DESIGN AND ANALYSIS TECHNIQUES FOR CONCURRENT BLACKBOARD SYSTEMS

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William and Mary in Virginia

In Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

by

John William McManus

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

John William McManus, Author

Approved, April 1992.

William L. Bynum, Dissertation Advisor

Stefan Feyock

Rex K. Kincaid

David M. Nicol

P. Douglas Arbuckle, NASA Langley Research Center

Dedication

This thesis is dedicated to my parents, Vincent Joseph and Maryelaine McManus; my grandparents Joseph and Margaret McManus, and John William and Mary Ramey; and Dr. William Shelton Gray, Jr. I thank each of your support and for the love and encouragement that you have given me.

O Attic Shape! Fair Attitude! with brede
  Of marble men and maidens overwrought.
With forest branches and the trodden weed;
  Thou, silent form, dost tease us out of thought
As dost eternity: Cold Pastoral!
  When old age shall this generation waste,
    Thou shall remain, in midst of other woe
Than ours, a friend to man, to whom thou say'st,
  'Beauty is truth, truth beauty'–that is all
    Ye know on earth, and all ye need to
      know.

Ode on a Grecian Urn
John Keats
1819

Table of Contents

Introduction 2
- 1.1 Statement of the Problem 2
- 1.2 Unified Design and Analysis Methodology for Concurrent Blackboard Systems 3
- 1.3 The Concurrent Object–Oriented Blackboard System 7
- 1.4 Paladin and Concurrent-Claws 8
- 1.5 Thesis Outline 9

The Blackboard Problem-solving model 13
- 2.1 The Blackboard Model 13
- 2.2 Description of the Blackboard Model 15
- 2.3 Concurrent Blackboard Systems 16
- 2.4 Types of Concurrent Blackboard Systems 19
- 2.5 Distributed Blackboard Systems 24
- 2.6 Conclusions 32

A Formal Model for Blackboard Systems 35
- 3.1 Formal Definition of a Blackboard System 35
- 3.2 Knowledge Source Activation Graph 41
- 3.3 Examples 42
- 3.4 Multiple Instances of a Knowledge Source 49

Blackboard System Design and Analysis Techniques 52
- 4.1 Unified Approach to System Design and Analysis 53
- 4.2 Knowledge Source Connectivity Analysis 54
- 4.3 Examples of Connectivity Analysis 59
- 4.4 Conclusions 70

A Simulation Model for Blackboard Systems 73
- 5.1 Blackboard System Simulation Model 73
- 5.2 Description of the Simulation Model 79
- 5.3 Conclusions 81

The Concurrent Object–Oriented Blackboard System. 83
- 6.1 Description of The COBS Architecture 83
- 6.2 Description of The COBS Design and Analysis Tools 88
- 6.3 Description of The COBS Simulation Model 91
- 6.4 Description of The COBS Code Generator 94

The Paladin Tactical Decision Generation System 97
- 7.1 The Tactical Guidance Research and Evaluation System 97
- 7.2 TiGRES Components 98
- 7.3 The Paladin Software 98
- 7.4 KBS Modules of Paladin 101
- 7.5 Paladin Testing Procedures 104
- 7.6 Concurrent Paladin 106
- 7.7 Evaluation of Cube_CLAWS Performance 109
- 7.8 Results 114

Table of Contents

7.9 Conclusions 114

Table of Contents

Validation of the COBS Simulation Model. 117
8.1 Simulation of Cube_CLAWS 117
8.2 Validating Cube_CLAWS Simulation Results 118
8.3 Simulation of Paladin 122
8.4 Validating the Paladin Simulation Results 123
8.5 Conclusions 125

Utilizing the Concurrent Object–Oriented Blackboard System 127
9.1 Analysis of the Cube_CLAWS Design Specifications 127
9.2 Analysis of the Cube Simulation Results 130
9.3 Analysis of the Paladin Design Specifications 134
9.4 Evaluation of Paladin Simulation Results 135
9.5 Evaluation of Alternative Hardware Configurations 136
9.6 Conclusions 142

Summary 144
10.1 The Formal Model for Blackboard Systems 144
10.2 Future Research 146

Glossary 148

Appendix A Example Blackboard System Specifications 152
A.1 Blackboard System Specifications 152
A.2 Blackboard System Execution Traces 155

Appendix B Example Connectivity Analysis Results 161
B.1 Examples of Connectivity Analysis 161
B.2 Example of a Simulation Run 176

Appendix C 189

Appendix D Concurrent Blackboard Simulation System Verification Results and COBS Simulation System Software 214
D.1 Simulation Software and Results for B1 214
D.2 Simulation Software and Results for B2 223
D.3 Simulation Software and Results for B3 235
D.4 Simulation Software and Results for B4 246
D.5 Simulation Support Software 256
D.6 COBS Simulation System Software 264

Appendix E The COBS Code Generator 279
E.1 Listing of the COBS Code Generator 279
E.2 COBS Generated B2 Code 293

Appendix F COBS Verification and Validation Results 300
F.1 Cube Design Analysis Results 300
F.2 Cube2 Design Analysis Results 300
F.3 Cube4 Design Analysis Results 301
F.4 Cube8 Design Analysis Results 303

Table of Contents

F.5 Cube Simulation Results 306
F.6 Cube2 Simulation Results 307
F.3 Cube4 Simulation Results 310
F.8 Cube8 Simulation Results 312
F.9 Paladin Design Analysis Results 318
F.10 Paladin Simulation Results 318

Index 321

Bibliography 322

Acknowledgments

The author would like to thank Dr. William Lee Bynum for all of the time, support, and ideas that he has provided. Thanks are given to the additional committee members for their willingness to serve on my committee: P. Douglas Arbuckle, Stefan Feyock, Rex K. Kincaid, and David M. Nicol. Special thanks go to David Nicol for reviewing the preliminary drafts of the Formal Model and the Simulation Model, his comments and ideas were invaluable to my research. Special thanks also go to P. Douglas Arbuckle for providing a safe haven in which to perform my research.

Gratitude is owed to the NASA Langley Research Center, and the Flight Systems Directorate for allowing me the opportunity to continue my education while working in the Aircraft Guidance and Control Branch. Appreciation is given to the Aircraft Guidance and Control Branch, which provided access to the computer facilities used for this research.

A special debt is owed to my wife, Mary Loving McManus, for being understanding and supportive while I have earned this degree.

List of Tables

Table 4.1 B1 Connectivity Analysis Results....60

Table 4.2 B1 Output Overlaps Γ....60

Table 4.3 B1 Specialization Values Ω....60

Table 4.4 B1 Output to Input Connectivity Λ....61

Table 4.5 B1 Interdependence Values Π....61

Table 4.6 B1 Serialization Values Σ....61

Table 4.7 B2 Connectivity Analysis Results....62

Table 4.8 B2 Output Overlaps Γ....63

Table 4.9 B2 Specialization Values Ω....63

Table 4.10 B2 Output to Input Connectivity Λ....63

Table 4.11 B2 Interdependence Values Π....64

Table 4.12 B2 Serialization Values Σ....64

Table 4.13 B3 Connectivity Analysis Results....65

Table 4.14 B3 Output Overlaps Γ....66

Table 4.15 B3 Specialization Values Ω....66

Table 4.16 B3 Output to Input Connectivity Λ....66

Table 4.17 B3 Interdependence Values Π....67

Table 4.18 B3 Serialization Values Σ....67

Table 4.19 B4 Connectivity Analysis Results....68

Table 4.20 B4 Output Overlap Γ....68

Table 4.21 B4 Specialization Values Ω....69

Table 4.22 B4 Output to Input Connectivity Λ....69

Table 4.23 B4 Interdependence Values Π....69

Table 4.24 B4 Serialization Values Σ....70

List of Tables

Table 7.1. Modes of Operation 102

Table 7.2. Software Test Configuration Processor Assignments 110

Table 8.1 Cube_CLAWS Knowledge Source Execution Times (msec.) 118

Table 8.2 Cube_CLAWS System Execution Times (msec.) 122

Table 8.3 Paladin Knowledge Source Execution Times (msec.) 124

Table 8.4 Paladin System Execution Times (msec.) 124

Table 9.1 Processor Evaluation (msec.) 137

Table 9.2 Dynamics Knowledge Source Execution Times (msec.) 139

List of Figures

Figure 1.1. Blackboard System Design Process....6

Figure 1.2. Concurrent Object-Oriented Blackboard System Schematic....8

Figure 2.1. Parallel Blackboard System Schematic....20

Figure 2.2. Distributed Blackboard System with Centralized Blackboard Data Structure....25

Figure 2.3. Distributed Blackboard System with Distributed Blackboard Data Structure....26

Figure 3.1 Unsafe Knowledge Source Activation Graph....45

Figure 3.2 Safe Knowledge Source Activation Graph....47

Figure 4.1. Large $P_{j,k}$ Value....57

Figure 4.2. Small $P_{j,k}$ Value....57

Figure 4.3 Connectivity Graph for B1....62

Figure 4.4 Connectivity Graph for B2....64

Figure 4.5 Connectivity Graph for B3....68

Figure 4.6 Connectivity Graph for B4....70

Figure 5.2 Simulation Execution Structure....80

Figure 7.1. Schematic Of Paladin....102

Figure 7.2. Angle Definitions....103

Figure 7.3. CLAWS Schematic....108

Figure 7.4. Cube_CLAWS Schematic....109

Figure 7.5. Simulation Speedup....112

Figure 7.6. Processor Efficiency for Simulation....112

Figure 7.7. Maneuver Evaluation Speedup....113

Figure 7.8. Processor Efficiency for Maneuver Evaluation Subroutine....113

Figure 8.1. Maneuver Evaluation Knowledge Source Speedup....120

List of Figures

Figure 8.2. Knowledge Source Processor Efficiency 121
Figure 9.1. Maneuver Evaluation Knowledge Source Speedup 131
Figure 9.2. Knowledge Source Processor Efficiency 132
Figure 9.3. Cube Design Speedup 133
Figure 9.4. Cube Knowledge Source Efficiency 133
Figure 9.5 Paladin Speedup 135
Figure 9.6 Processor Efficiency 136
Figure 9.7. System Speedup 138
Figure 9.8. Processor Efficiency 139
Figure 9.9 Paladin Speedup 140
Figure 9.10 Processor Efficiency 141
Figure A.1 Unsafe Knowledge Source Activation Graph 153
Figure A.2 Safe Knowledge Source Activation Graph 155
Figure B.1 Connectivity Graph for B1 165
Figure B.2 Connectivity Graph for B2 169
Figure B.3 Connectivity Graph for B3 173
Figure B.4 Connectivity Graph for B4 176

Abstract

Blackboard systems are a natural progression of knowledge-based systems into a more powerful problem solving technique. They provide a way for several highly specialized knowledge sources to cooperate to solve large, complex problems. Blackboard systems incorporate the concepts developed by rule-based and expert systems programmers and include the ability to add conventionally coded knowledge sources. The small and specialized knowledge sources are easier to develop and test and can be hosted on hardware specifically suited to the task that they are solving.

The lack of a coherent set of design tools and guidelines results in blackboard systems being developed in an ad hoc fashion. Designers have been forced to make design choices in a void. Often the designer is stuck with a poorly designed system simply because revisions are too difficult once the system has been implemented. The lack of design and analysis tools is one of the reasons why incorporating concurrency into the blackboard problem-solving model has not been successful. The use of a centralized control mechanism, and contention in accessing the blackboard have also restricted the success of previous systems.

The Formal Model for Blackboard Systems was developed to provide a consistent method for describing a blackboard system. The Formal Model outlines the basic components of a blackboard system, and how the components interact. A set of blackboard system design tools has been developed and validated for implementing systems that are expressed using the Formal Model. The tools are used to test and refine a proposed blackboard system design before the design is implemented. The set of blackboard system design tools consists of a Knowledge Source Organizer, a Knowledge Source Input/Output Connectivity Analyzer, and a validated Blackboard System Simulation Model. My research has shown that the level of independence and specialization of the knowledge sources directly affects the performance of blackboard systems. Using the design, simulation, and analysis tools, I developed a concurrent object-oriented blackboard system that is faster, more efficient, and more powerful than existing systems. The use of the design and analysis tools provided the highly specialized and highly independent knowledge sources required for my concurrent blackboard system to achieve its design goals.

Design and Analysis Techniques for Concurrent Blackboard Systems

Chapter 1

Introduction

Blackboard systems are a natural progression of Knowledge-Based systems into a more powerful problem-solving technique. They provide a way for several highly specialized knowledge sources to cooperate to solve large and complex problems. Blackboard systems incorporate the concepts developed by rule-based and expert systems programmers and include the ability to add conventionally coded software to cooperate in solving problems. The smaller, specialized knowledge sources are easy to develop and test, and can be hosted on hardware specifically suited to their.

1.1 Statement of the Problem

Designing and developing blackboard systems is a difficult process. The designer is trying to balance several conflicting goals and achieve a high degree of concurrent knowledge source execution while maintaining both knowledge and semantic consistency on the blackboard. Blackboard systems have not attained their apparent potential because no established tools or methods exist to guide in their construction or analyze their performance.

> "At present (1989), intuition is the primary criterion for making an appropriate set of design choices. Application characteristics, hardware and communication characteristics, and how the application is implemented are all likely to strongly affect the performance of a parallel or distributed blackboard application.
>
> What is needed are system engineering guidelines for making appropriate design choices and improved languages and tools for constructing parallel and distributed blackboard applications. Such engineering guidelines require empirical measurements of pioneering applications. A few data points are beginning to emerge, but direct measurement of applications designed and executed on real parallel or distributed hardware remains to be performed."[1]

My assessment of the current state–of–the–art in blackboard systems (Chapter 2) is that the techniques used to design, analyze, and develop blackboard systems are not sufficient. Successful blackboard systems application requires the development of a unified design and analysis methodology. This assessment is supported by the discussions at the AAAI Workshops on Blackboard Systems and other related research. In "A Survey of the Eighth National Conference on Artificial Intelligence:

Pulling Together or Pulling Apart?" Paul Cohen presents some interesting hypotheses on the state of AI research. His research shows that common methodological problems—from poor system evaluation to absurd assumption—arise because the methodologies applied by most current AI researchers are not sufficient.[2]

> "I offer evidence for four hypotheses: First, AI research is dominated by two methodologies. Second, with respect to the goal of developing science and technology to support the design and analysis of AI systems, neither methodology is sufficient alone. Third, the bulk of AI research consequently suffers from familiar methodological problems, such as a lack of evaluation, a lack of hypothesis and predictions, irrelevant models, and weak analytic tools. Fourth, a methodology exists that merges the current "big two" and eliminates the conditions that give rise to methodological problems."[3]

Cohen's results show that most AI research applies one of two existing methodologies: *model–centered research*, or *system–centered research*. Model–centered research involves defining, extending, differentiating, and generalizing models. System–centered research involves designing and implementing systems to perform tasks that are too large and complex to be accomplished by a single algorithm.[4] The results are clear: No single existing blackboard system design and analysis methodology is sufficient.

1.2 Unified Design and Analysis Methodology for Concurrent Blackboard Systems

To address these problems, I have developed a methodology and a set of unified techniques for the design, simulation, analysis, and implementation of concurrent blackboard systems. The techniques include: a formal model for blackboard systems, a set of design and analysis techniques, a simulation model, and an automatic code generator for blackboard systems. This methodology provides for automatic generation of the blackboard system software based on the formal blackboard system design specifications.

