991).

Appendix A

Glossary

BANES	a KS which generates hypotheses about the network elements' status by analyzing the status messages and relating them to the network configuration.
Blackboard model	a relatively complex problem-solving model which divides the domain knowledge needed to solve a problem among independent modules called knowledge sources and forces the communication between them to go through a global database called the blackboard database.
Diagnostic agent	a module in the Pegasus system implementing one of the diagnostic methods. Each diagnostic agent is coded as a separate process.
Dragon	a piece of software to simulate the sample network. The simulator is used to test the various diagnostic methods in the Pegasus system.
END	a rule-based KS used to verify BANES hypotheses and to investigate the problems reported to the operators.
ICMP	(Internet Control Message Protocol) a protocol used by hosts and gateways to exchange messages indicating errors and congestion problems.

KS	(Knowledge Source) The domain knowledge needed to solve the problem is divided into independent modules called knowledge sources.
KSAR	(Knowledge Source Activation Record) a data structure produced by the precondition part of a KS describing how its corresponding action part could contribute to solving the problem. Those data structures are usually kept in a queue which is presented to a scheduler to decide which action to execute next.
KSid or KS_id	Each KS in the Pegasus blackboard is given a unique number at run time, KS_id, which is used by the other KSs in the system to communicate with it.
LEAPR	a KS based on neural networks technology is used to monitor the network to detect performance degradation or any abnormal traffic.
Master KS	a special KS in the Pegasus system which is acquainted with the network status and activities going on in the system. The main task of the master KS is to control the activation of the other KSs in the system to enhance the system performance.
Mutex semaphore	Each area on the Pegasus blackboard is associated with a semaphore, Mutex Semaphore, to ensure the consistency of the area's control data structures.
Pegasus Blackboard	a framework based on the blackboard problem-solving model which were designed and developed to facilitate communication and interaction between several diagnostic agents.
Pegasus System	a multiple paradigm system under development at Carleton University to diagnose network faults.
PING	(Packet Internet Groper) a powerful program build on top of the ICMP protocol to analyze network problems.
Problem-solving model	"a scheme for organizing reasoning steps and domain knowledge to construct a solution to a problem" [8].

Self-activating KSs	KSs which could decide for themselves whether they can contribute to the solution state by looking at the blackboard changes.
SNMP	(Simple Network Management Protocol) a widely used management protocol for the Internet.

END

24 05 93

FIN