The lack of a coherent set of design techniques and methodologies for blackboard systems has resulted in blackboard systems being developed in an ad hoc fashion. Some shells for building blackboard systems exist, such as Cage, Poligon, and GBB. These shells have no design aids or analysis tools and are insufficient. The shells do not provide the system designer with the tools or information required to successfully implement a system. As a result, many system designers have been stuck with poorly

designed systems because revisions are too difficult once the system has been implemented. The lack of design and analysis techniques is a major reason why incorporating concurrency into the blackboard problem-solving model has not been successful.

1.2.1 A Formal Model for Blackboard Systems

The need for a formal model for blackboard systems was highlighted at The First Workshop on Blackboard Systems[5] and has been repeated at the subsequent workshops[6].

> "As more and more systems appear claiming to use the blackboard paradigm, it is increasingly necessary to determine a set of criteria that define the essential elements of blackboard systems."[7]

As interest in blackboard systems has grown, a formal description of the blackboard problem solving paradigm and the components of a blackboard system have become crucial. A formal model is required to define the basic components of a blackboard system. The formal model should specify what is, and what is not, a blackboard system. The Blackboard Architecture and Organization Panel at the workshop was concerned with identifying the basic attributes of a blackboard system, and with how far a system can deviate from the defined architecture and still be classified as a blackboard system.

I have developed a formal model for blackboard systems to address these issues. The Formal Model for Blackboard Systems is presented in detail in Chapter Three. The formal model outlines the basic components of a blackboard system, and how the components interact. Systems that cannot be expressed using the formal model are not valid blackboard systems. The formal model is consistent with the description of a blackboard system presented by H. Penny Nii[8], and the formal description presented at the blackboard systems workshops.

1.2.2 Blackboard System Design and Analysis Techniques

The set of blackboard system design techniques consists of a Knowledge Source Organizer, and a Knowledge Source Connectivity Analyzer. These generic techniques are used to test and refine a proposed blackboard system design before implementing the system. My preliminary research [9,10,11,12] has shown that the level of

independence and specialization of the knowledge sources directly affects the performance of blackboard systems.

This dissertation describes the first use of specialized blackboard system design and analysis techniques to design and predict the performance of concurrent blackboard systems. These techniques can be applied to many distributed system design problems and other parallelization problems. The design techniques measure knowledge source output/input connectivity across a shared blackboard data structure. This information is used to determine knowledge source independence and point out potential hotspots on the blackboard data structure.

The first technique, Knowledge Source Organization, decomposes the problem into component parts and aids in the initial selection of knowledge sources and the formation of the blackboard structure. Knowledge Source Organization develops the basic specification of the blackboard data objects and the knowledge sources.

The second technique, Knowledge Source Connectivity Analysis, completes the formal specification of a blackboard system, and computes a knowledge source connectivity graph showing blackboard data object connectivity between knowledge sources. The blackboard data object connectivity information is used to determine the interdependencies between knowledge sources and locate hotspots on the blackboard. The information is also used to predict the overall system performance and blackboard overhead for individual blackboard components. Knowledge sources that are tightly coupled may be combined into a single knowledge source to decrease the traffic on the blackboard and to increase system performance. Knowledge sources that have a high overhead and small execution times may be combined to reduce the overhead costs. This technique can be applied to many generic parallelization problems to analyze data connectivity in the system.

1.2.3 Blackboard System Simulation Model

The Blackboard System Simulation Model is a serial discrete event simulation. Verification of the simulation model and the implementation of the simulation system was performed using example blackboard system specifications. The performance of the simulation system was validated by comparing the performance predicted by the simulation model to the performance of known blackboard systems. Any discrepancies between the predicted and the actual performance of the system were

used to refine the blackboard simulation model, or possibly the blackboard system implementation. This refinement process was iterated at each stage of development until the predicted and actual performance values converged. The verification and validation of the simulation model is discussed in detail in Chapter Eight.

This dissertation describes the first use of simulation to predict the performance of blackboard systems. The knowledge source distribution information and the blackboard data connectivity generated by the knowledge source connectivity analysis is used to build the simulation model of the system. The execution of the knowledge sources and their interaction with the blackboard data objects are modeled.

Once validated, a blackboard simulation model allows tradeoff and performance analysis to be completed before systems are implemented. The simulation model may also be used to determine the effect of different hardware and software configurations on system performance. The data generated by the model can be used to predict the performance of a single blackboard system implemented using several hardware/software configurations. The designer can develop an initial design using the blackboard system design tools and then use the blackboard system simulation model and blackboard system analysis tools to refine the design. (Figure 1.1) When the design goals have been met, and the systems performance validated by the blackboard system simulation model, the design can be implemented. The use of a design–>simulate–>refine process makes it easier for designers to meet design and performance goals, resulting in more efficient concurrent blackboard systems.

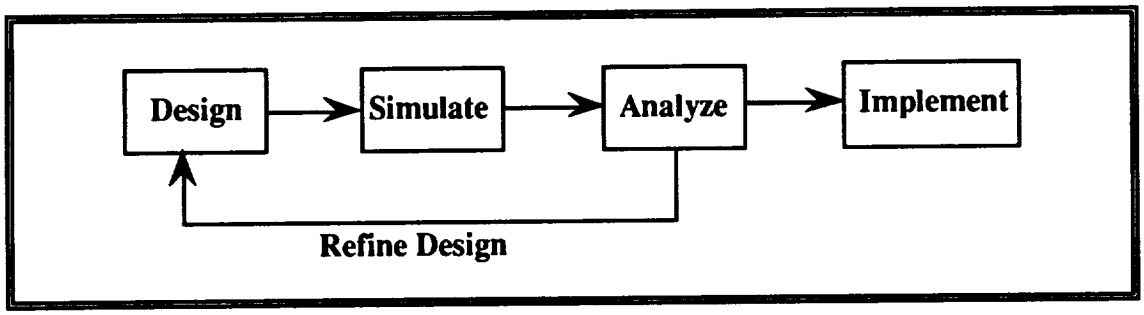

Figure 1.1. Blackboard System Design Process

1.2.3 Blackboard System Code Generator

The Blackboard System Code Generator generates all of the software required to implement a blackboard system based on a valid system specification. The code generator does not generate the software for the knowledge sources. The system

generates the Blackboard Data Structure and all required support software. This includes all required blackboard data object locks and synchronization primitives. The system generated software is implemented using the Concurrent Object–Oriented Blackboard System.

1.3 The Concurrent Object–Oriented Blackboard System

The Concurrent Object–Oriented Blackboard System (COBS) is an implementation of the formal model for blackboard systems, the design and analysis techniques, and the simulation model. The basic premise of COBS is that a concurrent blackboard system consisting of a set of highly independent, highly specialized knowledge sources cooperating using a shared object-oriented blackboard with no centralized blackboard controller will achieve the potential of the blackboard model of problem solving and outperform a conventionally designed blackboard system.

To date, most concurrent blackboard systems have centered on solving the memory contention problem or on reducing centralized control. Three generic concepts combine to form a synergistic solution to both of these problems.

1) A Daemon Driven Control Structure

2) The use of Blackboard Handlers

3) Highly Specialized, Highly Independent Knowledge Sources

COBS uses a associative memory-based control structure and blackboard handlers to implement a distributed control structure. The associative memory is implemented using an object-oriented data structure and daemon functions. This type of control structure models the associative memory concept of how the human brain functions and allows concurrent processing. The distributed control structure and the use of highly specialized, highly independent knowledge sources reduce memory contention on the blackboard and remove centralized control.

Blackboard control and knowledge source selection is achieved using daemons attached to the blackboard data objects. The daemons activate blackboard handlers when the data elements on the blackboard are updated. This daemon-driven control structure removes the need for a centralized blackboard control mechanism and allows concurrent knowledge source execution. The blackboard handlers control knowledge

source activation and provide the knowledge sources read/write access to the blackboard.

COBS was used to develop several applications. The applications, presented in detail in Chapter Seven, were used to validate the blackboard system simulation model. A preliminary blackboard simulation model was used in conjunction with the blackboard system design tools to fine tune the performance of the blackboard system before the design was implemented.

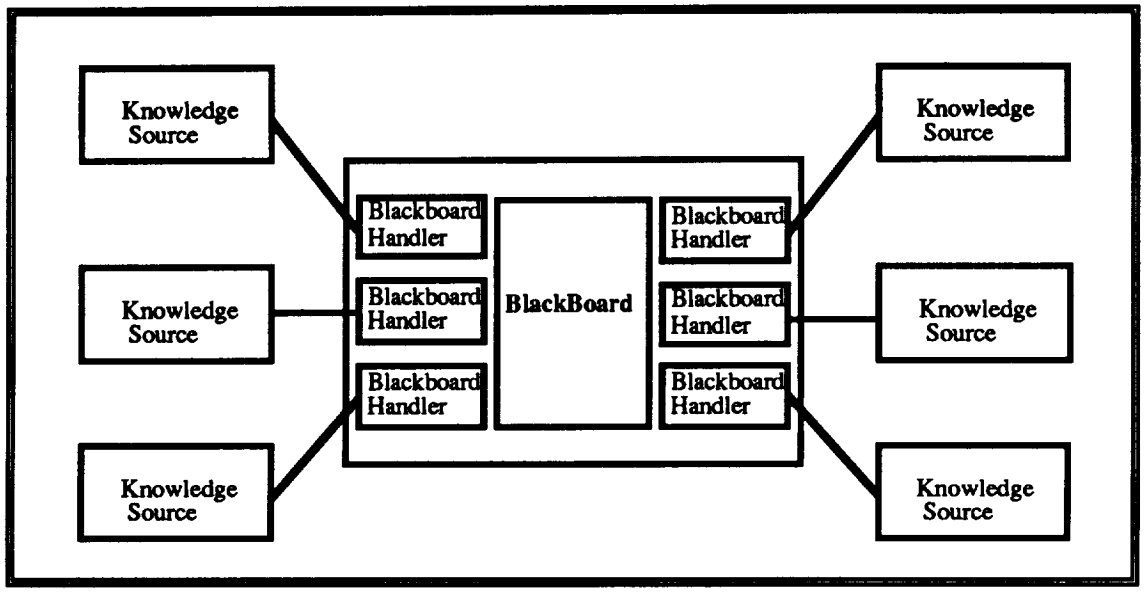

Figure 1.2. Concurrent Object-Oriented Blackboard System Schematic

1.4 Paladin and Concurrent-Claws

A Tactical Decision Generator for Air-to-Air combat was developed to evaluate COBS. Past research [13,14,15,16] has shown that this application requires a set of highly specialized knowledge sources cooperating in a dynamic environment. This application should effectively test all features of COBS. A concurrent blackboard system, Paladin, is used as an "Iron Pilot" for evaluating super-agile aircraft performance at NASA Langley Research Center. An "Iron Pilot" is a deterministic autopilot that acts as a challenging opponent in a simulated air combat engagement.

Paladin consists of multiple cooperating knowledge–based systems developed to study within–visual–range air combat engagements. This system incorporates modern airplane simulation techniques, sensors, and weapon systems. A concurrent message-passing version of the application, Concurrent-CLAWS, was developed and tested on

a 16 processor Intel Hyper Cube[17]. The concurrent version highlighted the gains that could be made by using a concurrent blackboard system and outlined many of the problems associated with designing concurrent blackboard systems. The experience gained in designing and developing these preliminary systems has affected the basic structure of COBS and highlighted the need for design and analysis tools for blackboard systems.

1.5 Thesis Outline

This thesis is organized in the following manner:

Chapter 2: The Blackboard Problem-solving model. This chapter restates the standard definitions of the blackboard problem-solving model and presents the development of serial implementations of the blackboard model. Concurrent blackboard systems are presented. Several problems associated with current implementations of parallel and distributed blackboard systems are discussed.

Chapter 3: A Formal Model For Blackboard Systems. This chapter presents a formal specification of the blackboard problem-solving model and the components of a blackboard system.

Chapter 4: Blackboard System Design and Analysis Techniques. This chapter describes a set of design and analysis techniques for blackboard systems. Knowledge Source Connectivity is defined, and its effect on the performance of a blackboard system design is presented. Four examples are given, and the results of the design analyses are presented.

Chapter 5: A Simulation Model for Blackboard Systems. This chapter discusses a serial simulation model for blackboard systems that is used to model the performance of proposed blackboard system designs. A set of blackboard system performance metrics is developed to measure blackboard system performance.

Chapter 6: The Concurrent Object–Oriented Blackboard System. This chapter describes the Concurrent Object–Oriented Blackboard System model. and how it implements the formal model for blackboard systems. The four examples defined in Chapter Four are analyzed, and the results of the design analyses are used to verify the performance of the COBS design and analysis tools

Chapter 7: Paladin. The application that will be developed to evaluate the Concurrent Object–Oriented Blackboard System, the set of design and analysis tools, and the blackboard simulation model is presented in this chapter.

Chapter 8: Validation of the Concurrent Object–Oriented Blackboard System Simulation Model. This chapter describes the validation of the COBS simulation system using the two applications described in Chapter Seven and the blackboard system simulation model. The results of the analysis and simulation of the two examples and the systems actual performance data is used to validate the performance of the COBS simulation system.

Chapter 9: Utilizing the Concurrent Object–Oriented Blackboard System Simulation Model. This chapter describes the design and development of two applications using the set of design and analysis tools for concurrent blackboard systems and the blackboard system simulation model.

Chapter 10: Conclusions This chapter presents the summary of the Formal Design and Analysis Techniques, and outlines future work.

Endnotes for Chapter One

1 Daniel D. Corkill: "Advanced Architectures: Concurrency and Parallelism." In *Blackboard Architectures and Applications*, ed. V . Jagannathan et al., Academic Press, Inc 1989.

2 Paul R. Cohen: "A Survey of the Eighth National Conference on Artificial Intelligence: Pulling Together or Pulling Apart?" *AI Magazine*, Spring 1991. pp 16-41

3 Ibid.

4 Ibid.

5 Rajendra Dodhiawala et al: "The First Workshop on Blackboard Systems." *AI Magazine*, Spring 1989. pp 77-80

6 Keith Decker, "Blackboard Systems." *IEEE Expert*, October 1991. pp 71-72

7 Dodhiawala

8 H. Penny Nii, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures." *AI Magazine*, 7 (2), 1986, pp. 38 - 53.

9 Richard M. Hueschen and John W. McManus. " Application of AI Methods to Aircraft Guidance and Control." In *Proceedings 1988 American Control Conference*, June 15-17, 1988. pp. 195 - 201.

10 John W. McManus and Kenneth H. Goodrich. "Application of Artificial Intelligence (AI) Programming Techniques to Tactical Guidance For Fighter Aircraft." In *Proceedings AIAA Guidance, Navigation, and Control Conference*, 1989. AIAA Paper # 89-3525, pp. 851-858.

11 John W. McManus and Kenneth H. Goodrich. " Artificial Intelligence (AI) Based Tactical Guidance For Fighter Aircraft." In *Proceedings AIAA Guidance, Navigation, and Control Conference*, 1990. AIAA Paper # 90-3435.

12 John W. McManus "A Parallel Distributed System for Aircraft Tactical Decision Generation." In *Proceedings of the 9th Digital Avionics Systems Conference*, 1990, pp.505 - 512.

13 Richard M. Hueschen and John W. McManus, 1988

14 John W. McManus and Kenneth H. Goodrich, 1989

15 John W. McManus and Kenneth H. Goodrich, 1990

16 McManus, 1990

[17] Ibid.

Chapter 2

The Blackboard Problem-solving model

This chapter presents a historical overview of the development of the blackboard model, an overview of concurrent blackboard systems, and some of the issues involved in designing and implementing a parallel or distributed blackboard systems.

2.1 The Blackboard Model

The basic concepts of the blackboard problem-solving model were first presented in 1962 by Allen Newell. Newell describes the following model of problem solving:

> "Metaphorically we can think of a set of workers, all looking at the same blackboard: each is able to read everything that is on it, and to judge when he has something worthwhile to add to it. This conception is just that of Selfridge's Pandemonium[1]: a set of daemons, each independently looking at the total situation and shrieking in proportion to what they see that fits their natures..."[2]

The problem solving technique outlined by Newell, and in Simon's paper on information processing theory[3] are the basis of the blackboard problem-solving model.[4] The concepts presented by Newell led to the development of the production system problem-solving approach, and were later used in the development of the OPS 5 system.

A blackboard system consists of a set of knowledge sources, a blackboard data structure, and an opportunistic control strategy used to activate the knowledge sources. The control strategy is termed "opportunistic" due to the self-activating nature of the knowledge sources. Each knowledge source monitors the blackboard and activates itself based on the state of the blackboard. The blackboard model of problem solving is best described by H. Penny Nii.

> "A Blackboard System can be viewed as a collection of intelligent agents who are gathered around a blackboard, looking at pieces of information written on it, thinking about the current state of the solution, and writing their conclusions on the blackboard as they generate them."[5]

The blackboard is a centralized global data structure, often partitioned in a hierarchical manner, used to represent the problem domain. The blackboard is also used to allow inter-knowledge source communication and acts as a shared memory

visible to all of the knowledge sources. This design allows for opportunistic problem solving and allows a knowledge source to contribute towards the solution of the current problem without knowing which of the other knowledge sources will use the information. The use of opportunistic problem solving allows the data transfers on the blackboard to determine which processes are active at a given time. The use of an opportunistic data–driven control structure, and highly specialized knowledge sources allows a set of knowledge sources to cooperate to solve large, complex problems.

The blackboard problem-solving model is an evolution of the rule-based/expert system model of problem solving that seeks to build on the inherent strengths of those systems.

- Knowledge sources should be highly specialized and highly independent. This allows several rule-based/expert systems to cooperate to solve problems that are larger than can be managed by a single expert system. Specialization and independence provide modular protection, restricting the effect of a rule firing to the knowledge source it fires in.

- The control of rule application in a blackboard system is distributed across the set of knowledge sources. Conflict resolution is carried out at the blackboard level instead of being centralized in one rule base.

- A different inference engine and knowledge representation scheme can be selected for each knowledge source. This allows each knowledge source to use the knowledge representation scheme and problem-solving strategy best suited to solve its subset of the problem.

- Blackboard systems provide for parallel execution of the knowledge sources. This parallel execution of the separate knowledge sources cannot be modeled in a rule– based system using a single rule base without having a set of rules in the rule base whose only purpose is to order the firing of other rules. The addition of the "sequencing" rules violates the basic principle that the rules in a rule-based system are not serialized and can fire in any order.

2.2 Description of the Blackboard Model

Due to the hierarchical structure of the blackboard, each data object on the blackboard will usually have only one knowledge source that can update it. It is important to note that although knowledge sources are often referred to as "experts", knowledge sources are not restricted to Expert Systems or other AI systems. Many knowledge sources are numeric or algorithmic in nature. The use of multiple, independent knowledge sources allows each knowledge source to use the data representation scheme and problem-solving strategy that best suit the specific purpose of the knowledge source. The use of independent knowledge sources also provides for modular protection between knowledge sources. A change in the functionality of one knowledge source cannot effect the function of another knowledge source.

Penny Nii and others[6,7,8] have delineated three properties that the knowledge sources of a blackboard system should possess:

1) Knowledge sources can see everything on the blackboard at all times, and what they see represents the current state of the solution. This implies the ability to do concurrent atomic reads.

2) Knowledge sources can write their conclusions on the blackboard at any time without getting in the way of any other knowledge sources. This implies the ability to do concurrent writes to the blackboard.

3) The act of writing on the blackboard will not confuse any other knowledge sources. This implies that serializability is maintained and that write operations are atomic.

The implication of these properties is that a single problem is being solved asynchronously, in parallel, with no memory contention. This is not the case in serial implementations of blackboard systems. Most serial blackboard systems, including the early versions of Paladin[9], modify the blackboard system problem-solving model in such a way that it cannot be executed in parallel. Knowledge sources are treated as schedulable entities that are queued for execution, with only one knowledge source executing at a time. This modification to the blackboard model requires the additional overhead of adding a centralized control mechanism to handle the scheduling and execution of the knowledge source.

Serial blackboard systems implement the blackboard data structure as an interconnected data structure that is not globally visible to all of the knowledge sources at all times. These systems make the assumption that the knowledge source execution sequence selected by the control module preserves the consistency of the blackboard. Although serial blackboard systems remove the concurrency described in the model by allowing only one knowledge source to execute at a time, the use of the shared blackboard data structure still allows opportunistic problem solving.

These constraints and modifications to the blackboard model must be removed for the promise of the model to be fully realized. Current research in the area of blackboard systems centers on the development of concurrent blackboard systems that allow the knowledge sources to execute concurrently.

2.3 Concurrent Blackboard Systems

The development of concurrent blackboard systems requires a combination of expertise and tools from the AI field and from the Operating Systems/Concurrency Control fields. Past research on concurrent blackboard systems has lacked this global approach and has not realized the full potential of concurrent blackboard systems.

There are two major motivations for developing concurrent blackboard systems. The first, and most obvious motivation, is that properly implemented concurrent blackboard systems will outperform existing rule-based systems, expert systems, and serial blackboard systems. Concurrent blackboards also deliver more processing power than can be obtained in a uniprocessing environment. This increase in processing power allows the solution of larger and more complex problems. The second motivation is to allow the system designers to integrate a network of computers, each best suited to execute a specific type of knowledge source, into a single problem solving approach. An example of such a system is a heterogeneous network consisting of conventional architecture machines (VAX 3200™, Sun SPARC™) for numeric processing, specialized AI hardware for symbolic processing, and a shared memory parallel processing machine to host the blackboard. This type of network allows each knowledge source to be implemented in a computationally favorable environment and programming language.

2.3.1 Performance Issues

There are several factors that can affect the performance of concurrent blackboard systems. The four most common are listed below. This list will be used as a reference when discussing the performance of existing blackboard systems later in this chapter.

- Contention for shared data and shared resources.
- The cost of communications in a multiprocessor system. Several current blackboard systems, including Concurrent-CLAWS[10] , have demonstrated that message-passing costs, including message creation and receipt costs, and the finite bandwidth of the communication network, can become a significant portion of the system's execution time.
- Maintaining semantic consistency. Semantic consistency requires that the inputs to a knowledge source do not change during the time in which the knowledge source performs its problem-solving activities and places its results onto the blackboard.
- Process creation and management costs. These costs are significant only for systems that dynamically create and delete processes. Corkill[11] outlines the problems with this type of approach and with the use of time-sliced multiplexing of knowledge sources. The major problem with time-slicing is blackboard consistency. Changes can be made to the blackboard while a knowledge source is paused that affect the computation of the paused knowledge source. Corkill presents two "simple" approaches to overcoming this problem. Both are computationally expensive and can lead to systems where little progress is made due to the overhead of blackboard consistency recovery. Corkill states that "dealing with multiplexed knowledge source executions in a uniprocessor environment has not been worth the effort."[12]

2.3.2 Opportunities for Concurrent Execution

There are many levels in blackboard systems at which concurrency can be exploited. Concurrency is found at the application level, where the designer must

recognize and exploit the inherently concurrent features of the problem being solved. It is important to consider the grain size of the tasks being executed concurrently to insure that the performance gains outweigh the cost of implementing concurrent knowledge source execution.

Concurrency can also be exploited at a lower level if a language with built-in parallel-processing constructs, such as vector operations, can be applied to the problem. In the case of blackboard systems, it is also important to find a language that supports frame-based or object-oriented programming features to ease implementation of the blackboard data structure. One of the driving factors in designing a blackboard system is the type or types of hardware the system will be implemented on. This decision lays the architectural framework for the system and may constrain other features such as programming language selection.

Current research has outlined many options for exploiting parallelism in blackboard systems. The following list is presented as a introduction to some of these options. More detailed descriptions of how an option is implemented follow in the sections describing parallel and distributed implementations of blackboard systems.

1) Knowledge Parallelism: This exploits the ability to run multiple instances of a knowledge source in parallel.

2) Pipeline Parallelism: Data flow through levels in the blackboard is used to create execution pipelines.

3) Data Parallelism: Create partitions in the blackboard and distribute them so that different knowledge sources can concurrently access the blackboard data structure partitions and execute in parallel.

4) Control Concurrency: Allow the control element to execute concurrently with the knowledge sources.

5) Rule Execution: This allows several (or all) rules in an active knowledge source to be executed concurrently.

6) Blackboard Internals: Allowing concurrent object creation, deletion, retrieval, and modification.

2.3.3 Multiple Instances of a Knowledge Source

Several systems[13,14] have proposed the use of multiple instances of a knowledge source. The use of multiple instances of a knowledge source may lead to significant data coherence problems on the blackboard data structure. There are two commonly used ways to implement multiple instances of a knowledge source.

- Instances of knowledge sources can be partitioned into parallel execution pipelines.

Pipelining knowledge sources requires data parallelism on the blackboard. Pipelining is implemented by developing knowledge source pipelines and having each pipeline execute on a unique set of input data. The use of unique sets of inputs and outputs for each knowledge source pipeline protects the system from data coherence problems.

- Individual instances of the knowledge source can run in parallel.

Several systems claim to allow multiple instances of a single knowledge source to execute concurrently. These systems all use a serial control component to activate knowledge sources. The serial control structure allows a single knowledge source to execute at any given time, protecting the system from memory coherence problems. If the knowledge sources are not pipelined and they are allowed to execute concurrently severe data coherence problems can result. Without pipelines it is difficult to insure that a knowledge source does not interfere with another activation of that knowledge source, and that instances of a knowledge source do not attempt to update the same blackboard data object. The effect of allowing multiple instances of a knowledge source in the blackboard model of problem solving is discussed in detail in Chapter Three.

2.4 Types of Concurrent Blackboard Systems

Concurrent blackboard systems can be partitioned into two classes: parallel blackboard systems and distributed blackboard systems. Parallel blackboard systems are developed for multiprocessor shared memory architectures and distributed blackboard systems are developed for distributed memory, message-passing architectures.

2.4.1 Parallel Blackboard Systems

Parallel blackboard systems(Figure 2.1) stress the concurrent execution of knowledge sources and the blackboard control components in a shared address space. As outlined by Corkill[15], the major point of concern in parallel blackboard systems is the potential for asynchronous modifications to the blackboard by concurrently executing knowledge sources. These modifications may invalidate the results of other executing knowledge sources. Concurrency in parallel blackboard systems is also restricted by the use of a single, centralized control module. Nii states

> " A centralized controller drastically limits speedup, and knowledge source invocation should be distributed without synchronization." [16]

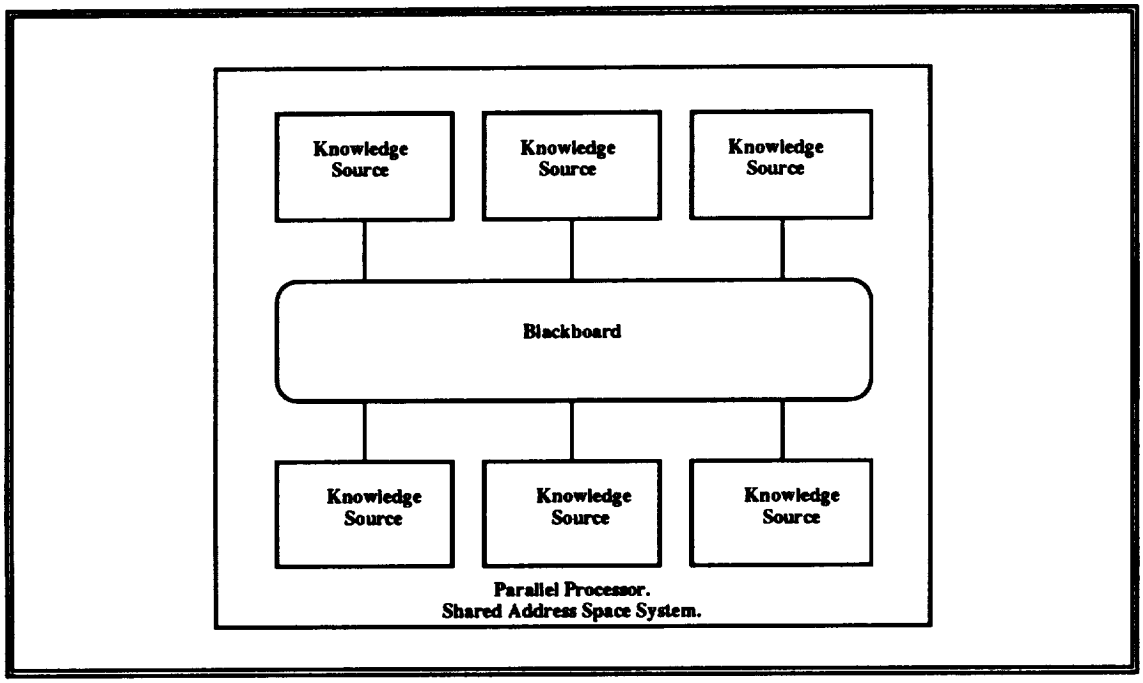

Figure 2.1. Parallel Blackboard System Schematic

2.4.1.1 The Cage Architecture

Rice, Aiello, and Nii[17] developed the Cage system. Cage extends the serial Attempt to GEneralize (AGE) system allowing for parallel execution of its applications components. The system allows for concurrent knowledge source execution, concurrent rule execution, and concurrent rule clause execution.

Two types of knowledge source concurrency exist in Cage. Multiple knowledge sources can execute on different levels of the blackboard data structure, or multiple

instances of a knowledge source can execute on different data objects on the same level of the blackboard. Multiple instances of a knowledge source executing concurrently are used to form execution pipelines that flow data up the blackboard data structure. Cage supports synchronized knowledge source execution. The synchronization is managed by the blackboard control structure. If synchronization is used, knowledge sources are held at a barrier until all previously selected knowledge sources complete execution. When no synchronization is used, knowledge sources execute when they are triggered.

The researchers investigated the use of concurrency at the rule execution level using a simulated parallel hardware environment. The condition clauses for all of the rules in the active knowledge sources were evaluated in parallel and the appropriate action clauses were executed in parallel. This sequence can be executed with or without synchronization between the evaluation of the condition clauses. When synchronization is used, it is implemented using barrier synchronization primitives.

Concurrent rule execution did not achieve the expected execution speedups. The cost of the barrier synchronization, the memory contention caused by the parallel execution of the condition clauses, the lack of a regular or "normal" rule form, and the lack of hardware support introduced overhead that outweighed the speedup gains possible with concurrent rule execution. This concept does not show promise for achieving speedup unless hardware support for concurrent rule execution is developed, and a normalized rule form is used to express knowledge in the knowledge sources.

Cage knowledge sources use atomic read and write operations to insure data coherence. The researchers discovered that a detailed specification of a knowledge source's activation conditions is required for concurrent blackboard systems. This is caused by the asynchronous operation of the knowledge sources. Cage uses complex preconditions to ensure that it is appropriate to activate a knowledge source and implements "conditional actions" to protect the system from race conditions. A conditional action tests the value of a data object before the data object is updated.

2.4.1.2 Shared Memory Blackboard Approach

Daniel Corkill[18] discusses several approaches to executing knowledge sources concurrently on a shared memory architecture and then proposes a Shared Memory Blackboard Approach. The first approach Corkill presents executes each knowledge source as an uninterruptable task. This approach can result in both memory contention

and resource allocation bottlenecks. The second approach Corkill discusses is time slicing the execution of the knowledge sources. Each knowledge source creates a "local" copy of all blackboard elements required for execution. This approach has several faults, including the introduction of context switching costs and other operating system overhead into the system execution time. The final approach discussed is to treat an entire knowledge source activation as an atomic operation. This approach requires the ability to treat blackboard objects, and even whole regions of the blackboard, as "allocatable resources" that can be locked for either exclusive access or for read-only access. This system places the responsibility for deadlock prevention on the programmer. Locking blackboard objects and regions is computationally expensive and can cause knowledge source blocking due to memory contention.

Corkill's shared memory blackboard approach allows each processor to directly access the shared blackboard data structure. Corkill proposes two ways to map the knowledge sources of a blackboard system to the available processors. He presents a functional mapping and a computational server mapping.

In the functional mapping approach each type of knowledge source is assigned to a specific processor. If multiple instances of a knowledge source are required the additional instances are queued for execution on the processor. This approach is suited for heterogeneous computer networks where different knowledge sources execute on specialized processors. The main problem with this approach is that processing demands may not be uniform. This will result in some processors being overloaded, with several instances of a knowledge source queued for execution, while other processors may be setting idle.

The computational server model is suited for a homogeneous computer network. Knowledge source activations are queued for execution. When a processor completes execution of a knowledge source it selects the highest rated knowledge source activation from the activation queue. This approach results in better load balancing across the network, but requires that all knowledge sources can be executed by all processors. Corkill outlines a way to extend the computational server model to heterogeneous networks. An activation queue is created for each type of processor in the network and knowledge sources are queued according to their processing requirements.

The implementation of Corkill's shared memory blackboard approach supports both object and region locking. Locks are ordered before they are acquired. If a lock cannot be acquired the process blocks waiting for that lock. Integrating blackboard monitoring and control requires determining which processes are responsible for which activities. The control shells define event handlers that are used by the processes to handle the initialization tasks associated with an event. The event handlers must be defined to interact correctly with the multiprocessing control shell. No performance data for the shared memory blackboard approach was presented.

2.4.1.3 The Agora System

Roberto Bisiani[19] developed the Agora system to support the ANGEL speech recognition system. ANGEL consists of four loosely connected blackboard subsystems communicating through a centralized blackboard data structure. A pipelining control structure is used to activate the blackboard subsystems. Bisiani stresses that each blackboard subsystem has different performance and hardware requirements. The use of a blackboard problem solving structure provides software engineering support by decoupling the system integration mechanisms from the behavior of the knowledge sources.

The Agora system supports the concept of Shared Data Types (SDT). The knowledge sources in Agora communicate via shared access to the blackboard data objects. The data objects are strongly typed and can only be accessed by used defined accessor functions. Data objects can be created dynamically, and modifications to the object are reported to all interested knowledge sources. Data object sharing is used to insure that data objects are consistent and accessible across processor boundaries.

SDT's are abstract data types consisting of two elements. Data definitions specify the structure of the data elements, and accessor functions specify the available operations for instances of the data type. An instance of an SDT is refered to as a Shared Data Structure (SDS). Agora provides a set of basic accessor functions for managing SDS's. Knowledge sources are activated when they receive an activation request posted from another knowledge source. When an SDS is modified the knowledge source updating the SDS sends an activation message to all knowledge sources that "have declared an interest" in the SDS. The activation operations are managed by the Agora system, not by the applications programmer.

Bisiani presents a unique memory management scheme to reduce hotspots in memory. Agora uses a write-once memory management system. The use of a write-once memory management system removes the need for blackboard data element read locking. This allows read operations to return stale data. The programmer is required to implement any protocols required to prevent stale data from being read by a knowledge source.

The write once memory management system implemented by Agora is extraordinarily memory expensive. Additional system overhead is required to support either "stop and wait" or "background" garbage collection. Memory coherence problems can occur if the user does not implement the memory coherence protocols correctly, or if no protocols are implemented. The cost of the background garbage collection and the memory coherence protocols may outweigh any performance gains achieved by Agora. Only minimal performance data was presented for the Agora system.

2.4.2 Conclusions

The systems discussed above approach the problems presented by shared memory parallel blackboard systems from many different perspectives. The Cage system and Corkill's Shared Memory Blackboard face bottlenecks at the shared blackboard data structure. The lack of two-phase locking protocols cause knowledge source serializability problems. The Agora systems uses its write-once memory management scheme to avoid memory contention problems, but allows access to stale data and adds the overhead of the garbage collection system. A successful blackboard system implementation must support knowledge source pipelining and two-phase locking protocols, and reduce memory contention on the blackboard data structure.

2.5 Distributed Blackboard Systems

Distributed blackboard systems are designed to use a distributed, message-passing architecture. This design allows blackboard data to be communicated among autonomous blackboard subsystems or knowledge sources. The major issue in designing a distributed blackboard system is deciding what information to communicate, where to store it, and when and where to send it.

The are two types of distributed blackboard systems. The first type of distributed blackboard system, shown in figure 2.2, uses a centralized blackboard data structure

that is accessed by a set of distributed knowledge sources. The second type, shown in figure 2.3, distributes the blackboard data structure, allowing each knowledge source direct access to some part of the blackboard data structure. Systems that distribute the blackboard data structure have additional data coherence problems not found in centralized blackboard based systems, and in many cases violate the fundamental requirement that all knowledge sources can see and access all of the blackboard data objects at all times. Several systems that use a distributed blackboard data structure will be discussed in detail later in this chapter.

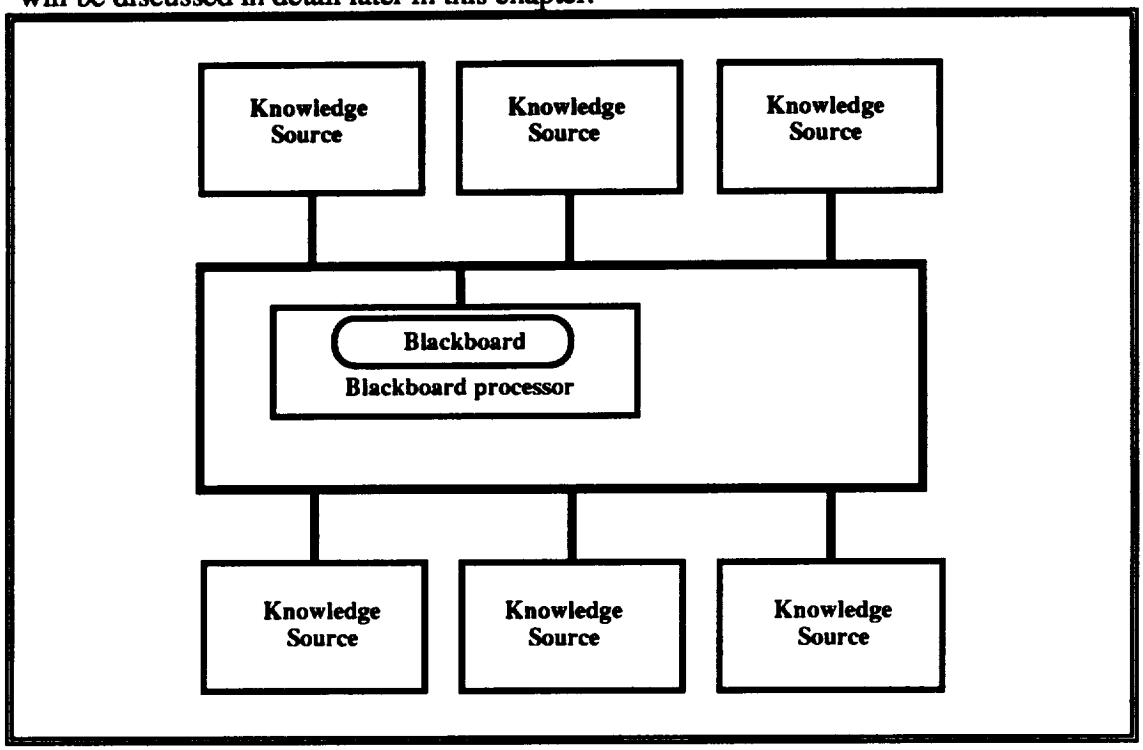

Figure 2.2. Distributed Blackboard System with Centralized Blackboard Data Structure

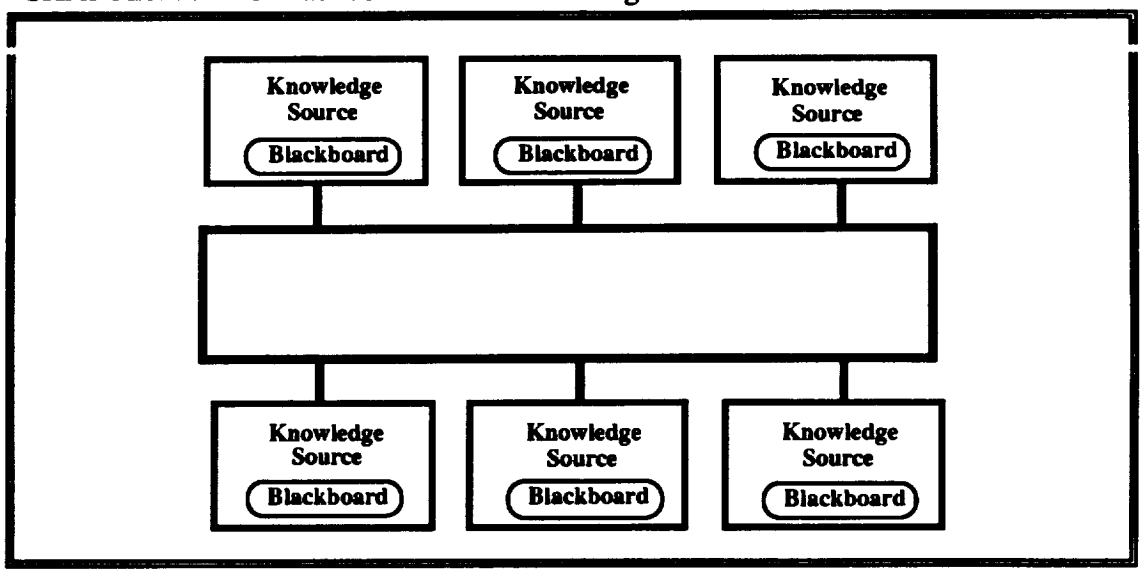

Figure 2.3. Distributed Blackboard System with Distributed Blackboard Data Structure

2.5.1 Centralized Blackboard Data Structure

How a problem is distributed across the available hardware platforms and how tightly the knowledge sources are coupled has a significant impact on the system's performance. Currently, no techniques exist to aid the designer in decomposing a problem into a set of knowledge sources and in evaluating the preliminary design. If the knowledge sources are tightly coupled, a large number of messages will be passed and blackboard consistency becomes a major concern. If the knowledge sources are highly independent but share large amounts of data, communication delays will limit concurrency as nodes are forced to wait for information.

2.5.1.1 Poligon

Rice, Aiello and Nii[20] present several options for gaining speedups in a distributed blackboard system.

1) Eliminate the centralized scheduling mechanism

2) Optimize system design for a distributed memory, message-passing hardware

3) Distribute the data across the blackboard to reduce hotspots

Items one and three are directly applicable to both parallel blackboard systems and distributed blackboard systems. Item two is not a trivial task due to the lack of existing algorithms or tools to perform the optimization. In many cases, the execution of the system's knowledge sources is so irregular, due in part to the use of opportunistic problem solving techniques, that a complete optimization is not possible.

Poligon[21] is based on a distributed memory hardware model when each processor is viewed as a blackboard node. They define a blackboard node as follows:

> "a blackboard node is a process on a processor, surrounded by a collection of processors able to service its requests to execute rules."[22]

The implicit assumption in this definition is that all knowledge sources are rule–based systems. This assumption may severely limit the performance of systems implemented using Poligon, and limits the types of problems it is suited to address. Poligon provides no user accessible data locks, nodes are automatically locked during read and write operations. The blackboard architecture is designed to prevent deadlock from occuring, but livelock can still occur. Processing nodes are capable of evaluating their own performance. The nodes assess each request to modify their local state. The node determines if it should perform the update or take other action based on its progress towards solving the current problem.

Rice, Aiello and Nii draw several conclusions from their experiments using Poligon. Their first conclusion is :

> "The difficulty of implementation of applications is due largely to the divergence of implementations of serial blackboard systems from the pure blackboard model in order to make implementation and programming more manageable."[23]

They state that the applications they tested did not have enough data parallelism to support pipeline parallelism so the effect of pipeline parallelism could not be fully evaluated. Data communication costs were not viewed as being excessive, as long as data flowed through the pipelines. Hardware supporting a non-blocking message passing protocol was used for communication. This type of protocol supports high speed data transfers but may result in data inconsistencies. The final point presented may be the most important:

> "The use of a central controller to determine which knowledge sources to run in parallel drastically limited the speedup possible, no matter how many knowledge sources were executed in parallel."[24]

2.5.1.2 The Blackboard Server Approach

Daniel Corkill[25] also describes a blackboard server approach to developing a distributed blackboard system. The blackboard server approach implements a centralized shared blackboard. The approach does not allow multiple access paths to the blackboard and maintains a single version of each blackboard data object. The blackboard is divided into non-overlapping blackboard responsibility areas and a unique blackboard server is allocated to each area. The knowledge sources send all blackboard interaction requests to the appropriate server, which then queues the request for processing. This implementation requires that all knowledge sources know which blackboard server handles which specific data elements on the blackboard. Corkill discusses two blackboard partitioning extremes. In one case a single node can act as a server for the whole blackboard. In the other extreme a node can be assigned to act as a server for a disjoint section of the blackboard. The correct balance between these two extremes for each unique application may be difficult to determine. If the required message routing information is not made available to all of the knowledge sources, then all of the data is not visible to all of the knowledge sources and opportunistic problem solving performance will degrade.

The blackboard server approach may also suffer from several performance problems. If hotspots in memory occur, some blackboard servers may become overloaded while others are idle. Queuing the memory requests adds a serialized component to the memory access system and additional system overhead. The blackboard server approach outlined by Corkill is based on some partitioning of blackboard data structure. It may be more efficient to assign a unique knowledge source server to each knowledge source. Assigning a unique knowledge source server does not require message routing information and should reduces hotspots on the blackboard. This does not require queuing of blackboard data requests and the associated system overhead.

2.5.2 Distributed Blackboard Data Structure

To incorporate message-passing into a distributed blackboard system the central blackboard controller must handle all incoming data requests. This requirement is

nontrivial and may reduce the use of opportunistic problem solving by forcing serialized message handling, creating a bottleneck at the blackboard. Several methods of distributing the blackboard data structure have been proposed to address this problem.

Distributing the blackboard data structure introduces several subtle changes in the way knowledge sources function. One added requirement is that some entity, either the blackboard controller or the knowledge sources themselves, must know where all of the data a knowledge source requires can be found, and who the knowledge source must notify when it completes execution. Without the use of specialized knowledge source organization and knowledge source connectivity analysis techniques, the affect of this requirement cannot be assessed during the system design phase. This requirement is nontrivial and may reduce the use of opportunistic problem solving by forcing lockstep execution of knowledge sources.

2.5.2.1 Functionally Accurate, Cooperative Communication

To address problems caused by having a distributed blackboard data structure Lesser and Corkill[26] have proposed a Functionally Accurate, Cooperative (FA/C) communication system for building distributed blackboard systems. The system utilizes cooperating instances of a blackboard system running concurrently and communicating using a global blackboard. Each knowledge source, or processing node, can be viewed as an instance of a blackboard system. "Functionally accurate" refers to the ability of the system to generate "acceptably accurate" solutions without requiring that all of the intermediate results shared by the knowledge sources (processing nodes) are correct and consistent. "Cooperative" refers to the iterative style of knowledge source interaction. Each knowledge source produces tentative results that may be incorrect, incomplete, or inconsistent with the tentative results produced by other knowledge sources. Consistencies between the tentative results can be seen as reinforcing the results produced by the other knowledge sources. The hope of this approach is that significantly less communication is required to exchange tentative results instead of passing all of the raw data and processing the results. Blocking and synchronization overhead can be reduced or eliminated. This should increase parallelism and opportunistic problem solving. FA/C was developed to address the problem of maintaining semantic consistency on the blackboard data structure. No test results were presented that measured the effectiveness and cost of implementing this

type of system, but FA/C communication does show some potential for reducing message-passing costs and "lockstep" style system execution.

2.5.2.2 The Distributed Blackboard Approach

In the distributed blackboard approach Corkill[27] describes a system where each processing node has a separate local blackboard that contains the local data objects and copies of all data objects received from other nodes. The distribution of the blackboard data objects is an important design consideration for this type of system. The user must decide if local blackboard data structures will "overlap" and on the level blackboard data object consistency that will be implemented. Blackboard data structures "overlap" if two or more blackboard data structures have a copy of a blackboard data object.

There is a direct relationship between the degree of overlap of the blackboard data structures and the communications costs required to maintain blackboard data object consistency. Allowing blackboard data objects to overlap increases the cost of maintaining blackboard data structure consistency. An update message must be sent to each node that "shares" a blackboard data object. Blackboard overlap also increases the chance that nodes will process inconsistent data, resulting in an incorrect conclusion.

The benefit of not allowing backboard data structures to overlap is that a single copy of each blackboard data object exists making blackboard consistency easier to maintain. This data-object sharing approach results in increased communication costs if a large number of nodes share data objects. Each node has to generate a request for a data object stored on a non–local node every time the data object is needed. These communications costs can become a significant part on a node's execution time. It is clear that the amount of blackboard overlap and the required level of blackboard consistency directly affect a systems communications costs and overall performance. Design, simulation, and analysis techniques are required to balance these costs before a system is implemented. Inter–node communications costs and the cost of blackboard consistency are a function of the implementation of a distributed blackboard data structure, and are not inherent in the blackboard system problem solving paradigm.

2.5.2.3 "Virtual" Blackboard Systems

V. Jagannathan[28] proposes a "Virtual Blackboard" for reducing communication costs when using a distributed blackboard data structure. In this model, all of the

knowledge sources see a single blackboard, even though the blackboard is actually distributed among the cooperating blackboard "instances." The blackboard system is divided into blackboard instances. Each instance consists of a blackboard and a set of knowledge sources. Each blackboard instance contains a subset of the current state information. The virtual blackboard instances do not overlap. Concurrency is restricted to the blackboard level and is not available at the knowledge source level. This virtual blackboard model reduces to a set of serial blackboard systems executing concurrently on a distributed architecture machine and communicating using a message-passing system.

The virtual blackboard model uses message passing to allow each knowledge source to see and access any part of the blackboard. The use of non–overlapping blackboard instances reduces data consistency costs, but increases message passing costs when knowledge sources must access non–local data objects. Jagannathan states:

> "If data is *judiciously decomposed* among multiple blackboard instances, such that knowledge sources within one blackboard instance by and large access the blackboard local to it, the blackboard will be less of a bottleneck."[29]

Unfortunately, he does not provide any design or analysis techniques to aid the user in "judiciously decomposing" the blackboard data structure. Decomposing the blackboard is a difficult task, and an improper decomposition results in increased message passing costs. The increased message passing costs may overcome any performance increase gained by concurrent execution of the linked blackboard systems.

The virtual blackboard system provides three methods for synchronizing knowledge source execution:

1) Trigger Conditions. Events can set trigger conditions that determine when a knowledge source should be fired. A knowledge source can generate a trigger condition that will cause another knowledge source to be activated.

2) Preconditions. Preconditions can be used as a filter to "guard" a knowledge source that has been triggered by another knowledge source.

3) Control Knowledge. A centralized controller can be used to order knowledge source execution. This approach adds the overhead on a

centralized controller and may reduce the opportunity for concurrent knowledge source execution.

The performance data presented for the virtual blackboard system is inconclusive and Jagannathan states that further research is being conducted. The current implementation of the virtual blackboard system has several problems. Problems caused by an improper decomposition of the blackboard have already been discussed. The system's use of blackboard level locks to maintain consistency reduces the level of concurrency available to the user. The system implements a deadlock detection system but cannot detect knowledge source feedback loops. Infinite knowledge source looping occurs when a set knowledge sources form a feedback loop, with each knowledge source activating the other next in the loop when it completes execution.

2.5.3 Conclusions

These approaches to concurrent blackboard systems require the user partition the blackboard to increase system performance, but do provide techniques to aid in the partitioning. The systems that allow overlapping of blackboard data objects add to the systems overhead and increase the chance of blackboard consistency problems. These approaches are best suited to "*mega-blackboard systems*", systems built using blackboard systems as the knowledge sources.

2.6 Conclusions

This chapter presents the blackboard problem-solving model and the original serial implementations of blackboard systems. The cost of adding the centralized control module to serial implementations of blackboard systems has a detrimental effect on the performance of the blackboard systems. The centralized control module is not a part of the original blackboard system problem-solving model. The results of the research presented shows that the development of concurrent blackboard systems requires a unified approach, combining techniques from the AI field and from the Operating Systems/Concurrency Control fields.

Endnotes for Chapter Two

[1] Oliver Selfridge. Pandamonium: A Paradigm for Learning. *Proceedings of the Symposium on the Mechanization of Thought Processes*: 511 - 529, 1959

[2] Allen Newell. Some Problems of Basic Organization in Problem-Solving Programs. In M. C. Youvits, G. T. Jacobi, and G. D. Goldstein (editors), Conference on Self-Organizing Systems, pages 393 -423. Spartan Books, Washington D.C., 1962

[3] Herbet A. Simon. Scientific Discovery and the Psychology of Problem Solving. In *Models of Discovery*. D. Reidel Publishing Company, Boston, Mass, 1977.

[4] Nii, 1986..

[5] H. Penny Nii; Nelleke Aiello; and James Rice, "Frameworks for Concurrent Problem Solving: A Report on Cage and Poligon." Presented at AAAI-88 Workshop on Blackboard Systems, St Paul Minn. In *Blackboard Systems*, ed. R. S. Englemore & A. J. Morgan, Addison Wesley, 1988.

[6] H. Penny Nii; Nelleke Aiello; and James Rice, 1988

[7] Glenn Pearson, "Mission Planning within the Framework of the Blackboard Model." In *Blackboard Systems*, R. S. Englemore & A. J. Morgan ed; 1988. Reprinted, with permission, from *Expert Systems in Government Symposium*. Ed. by Kamal N. Karna. 10.Englemore, R. S.; and Morgan A. J. : Conclusion. In *Blackboard Systems*, ed. R. S. Englemore & A. J. Morgan, Addison Wesley, 1988.

[8] R. S.Englemore and A. J. Morgan: Conclusion. In *Blackboard Systems*, ed. R. S. Englemore & A. J. Morgan, Addison Wesley, 1988.

[9] McManus, 1990

[10] Ibid.

[11] Daniel D. Corkill: Design Alternatives for Parallel and Distributed Blackboard Systems. Presented at AAAI-88 Workshop on blackboard systems, St. Paul Minn. In *Blackboard Architectures and Applications*, V. Jagannathan et al. ed; 1989. Academic Press, Inc.

[12] Ibid.

[13] Ibid.

[14] James Rice; Nelleke Aiello; and H. Penny Nii,: "See How They Run... The Architecture and Performance of Two Concurrent Blackboard Systems." In *Blackboard Architectures and Applications*, V. Jagannathan ed; 1989. Academic Press, Inc.

[15] Corkill, 1989

[16] Nii, 1986

[17] James Rice; Nelleke Aiello; and H. Penny Nii, 1989

[18] Corkill, 1989

[19] Roberto Bisiani ; and A.Forin, "Parallelization of Blackboard Architectures and the Agora System". In *Blackboard Architectures and Applications*, V. Jagannathan et al. ed, Academic Press, Inc 1989.

[20] Nii, H. Penny; Aiello, Nelleke; and James Rice, 1988.

[21] Ibid.

[22] Ibid.

[23] Ibid.

[24] Ibid.

[25] Corkill, 1989.

[26] Lesser and Corkill, 1983

[27] Corkill, 1989.

[28] V. Jagannathan: "Realizing the Concurrent Blackboard Model.", Technical Report BCS-G2010-61, Boeing Advanced Technology Center, P.O. Box 24346, M.S. 7L-64, Seattle Washington, In *Blackboard Architectures and Applications,* V . Jagannathan et al. ed; 1989. Academic Press, Inc.

[29] Ibid.

Chapter 3

A Formal Model for Blackboard Systems

Blackboard systems can be viewed as a set of cooperating intelligent agents, working to solve a common problem. Each intelligent agent, known as a knowledge source, independently monitors the global blackboard data structure and determines when it can advance the current solution of the problem. Based on its current inputs, the knowledge source computes a set of outputs and posts the outputs on the blackboard. A description of the blackboard data structure, the function computed by each knowledge source, and the knowledge source's input and output variables are sufficient to create a formal model of a blackboard system.

3.1 Formal Definition of a Blackboard System

All blackboard systems can be modeled as a blackboard data structure and a set of cooperating knowledge sources. The blackboard data structure contains all shared blackboard data objects and can be accessed by all of the knowledge sources. The blackboard also contains all data used to determine when a knowledge source can be activated. Knowledge sources are processes that take inputs from the blackboard, perform some computation, and then place results on the blackboard.

Definition 3-1: *Blackboard Data Object*

A *blackboard data object*, d_j, consists of a value.

♣

Each blackboard data object is of a predefined blackboard data object type, and each blackboard data object type has range of legal values.

Definition 3-2: *Blackboard Data Structure*

A *blackboard data structure* is a global data structure consisting of a set of blackboard data objects, $\{d_1,\ldots,d_j\}$, used to represent the problem domain.

♣

The blackboard is used to allow inter–knowledge source communication and acts as a shared memory that is visible to all of the knowledge sources.

Definition 3-3: *Knowledge Source*

A *knowledge source* consists of a set of input variables, IV = $\{iv_1,\ldots, iv_n\}$, a set of input conditionals, IC = $\{ic_1,\ldots, ic_n\}$, a set of output variables, OV = $\{ov_1,\ldots, ov_m\}$, a description of the computation performed by the knowledge source, F, a set of preconditions, PR = $\{pr_1,\ldots, pr_k\}$, a set of postconditions, PT = $\{pt_1,\ldots, pt_k\}$ and a input queue, IQ.

♣

Informally, a knowledge source's input conditionals are a set of boolean variables used to notify a knowledge source when one of its input variables has been updated. The preconditions are a set of boolean functions that all must be TRUE for a knowledge source to be activated, and the postconditions are a set of boolean functions that all must be TRUE for a knowledge source to post the result of its computation to the blackboard. If all of a knowledge source's activation conditions are met while the knowledge source is executing, the input queue is used to store the knowledge source's input variables. These terms are defined in detail later in this chapter.

There are two classes of knowledge source input variables: *explicit* input variables and *generic* input variables. An explicit input variable specifies a single, unique blackboard data object that is used as the input variable to a knowledge source. A knowledge source can only use the blackboard data object specified by the explicit input variable as a valid input. A generic input variable specifies a class or type of blackboard data object that can be used as the input variable to the knowledge source. The knowledge source will accept an instance of a blackboard data object of the specified class as an input variable. The use of generic input variables allows development of knowledge sources that function on a class of blackboard data objects. Knowledge sources can be classified by their input variables: *Explicit Knowledge Sources* have only explicit input variables, *Mixed Knowledge Sources* have both explicit and generic input variables, and *Generic Knowledge Sources* have only generic input variables.

Definition 3-3.1: *Knowledge Source Input Conditionals*

The *input conditionals*, IC, for a knowledge source are a set of boolean variables $\{ic_1,\ldots, ic_n\}$, one for each generic or explicit blackboard data object that is an input variable for the knowledge source.

♣

For each knowledge source, ks_i, the set of knowledge source input conditionals, $\{ic_1,\ldots, ic_n\}$, define the current state of the set of input variables for the knowledge source. The knowledge sources input conditionals for knowledge source ks_i contain a boolean flag for each of the input variables for ks_i. Each knowledge source stores the current value of its input conditionals locally. At knowledge source activation all of the knowledge source's input conditionals are reset to FALSE. The input conditional for an explicit input variable is set to TRUE when that specific input variable is updated, the input conditional for a generic input variable is set to TRUE when an blackboard data object of the specified class is updated.

Definition 3-3.2: *Knowledge Source Preconditions*

The *knowledge source preconditions*, PR, are a set of preconditions, $\{pr_1,\ldots,pr_k\}$, that must be TRUE for a knowledge source to be activated.

♣

The knowledge source preconditions are a set of boolean functions that access the current values of any collection of blackboard data objects. The set of preconditions is defined by the user to include application–dependent conditions. Once a precondition is evaluated the precondition holds that value until the knowledge source is signaled to reevaluate the boolean function. Knowledge sources are signaled to evaluate their preconditions when an input data object of the precondition function is updated on the blackboard.

A precondition is different from an input conditional. A precondition is an activation condition based on the state of a blackboard data object, or a collection of blackboard data objects. These blackboard data objects may or many not be inputs to the function computed by the knowledge source. Input conditionals are used to notify a knowledge source that its input variables have been updated on the blackboard and the

knowledge source is ready to activate. Preconditions are used to block the activation of knowledge sources that are ready to execute until a set of user defined conditions are TRUE. The set of knowledge source preconditions may be the empty set.

Definition 3-3.3: *Knowledge Source Postconditions*

The *knowledge source postconditions*, PT, are a set of postconditions, $\{pt_1,\ldots, pt_p\}$, that must be true for a knowledge source activation to post output to the blackboard.

♣

The set of knowledge source postconditions is application–dependent and must be created by the user. The use of postconditions allows a knowledge source to insure that the results of a computation are still valid and to keep confusing or conflicting data from being posted on the blackboard. The set of knowledge source postconditions may be the empty set.

A knowledge source cannot be activated until all of the knowledge source's input conditionals are TRUE, and all of the knowledge source's preconditions are TRUE. Once a knowledge source has been activated the knowledge source is ready to be executed. When a knowledge source completes execution it updates its output data objects on the blackboard. The act of updating a blackboard data object causes the input conditionals for all knowledge sources that access the blackboard data object to be set to TRUE. All knowledge sources that use the data object as an input to a precondition are notified that the data object has been updated.

Definition 3-4: *Knowledge Source Activation*

A *knowledge source activation* is an instantiation of a knowledge source. A knowledge source can only be activated when all of the blackboard data objects that it uses as inputs have been updated and the associated input conditionals, $\{ic_1,\ldots, ic_n\}$, are TRUE, and all of the knowledge source's preconditions $\{pr_1,\ldots, pr_n\}$ are TRUE.

♣

When activated, a knowledge source performs an atomic read operation to copy all required input data objects, and stores them in its input variables. The input variables

are then placed in the knowledge source's input queue and the knowledge source's input conditionals are reset to FALSE. The activation of a knowledge source is an atomic operation.

Definition 3-5: *Knowledge Source Execution*

A *knowledge source execution* is the execution of a knowledge source's computation. A knowledge source can only be executed when when the knowledge source's input queue is not empty.

♣

All knowledge sources are uninterruptable processes. Once a knowledge source starts execution it runs to completion. All knowledge source input and output operations are atomic operations. Knowledge sources are serializable, and knowledge sources allow multiple, concurrent read operations. Knowledge source serialization can be achieved using a two-phase locking protocol or read/write synchronization. Executing a knowledge source is an atomic operation. At completion a knowledge source checks its input queue. If there are knowledge source activations queued for execution the knowledge source is executed with the inputs from the top of the input queue and the input queue is updated by removing the top element from the queue.

The blackboard system model has three types of knowledge sources: *sensors*, *actuators*, and *knowledge processors*. Sensors take their inputs from external sources and place their outputs on the blackboard. Actuators take their inputs from the blackboard and do not generate blackboard data objects as outputs. Knowledge processors take their inputs from the blackboard and place their outputs on the blackboard.

Definition 3-6: *Sensor*

A *sensor*, ks_i, is a specialized type of knowledge source that handles inputs from external sources. Each sensor has a set of explicit input variables , $IV = \{iv_1,\ldots, iv_n\}$, and an empty set of postconditions, $PT = \phi$.

♣

The inputs to a sensor are generated from a stochastic model of the sensor's domain, from pre-recorded actual sensor data, or from actual real-time sensor data. At

completion, a sensor performs an atomic write operation to post its output variables on the blackboard. All sensors have explicit external input variables, so all sensors fall in the class of explicit knowledge sources.

Definition 3-7: *Actuator*

A *actuator*, ks_i, is a specialized type of knowledge source that uses blackboard data objects as inputs but does not update data objects on the blackboard. Each actuator has a set of input variables, $IV = \{iv_1,\ldots, iv_n\}$, an empty set of output variables, $OV = \phi$, and an empty set of postconditions, $PT = \phi$.

♣

Actuators take data objects from the blackboard, perform a computation, and modify their local state. Actuators do not update blackboard data objects on the blackboard. Actuators can have generic or explicit input variables, so they can be of any of the three knowledge source classes; explicit, mixed, or generic.

Definition 3-8: *Knowledge Processor*

A *knowledge processor*, ks_i, is a specialized type of knowledge source. Knowledge processors take all of their input directly from the blackboard.

♣

At completion, a knowledge processor tests its postconditions. If the postconditions are TRUE, the knowledge processor execution performs an atomic write operation to update its output variables on the blackboard. If the postconditions are not valid, the knowledge processor activation discards the output and terminates. Knowledge processors can have generic or explicit input variables, so they can be of any of the three knowledge source classes; explicit, mixed, or generic.

We can now define a blackboard system:

Definition 3–9: *Blackboard System*

A *blackboard system*, B, is a tuple $\langle X, P, \beta, I_s, \Theta\rangle$

- X is a set of blackboard data objects,
  $X = \{d_1,\ldots,d_i\}$;

- P is the set of blackboard data object states:
  $P = V_1 \times V_2 \times \ldots \times V_i$
  where V_q is a set of all valid values for blackboard data object d_q;

- β is the set of knowledge sources. $\beta = \{ks_1, \ldots, ks_j\}$; each knowledge source's domain is a subset of P, and its range is a subset of P;

- I_s is an *i*–vector describing the *i* initial values of the blackboard data objects, so $I_s \in P$;

- Θ is a relation on β, $\Theta \subset \beta \times \beta$.

  $\langle ks_j, ks_k \rangle \in \Theta$ if and only if $\exists\, d_j \in X$ such that $d_j \in OV(ks_j) \wedge d_j \in IV(ks_k)$.

  If $\langle ks_j, ks_k \rangle \in \Theta$ we say that ks_k is a *successor* of ks_j, and ks_j is a *predecessor* of ks_k.

♣

3.2 Knowledge Source Activation Graph

For a given blackboard system, B, we can form a Knowledge Source Activation graph.

Definition 3–10: *Knowledge Source Activation Graph*

A *Knowledge Source Activation Graph,* K, for a blackboard system B, is a directed graph formed using the input set and output set for each knowledge source ks_k in β. A node in the graph is formed for each knowledge source in β. The arcs in the graph are generated using the input and output sets for each knowledge source. An outgoing arc is generated for each output, ov_m, for each knowledge source ks_k in β. The arcs are connected to all knowledge sources in β that use the output data object, ov_m, as an input. Incoming arcs are then generated for all sensor inputs for each knowledge source in β.

♣

Each knowledge source ks_j in β corresponds to a node in the knowledge source activation graph K. The arcs in the graph represent the flow of data objects between the

knowledge sources. Each arc in the graph represents either a specific or generic blackboard data object. If the activation graph of a blackboard system is acyclic, the blackboard system is safe and deadlock cannot occur. If any cycles exist in the activation graph, unsafe states exist in the system and deadlock may occur between the associated knowledge sources. Cycles also highlight a circular relationship, or feedback loop, between the connected knowledge sources.

Definition 3–11: *Safe Blackboard System*

A *Safe Blackboard System, B*, is blackboard system with an acyclic knowledge source connectivity graph

♣

The knowledge source activation graph for a safe blackboard system contains no closed paths, and the blackboard system cannot deadlock. A safe blackboard system is a *feed–forward* system, where all data objects flow in one direction, and there are no feedback loops in the system.

3.3 Examples

This section presents a formal specification of two blackboard systems and the resulting knowledge source activation graphs. The detailed specifications of the systems are contained in Appendix A. These two systems were developed to show the power of the formal model, and the complexity of the problem facing a blackboard system designer. The first system was designed with some common blackboard system problems incorporated in the design. The first blackboard system specification produces a knowledge source activation graph that has cycles and contains unsafe states. The second blackboard system specification redefined the preliminary design, producing an safe acyclic knowledge source activation graph.

3.3.1 Description of the Example Problem

The application used for this example is a simplified fire control system for a modern attack submarine. The example includes a subset of the knowledge sources and blackboard data objects used by the fire control system. The example contains the four knowledge sources described below:

- Ks_1 is a *Weapons Data Preprocessor.* This knowledge sources accepts two sensor inputs,d_1 and d_2, and computes three outputs, d_3, d_4, and d_5.

  d_1 = Line-of-sight-angle to the opponent.

  d_2 = Range to the opponent.

  d_3 = Line-of-sight-angle Rate of Change.

  d_4 = Are the Weapons Parameters Met?

  d_5 = Range Rate.

- Ks_2 performs *Positional assessment.* This knowledge sources accepts d_3, and d_9 as inputs, and computes two outputs, d_6 and d_7.

  d_6 = Is the Weapon Solution Valid?

  d_7 = Is the Position Improving?

- Ks_3 performs *Weapons Control.* This knowledge sources accepts d_4 and d_6 as inputs and computes d_8 as an output.

  d_8 = Is the Weapon Locked and Armed?

- Ks_4 performs *Fire Control.* This knowledge sources accepts d_5, d_7, and d_8 as inputs and computes d_9 as an output.

  d_9 = Trigger.

Knowledge source ks_1 accept the raw sensor inputs, computes the line-of-sight-angle rate of change, determines if the weapons parameters are met, computes the range rate of change, and posts it's outputs to the blackboard. Knowledge source ks_2 reads d_3 from the blackboard, determines if the weapons solution is valid, determines if the position is improving, and posts it's outputs to the blackboard. Knowledge source ks_3 reads d_4 and d_6 from the blackboard, determines if the weapons system is locked

and armed, and posts it's output to the blackboard. Knowledge source ks_4 reads d_5, d_7, and d_8 from the blackboard, determines if the trigger should be pulled, and posts it's output to the blackboard. The functions computed by each knowledge source are included in the specification of B_1.

3.3.2 Blackboard System Specifications

Specification of the blackboard system B_1:

- $X = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9\}$;

- $P = \{V_1 \times V_2 \times V_3 \times V_4 \times V_5 \times V_6 \times V_7 \times V_8 \times V_9\}$;
  $V_1 = V_2 = \ldots V_9 = \{\{U\} \cup \Re\}$ U = Undefined, $\Re$ = the set of real numbers

- $\beta = \{ks_1, ks_2, ks_3, ks_4\}$;
  $ks_1 = \{IV = \{d_1, d_2\},$
  $IC = \{ic_1, ic_2\},$
  $F = \{d_3 = (d_{1past} - d_1 / \text{update rate}),$
  $d_4 = ((d_1 <= 30) \text{ and } (d_2 <= 30{,}000))$
  $d_5 = (d_{2past} - d_2 / \text{update rate})\}$
  $OV = \{d_3, d_4, d_5\},$
  $PR = \phi,$
  $PT = \phi\}.$

  $ks_2 = \{IV = \{d_3, d_9\},$
  $IC = \{ic_3, ic_9\},$
  $F = \{d_6 = ((d_3 >= 4.10) \text{ and } d_9),$
  $d_7 = (d_3 >= d_{3past}) \text{ and } (d_3 <= 4.10))\}$
  $OV = \{d_6, d_7\},$
  $PR = \{pr_1 = (\#\text{ of weapons} \neq 0), pr_2 = (\text{sonar\_valid})\},$
  $PT = \phi\}.$

ks_3 = {IV = {d_4, d_6},
IC = {ic_4, ic_6},
F = {d_8 = ((d_4 = 1) and ($d_6 \neq 1$))},
OV = {d_8},
PR = {pr_3 = (# of weapons ≠ 0), pr_4 = (tracking-system_valid)},
PT = {pt_1 = ($d_5 \neq$ U)}.

ks_4 = {IV = {d_5 ,d_7 ,d_8},
IC = {ic_5, ic_7, ic_8},
F = {d_9 = ((d_5 <= 100) and d_7 and d_8))}
OV = {d_9},
PR = {pr_5 = (aggressive_mode),
pr_6 = (aggressive_mission),
pr_7 = (fire_command)},
PT = ϕ.

- I_S = {d_1 = 1, d_2 = 1, d_3 = U, d_4 = U, d_5 = U, d_6 = U, d_7 = U, d_8 = U, d_9 = U}

- Θ ={⟨ks_1, ks_2⟩, ⟨ks_1, ks_3⟩, ⟨ks_1, ks_4⟩, ⟨ks_2, ks_3⟩, ⟨ks_2, ks_4⟩, ⟨ks_3, ks_4⟩, ⟨ks_4, ks_2⟩}

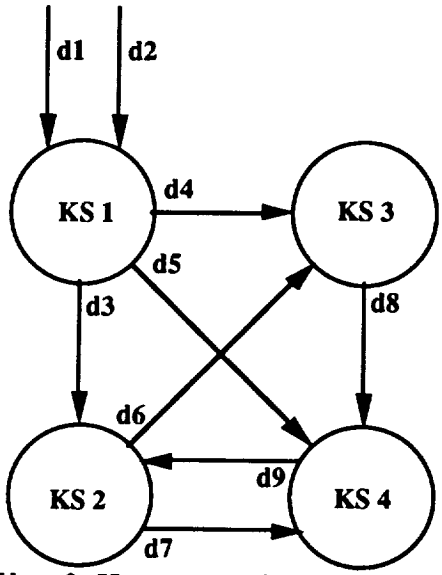

Figure 3.1 Unsafe Knowledge Source Activation Graph

The blackboard system B_1 has two cycles. A direct *loop*, or closed path of length two, exists between the Positional Assessment knowledge source and the Fire Control

Knowledge Source. A closed path of length three exists between the Positional Assessment knowledge source, the Weapons Control knowledge source, and the Fire Control Knowledge Source.

The B_2 specification modifies the functionality of the Positional Assessment knowledge source and the Fire Control knowledge source to remove the feedback link created by d_9, the Trigger data object. Data object d_6, Is the Weapon Solution Valid?, is redefined so that it does not require the value of Trigger, and the Fire Control knowledge source is modified to hold the past value of Trigger and use the past value in its computation.

Specification of the blackboard system B_2:

- $X = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9\}$;

- $P = \{V_1 \times V_2 \times V_3 \times V_4 \times V_5 \times V_6 \times V_7 \times V_8 \times V_9\}$;
  $V_1 = V_2 = \ldots V_9 = \{\{U\} \cup \Re\}$

- $\beta = \{ks_1, ks_2, ks_3, ks_4\}$;
  $ks_1 = \{IV = \{d_1, d_2\},$
  $IC = \{ic_1, ic_2\},$
  $F = \{d_3 = (d_{1past} - d_1 / \text{update rate}),$
  $d_4 = ((d_1 <= 30) \text{ and } (d_2 <= 30{,}000))$
  $d_5 = (d_{2past} - d_2 / \text{update rate})\}$
  $OV = \{d_3, d_4, d_5\},$
  $PR = \phi,$
  $PT = \phi\}.$

  $ks_2 = \{IV = \{d_3\},$
  $IC = \{ic_3\},$
  $F = \{d_6 = (d_3 >= 4.10),$
  $d_7 = (d_3 >= d_{3past}) \text{ and } (d_3 <= 4.10))\}$
  $OV = \{d_6, d_7\},$
  $PR = \{pr_1 = (\#\text{ of weapons} \neq 0), pr_2 = (\text{sonar\_valid})\},$
  $PT = \phi\}.$

ks_3 = {IV = {d_4, d_6},
IC = {ic_4, ic_6},
F = {d_8 = ((d_4 = 1) and ($d_6 \neq 1$))},
OV = {d_8},
PR = {pr_3 = (# of weapons ≠ 0), pr_4 = (tracking-system_valid)},
PT = {pt_1 = ($d_5 \neq U$)}.

ks_4 = {IV = {d_5 ,d_7 ,d_8},
IC = {ic_5, ic_7, ic_8},
F = {d_9 = ((d_5 <= 100) and d_7 and d_8))}
OV = {d_9},
PR = {pr_5 = (aggressive_mode),
pr_6 = (aggressive_mission),
pr_7 = (fire_command)},
PT = ϕ.

- I_s = {d_1 = 1, d_2 = 1, d_3 = U, d_4 = U, d_5 = U, d_6 = U, d_7 = U, d_8 = U, d_9 = U}

- Θ ={⟨ks_1, ks_2⟩, ⟨ks_1, ks_3⟩, ⟨ks_1, ks_4⟩, ⟨ks_2, ks_3⟩, ⟨ks_2, ks_4⟩, ⟨ks_3, ks_4⟩}

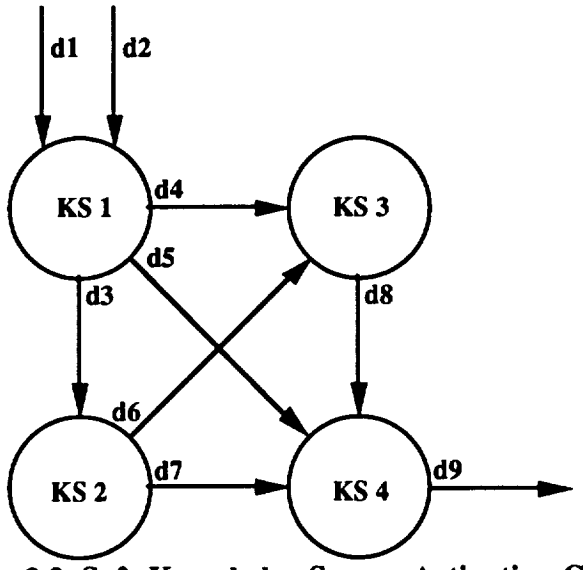

Figure 3.2 Safe Knowledge Source Activation Graph

3.3.3 Blackboard System Execution Traces

The blackboard system B_1 is unsafe, and due to the initial values of the blackboard data objects specified in I_s, the system will livelock after the execution of ks_1. The blackboard system B_2 is safe and will not livelock or deadlock. A detailed trace of the blackboard system execution is included in Appendix A. The trace shows the state of the B_1 from system initialization to livelock.

B_1 livelocks after ks_1 is executed. Ks_2 cannot be activated until ks_4 is activated and computes a value for d_9, setting ic_9 to TRUE. Ks_3 cannot be activated until ks_2 is activated and computes a value for d_6, setting ic_6 to TRUE. Ks_4 cannot be activated until ks_2 is activated and computes a value for d_7, setting ic_7 to TRUE, and ks_3 is activated and computes a value for d_8, setting ic_8 to TRUE. Ks_2 is waiting for ks_4, ks_3 is waiting for ks_2, and ks_4 is waiting for ks_2 and ks_3. Ks_1 will activate and execute until all of its sensor data has be processed. B_1 will halt execution at that time.

By changing the initial values of the blackboard data objects specified in I_s we can show that fact that a system is unsafe does not imply that the system will livelock or deadlock. A trace of the blackboard system execution (included in Appendix A) shows the state of the blackboard system B_1 from system initialization through the completion of the first execution cycle.

B_1 completes the first path through the execution cycle without experiencing liveness problems. The system will continue to execute safely as long as data arrives at the sensor, ks_1. It is the state of the blackboard data objects and sensor inputs that can cause an unsafe system to livelock or deadlock. An unsafe system design does not insure that a system will livelock or deadlock, but reveals where the potential for liveness problems exists.

A trace of the execution of blackboard system B_2 is included in Appendix A. B_2 is a safe blackboard system and does experience liveness problems. The system will execute safely as long as data arrives at the sensor, ks_1.

3.3.4 Conclusions

Preconditions are not included in the static analysis of the knowledge source activation graphs since the values of the precondition input variables required to compute the state of the preconditions are dynamic. The safety of proposed blackboard

system designs is an important consideration. Two methods for guaranteeing knowledge source serializability are supported by the formal model. Knowledge sources use a two-phase locking system for all communications with the blackboard handlers. Communication between parent and child knowledge sources in message passing blackboard systems or meta-blackboard systems is implemented using read/write synchronization. The two-phase locking protocol and read/write synchronization both guarantee serialization.

If it is not possible to develop a safe blackboard system design, the designer must use the knowledge source connectivity information to determine the inter-knowledge source data object dependencies. The data object dependencies can be used to help determine an initial blackboard state that will allow the system to initialize safely. The knowledge source connectivity data can also be used to develop the sets of knowledge source preconditions and knowledge source postconditions.

3.4 Multiple Instances of a Knowledge Source

The Blackboard System Formal Model will support both single instances of a knowledge source and multiple instances of a knowledge source. Although the system will support multiple instances of a knowledge source, the use of multiple instances of a knowledge source is a complex problem that requires further research. The use of multiple instances of a knowledge source may lead to significant data coherence problems on the blackboard data structure if the multiple instances of the knowledge source are updating the same specific blackboard data object.

Past research[1] has exploited the use of multiple instances of a knowledge source executing on a restricted class of generic blackboard data objects. The Cube_CLAWS system demonstrated that multiple instances of a generic knowledge source can be used to develop multiple execution pipelines that increase system performance and decrease system execution times. The success of Cube_CLAWS is a direct result of the use of generic knowledge sources. Each instance of the generic knowledge source executes with a unique set of inputs and posts a unique set of outputs on the blackboard.

In the case of specific or mixed knowledge sources, the situation is different. Multiple instances of these types of knowledge sources execute with multiple instances of a specific blackboard data object as inputs and post multiple instances of a specific blackboard data object as outputs. The posting of multiple copies of a specific

blackboard data object to the blackboard results in data coherence problems on the blackboard unless additional data coherence management software, and the resulting system overhead, is added to the formal model. Lesser and Corkill developed the FA/C system to address this problem. FA/C is intended to allow inconsistent blackboard data objects to be passed among the shared blackboard data structure. The FA/C concept is best suited for distributed blackboard systems. FA/C was tested using knowledge sources that are all similar instances of a of blackboard system designed to solve a single problem. Several instances of the knowledge source process overlapping inputs. The results are used to support or reject conclusions reached by other knowledge sources.

Several past systems (e.g. Cage, Poligon, Erasmus, and GBB)[2, 3, 4] have implemented "multiple instances" of knowledge sources, but these systems have used serial control structures to manage the knowledge source activations. The use of a serial control structure resulted in serial knowledge source executions and protected the systems from blackboard data coherence problems. The formal model for blackboard systems can model these types of systems. Separating a knowledge source's activation from the knowledge source's execution allows the formal model to model the concurrent activation of multiple instances of a knowledge source followed by their serial execution.

Endnotes for Chapter Three

1 McManus,1990

2 Nii, Aiello, and Rice, 1988

3 Jagannathan, 1989.

4 Corkill, 1989.

Chapter 4

Blackboard System Design and Analysis Techniques

Designing and developing a concurrent blackboard system is a difficult process. The designer is trying to balance conflicting goals: achieving a high degree of concurrent knowledge source execution while maintaining both memory and semantic consistency. Several major problems facing the designers of concurrent blackboard systems have been outlined in the literature.

- Speculative parallelism can be very costly and adversely affect system performance.
- Memory management and memory coherence can become difficult and expensive.
- Process locking and concurrency control overhead can become costly and adversely affect opportunistic problem solving.
- Real-time systems require predictable response times.
- Overly restrictive concurrency control can restrict the parallel execution of knowledge sources.
- Dynamically allocating and de–allocating knowledge sources to achieve good processor utilization and load balancing can be expensive, and the overhead required may outweigh the gains of parallel execution if the knowledge sources are small or execute swiftly

Blackboard systems have not attained their apparent potential because there are no established tools or methods to guide in their construction and analyze their performance. Building systems that realize the full potential of the blackboard problem-solving model requires a coherent set of blackboard system design and analysis techniques that address the problems listed above. I have developed a set of design and analysis techniques for concurrent blackboard systems that support the formal model for blackboard systems and address these issues.

4.1 Unified Approach to System Design and Analysis

My preliminary research [1, 2, 3, 4] has shown that efficient implementation of blackboard systems requires that the knowledge sources be highly specialized and highly independent.

Definition 4-1: *Knowledge Source Specialization*

A pair of knowledge sources $\{ks_j, ks_k\}$ are *specialized* if each knowledge source is designed to solve a specific task or subtask .

♣

Designing specialized knowledge sources provides the following benefits:

- Improves performance of rule-based and knowledge-based systems. Specialization results in smaller rule bases and allows for the use of meta-knowledge to guide rule–partition selection, which results in fewer rules being active and faster system performance[5].

- Provides for small knowledge sources that solve specific problems and can cooperate in parallel on large problems. The small, specialized knowledge sources are easy to develop and test since each knowledge source solves a small part of the total problem.

- Reduces input and output overlap, write set conflicts, memory contention at the blackboard level, and concurrency control requirements. Specialization reduces the number of knowledge sources accessing a data object, reducing data object locking and updating overhead.

- Reduces the chance of semantic synchronization being violated because it reduces the chance of one knowledge source's output invalidating the output of another knowledge source.

Definition 4-2: *Knowledge Source Interdependence*

A pair of knowledge sources $\{ks_j, ks_k\}$ are *interdependent* if ks_j provides the majority of the inputs to ks_k.

♣

Definition 4-3: *Knowledge Source Independence*

A pair of knowledge sources $\{ks_j, ks_k\}$ are *independent* if ks_j does not provide the majority of the inputs to ks_k.

♣

Designing independent knowledge sources provides the following benefits:

- Centralizes problem solving functionality, making the function of each knowledge source easier to understand and providing for modular protection.
- Increases opportunistic execution of knowledge sources. Independence reduces the development of unintended knowledge source pipelines. Pipelining results in serial execution of the related knowledge sources, diminishing the opportunities for concurrent knowledge source execution.

A system implemented using a set of highly specialized, highly independent knowledge sources with a data transfer driven control mechanism will be faster, more efficient, and more powerful than current systems.

4.2 Knowledge Source Connectivity Analysis

Knowledge source connectivity analysis is a method for evaluating blackboard system design specifications developed using the formal model for blackboard systems. Connectivity analysis determines the data transfers between the knowledge sources and data migration across the blackboard. Specialization, serialization, and interdependence are evaluated for each knowledge source pair and feedback loops in the design are detected. These techniques evaluate a design specification before the blackboard system is developed. This allows the designer to address knowledge

source interdependence problems, connectivity problems, and design feedback loops as a part of the initial design process.

Knowledge source connectivity analysis measures the *output to input connectivity, functional connectivity*, and *output set overlap* between pairs of knowledge sources. Output to input connectivity is a measure of the interdependence and functional connectivity between pairs of knowledge sources, and output set overlap is a measure of the specialization of pairs of knowledge sources.

Knowledge source pairs with a high level of output to input connectivity are functionally connected, while knowledge source pairs with a low level of output to input connectivity are independent of each other. Knowledge source pairs with a high level of input to output connectivity are serially connected, while knowledge source pairs with a low level of input to output connectivity are independent of each other. Knowledge source pairs that have a high level of output set overlap compute similar functions and lack specialization. Knowledge source pairs with a low level of output set overlap are highly specialized with respect to each other. Interdependence, serialization, and specialization values are used to refine the design of blackboard systems.

Knowledge source connectivity analysis requires a specification of the system developed using the formal model for blackboard systems. For each knowledge source, ks_j, in β we can form a set Ψ_j containing all of the input variables of ks_j; and an output set Φ_j containing all of the output variables of ks_j. Ψ_j consists of both the blackboard inputs and the external inputs. If ks_j is a sensor, Ψ_j may contain only external inputs.

$$\Psi_j = \{iv_1, iv_2, ..., iv_n\} \tag{4.1}$$

$$\Phi_j = \{ov_1, ov_2, ..., ov_m\} \tag{4.2}$$

Once Ψ_j and Φ_j have been formed for all ks_j in β we can compute the sets $\Gamma_{j,k}$ and $\Theta_{j,k}$ for all knowledge source pairs $\{ks_j, ks_k\}$ in β $(j \neq k)$.

$$\Gamma_{j,k} = \Phi_j \cap \Phi_k \tag{4.3}$$

$$\Lambda_{j,k} = \Phi_j \cap \Psi_k \tag{4.4}$$

The set $\Gamma_{j,k}$ is computed to assess the *functional specialization* of a pair of knowledge sources. The cardinality of the set $\Gamma_{j,k}$ for each pair $\{ks_j, ks_k\}$ in β is a

measure of *output overlap* for the pair $\{ks_j,ks_k\}$. Knowledge source pairs $\{ks_j,ks_k\}$ with a large output overlap imply that ks_j and ks_k share a large number of output variables and compute similar functions. Knowledge source pairs $\{ks_j,ks_k\}$ with a low overlap imply that ks_j and ks_k compute different functions. A proposed heuristic to measure knowledge source specialization is to compute a *specialization value*, $\Omega_{j,k}$ for each pair $\{ks_j,ks_k\}$ in β.

Definition 4-4: *Specialization Value*

Specialization values measure the specialization of a pair of knowledge sources, $\{ks_j,ks_k\}$. The specialization value is computed using Equation 4.5

♣

$$\Omega_{j,k} = \frac{\text{card}(\Gamma_{j,k})}{\min(\text{card}(\Phi_j)\,\text{card}(\Phi_k))} \tag{4.5}$$

This heuristic was developed to compute the percentage of overlap between the sets Φ_j and Φ_k. The cardinality of the set $\Gamma_{j,k}$ divided by the minimum of the cardinalities of the sets Φ_j and Φ_k computes a percentage of overlap between the set $\Gamma_{j,k}$ and the smaller of the sets Φ_j and Φ_k. As $\Omega_{j,k}$ approaches 1.0 the output overlap between ks_j and ks_k increases. As $\Omega_{j,k}$ approaches 0.0 the output overlap between ks_j and ks_k decreases. For the limiting cases, if $\Phi_j \supset \Phi_k$ or $\Phi_k \supset \Phi_j$ we know that $\Omega_{j,k} = 1.0$, and ks_j and ks_k compute the same outputs and the knowledge sources are not specialized. If $\Gamma_{j,k} = \phi$ then $\Omega_{j,k} = 0.0$, and the two knowledge sources have no common outputs, and they are highly specialized in relation to each other.

The set $\Lambda_{j,k}$ is computed to assess the connectivity between a pair of knowledge sources. The cardinality of the set $\Lambda_{j,k}$ for each pair $\{ks_j,ks_k\}$ in β is a measure the *output to input connectivity* for the pair $\{ks_j,ks_k\}$. Knowledge source pairs $\{ks_j,ks_k\}$ with a high output to input connectivity imply that ks_k is highly dependent on ks_j for its input variables. Knowledge source pairs $\{ks_j,ks_k\}$ with a low output to input connectivity imply that ks_k's inputs are independent of ks_j's outputs. A proposed heuristic to measure knowledge source output to input connectivity is to compute a *interdependence value*, $\Pi_{j,k}$ for each pair $\{ks_j,ks_k\}$ in β.

Definition 4-5: *Interdependence Value*

Interdependence values measure the functional connectivity between a pair of knowledge sources, $\{ks_j, ks_k\}$. The interdependence value is computed using Equation 4.7

♣

$$\Pi_{j,k} = \frac{(\text{card } \Lambda_{j,k})}{(\text{card } \Phi_j)} \tag{4.7}$$

This heuristic was developed to compute the percentage of the output data objects of ks_j that are used as input data objects by ks_k. The cardinality of the set $\Lambda_{j,k}$ divided by Φ_j computes a percentage of overlap between the set $\Lambda_{j,k}$ and Φ_j. As $\Pi_{j,k}$ approaches 1.0 the output to input connectivity between ks_j and ks_k strengthens and the knowledge sources become more interdependent. As $\Pi_{j,k}$ approaches 0.0 the output to input connectivity between ks_j and ks_k weakens, and the knowledge sources become independent. For the limiting cases, if $\Phi_j \supset \Psi_k$, $\Pi_{j,k} = 1.0$ and ks_j and ks_k have direct output to input connectivity and are interdependent. If $\Lambda_{j,k} = \phi$, $\Pi_{j,k} = 0.0$, and the two knowledge sources have no output to input connectivity and are independent.

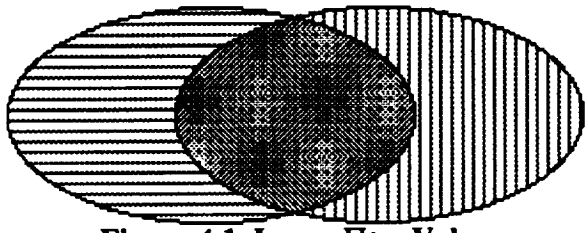

Figure 4.1. Large $\Pi_{j,k}$ Value

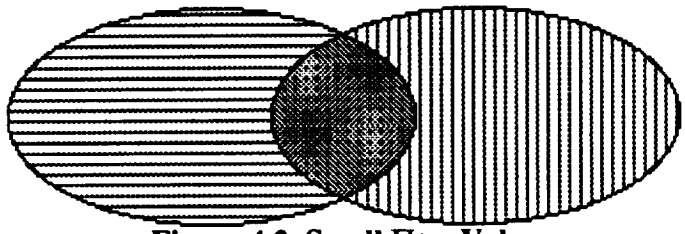

Figure 4.2. Small $\Pi_{j,k}$ Value

Knowledge source pairs that have high interdependence values are good candidates for knowledge source consolidation. The first knowledge source is providing a significant percentage of its outputs to a single knowledge source. The *functionally*

connected pair can be reduced to a single knowledge source that combines the functionality of the two.

Definition 4-6: *Serialization Value*

Serialization values measure the output to input connectivity between a pair of knowledge sources, $\{ks_j, ks_k\}$. The serialization value is computed using Equation 4.8

♣

$$\Sigma_{j,k} = \frac{(\text{card } \Lambda_{j,k})}{(\text{card } \Psi_k)} \tag{4.8}$$

This heuristic was developed to compute the percentage of serialization between the the knowledge sources ks_j and ks_k. This heuristic computes the percentage of the input data objects for knowledge source ks_k that are provided by knowledge source ks_j. The cardinality of the set $\Lambda_{j,k}$ divided by Ψ_k computes a percentage of overlap between the set $\Lambda_{j,k}$ and Ψ_k. As $\Sigma_{j,k}$ approaches 1.0 the serialization between ks_j and ks_k strengthens . As $\Sigma_{j,k}$ approaches 0.0 the serialization between ks_j and ks_k weakens. For the limiting cases, if $\Psi_k \supset \Phi_j$, $\Pi_{j,k} = 1.0$ and ks_j and ks_k have direct serialization. If $\Lambda_{j,k} = \phi$, $\Sigma_{j,k} = 0.0$ and the two knowledge sources are independent and can execute concurrently.

Strongly connected knowledge sources have high serialization values. These knowledge sources form serialized execution pipelines, with each knowledge source blocking for the knowledge sources ahead of it to complete. Unless multiple copies of the serialized knowledge sources are developed the serial pipelines reduce concurrent execution. Weakly connected knowledge sources reduce knowledge source serialization and increase the opportunity for concurrent knowledge source execution.

Knowledge source pairs that have high serialization values are good candidates for knowledge source consolidation. The first knowledge source is providing all of the inputs to the second knowledge source. The *serially connected* pair can be reduced to a single knowledge source that combines the functionality of the two.

4.3 Examples of Connectivity Analysis

This section demonstrates the application of the connectivity analysis techniques using a set of example blackboard systems. B_1 is the specification of the submarine Fire Control system presented in Chapter Three. The other example specifications, B_2, B_3, and B_4, are redesigns of the B_1 specification. The blackboard systems described in this section were used to verify the connectivity analysis techniques and the simulation system.

During the evaluation process tightly coupled knowledge sources are detected using interdependence values and serialization values to measure knowledge source coupling. The blackboard system design is refined to increase the specialization of the knowledge sources and reduce knowledge source serialization. This process is repeated until design constraints are met, or until $\Omega_{j,k}$, $\Pi_{j,k}$ and $\Sigma_{j,k}$ stop decreasing

The blackboard system design is also tested for feedback loops. The effect of feedback loops on the execution of blackboard systems was described in detail in Chapter Three. A knowledge source connectivity graph of the system is evaluated using a cycle detection algorithm[6,7] and all cycles are highlighted. This information is used to redesign the system to remove as many of the cycles as possible.

These techniques highlight potential memory hotspots, system bottlenecks, and knowledge sources where the message-passing costs outweigh the gains made by concurrent knowledge source execution. The techniques can predict the amount of time each knowledge source should require to read/write their blackboard objects, and where potential memory access conflicts can occur between knowledge sources.

4.3.1 Blackboard System B_1 Analysis Results;

The complete results of the connectivity analysis of B_1 is included in Appendix B. The data has been reduced to table form for ease of presentation and discussion. Table 4.1 contains the basic connectivity analysis results for each knowledge source in B_1. The table lists Ψ, Φ, the predecessors, the successors, the cardinality Ψ, and the cardinality Φ for each knowledge source. The data shows that ks_4 is both a predecessor and a successor of ks_2. This uncovers a direct loop between ks_2 and ks_4. The data for ks_4 also shows the direct loop between ks_2 and ks_4.

Table 4.1 B_1 Connectivity Analysis Results

Knowledge Source	ks_1	ks_2	ks_3	ks_4
Ψ:	$\{d_1\ d_2\}$	$\{d_3\ d_9\}$	$\{d_4\ d_6\}$	$\{d_5\ d_7\ d_8\}$
Φ:	$\{d_3\ d_4\ d_5\}$	$\{d_6\ d_7\}$	$\{d_8\}$	$\{d_9\}$
Predecessors:	ϕ	$\{ks_4\ ks_1\}$	$\{ks_1\ ks_2\}$	$\{ks_1\ ks_2\ ks_3\}$
Successors:	$\{ks_2\ ks_3\ ks_4\}$	$\{ks_3\ ks_4\}$	$\{ks_4\}$	$\{ks_2\}$
Cardinality Ψ:	2	2	2	3
Cardinality Φ:	3	2	1	1

Table 4.2 contains the Output Overlap sets (Γ) for each knowledge source in B_1. $\Gamma = \phi$ for all of the knowledge sources.

Table 4.2 B_1 Output Overlaps: Γ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		ϕ	ϕ	ϕ
ks_2	ϕ		ϕ	ϕ
ks_3	ϕ	ϕ		ϕ
ks_4	ϕ	ϕ	ϕ	

Table 4.3 contains the Specialization Values (Ω) for each knowledge source. Since there is no output overlap ($\Gamma = \phi$ for all of the knowledge sources) the specialization values are 0.0. All of the knowledge sources are highly specialized in relation to each other.

Table 4.3 B_1 Specialization Values: Ω

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		0.0	0.0	0.0
ks_2	0.0		0.0	0.0
ks_3	0.0	0.0		0.0
ks_4	0.0	0.0	0.0	

Table 4.4 contains the Output to Input Connectivity sets (Λ) for pair of each knowledge sources.

Table 4.4 B_1 Output to Input Connectivity: Λ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		$\{d_3\}$	$\{d_4\}$	$\{d_5\}$
ks_2	ϕ		$\{d_6\}$	$\{d_7\}$
ks_3	ϕ	ϕ		$\{d_8\}$
ks_4	ϕ	{d9}	ϕ	

Table 4.5 contains the Connectivity Values (Π) for each pair of knowledge sources. For this design, interdependence values of 0.0 to 0.5 are considered acceptable. The data shows that ks_4 and ks_2 are functionally connected (Π = 1.0) and ks_3 and ks_4 are functionally connected (Π = 1.0).

Table 4.5 B_1 Interdependence Values: Π

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		0.33	0.33	0.33
ks_2	0.0		0.5	0.5
ks_3	0.0	0.0		1.0
ks_4	0.0	1.0	0.0	

The low serialization values show that this design produces a system that meets our serialization goals. For this design, serialization values of 0.0 to 0.5 are considered acceptable.

Table 4.6 B_1 Serialization Values: Σ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		0.5	0.5	0.33
ks_2	0.0		0.5	0.33
ks_3	0.0	0.0		0.33
ks_4	0.0	0.5	0.0	

The connectivity graph analysis detects two cycles in the design of blackboard system B_1. As uncovered by the preliminary analysis, a direct *loop*, or closed path of length two, exists between ks_2 and ks_4. A closed path of length three exists between ks_2, ks_3, and ks_4. The system will be redesigned to remove these cycles.

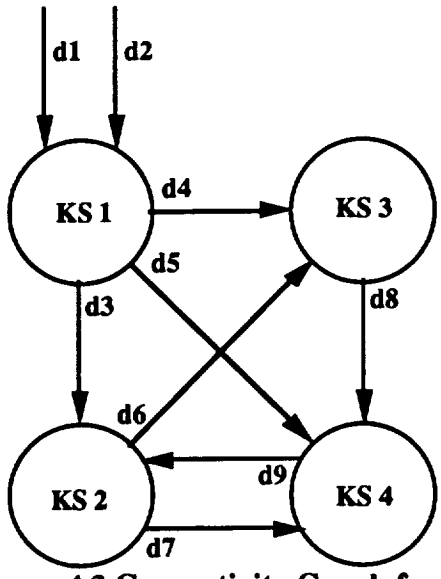

Figure 4.3 Connectivity Graph for B_1

4.3.2 Blackboard System B_2 Analysis Results

Blackboard system specification B_2 is the result of redesigning B_1 to remove the feedback loops. The process used to redesign B_1 is described in Chapter Three. The redesign does not consider knowledge source specialization or interdependence. The complete results of the connectivity analysis of B_2 are also included in Appendix B. Table 4.7 contains the basic connectivity analysis results for each knowledge source in B_2. No direct loops exist in the design.

Table 4.7 B_2 Connectivity Analysis Results

Knowledge Source	ks_1	ks_2	ks_3	ks_4
Ψ:	(d_1 d_2)	(d_3)	(d_4 d_6)	(d_5 d_7 d_8)
Φ:	(d_3 d_4 d_5)	(d_6 d_7)	(d_8)	(d_9)
Predecessors:	ϕ	(ks_1)	(ks_1 ks_2)	(ks_1 ks_2 ks_3)
Successors:	(ks_2 ks_3 ks_4)	(ks_3 ks_4)	(ks_4)	ϕ
Cardinality Ψ:	2	1	2	3
Cardinality Φ:	3	2	1	1

Table 4.8 contains the Output Overlap sets (Γ) for each knowledge source in B_2. $\Gamma = \phi$ for all of the knowledge sources.

Table 4.8 B_2 Output Overlaps: Γ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		ϕ	ϕ	ϕ
ks_2	ϕ		ϕ	ϕ
ks_3	ϕ	ϕ		ϕ
ks_4	ϕ	ϕ	ϕ	

Table 4.9 contains the Specialization Values (Ω) for each knowledge source. Since there is no output overlap ($\Gamma = \phi$ for all of the knowledge sources) the specialization values are 0.0. As with the design on B_1, all of the knowledge sources are highly specialized in relation to each other.

Table 4.9 B_2 Specialization Values: Ω

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		0.0	0.0	0.0
ks_2	0.0		0.0	0.0
ks_3	0.0	0.0		0.0
ks_4	0.0	0.0	0.0	

Table 4.10 contains the Output to Input Connectivity sets (Λ) for pair of each knowledge sources.

Table 4.10 B_2 Output to Input Connectivity: Λ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		$\{d_3\}$	$\{d_4\}$	$\{d_5\}$
ks_2	ϕ		$\{d_6\}$	$\{d_7\}$
ks_3	ϕ	ϕ		$\{d_8\}$
ks_4	ϕ	ϕ	ϕ	

Table 4.11 contains the Interdependence Values (Π) for each pair of knowledge sources. For this design, interdependence values of 0.0 to 0.5 are considered acceptable. The data shows that ks_3 and ks_4 are functionally connected ($\Pi = 1.0$).

Table 4.11 B_2 Interdependence Values: Π

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		.33	0.33	0.33
ks_2	0.0		0.5	0.5
ks_3	0.0	0.0		1.0
ks_4	0.0	0.0	0.0	

Table 4.12 contains the serialization values for each pair of each knowledge sources. The high serialization value, $\Sigma <ks_1\ ks_2> = 1.0$, shows that this design produces a system with a serial component. Ks_2's only input variable is an output from ks_1, so ks_2 can't execute until ks_1 executes.

Table 4.12 B_2 Serialization Values: Σ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		1.0	0.5	0.33
ks_2	0.0		0.5	0.33
ks_3	0.0	0.0		0.33
ks_4	0.0	0.0	0.0	

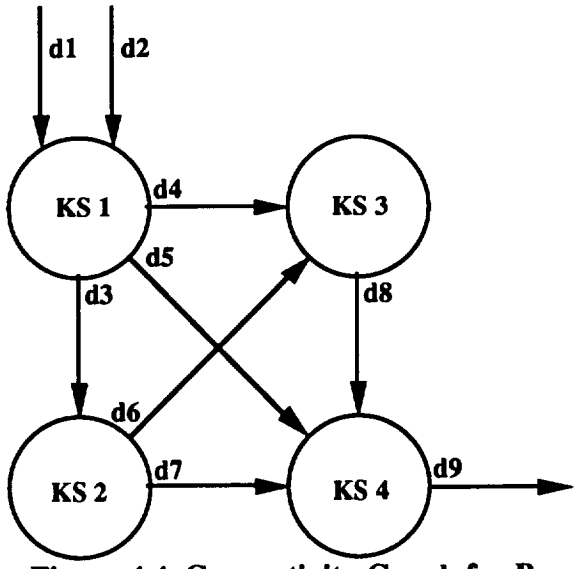

Figure 4.4 Connectivity Graph for B_2

The connectivity graph analysis detects no cycles in the design of blackboard system B_2. The redesign of B_1 did not address the knowledge source interdependence and serialization problems. As a result, the redesigned system still has unacceptable

knowledge source interdependence values and serialization values. Further redesign is required to correct these problems. The application of the redesign techniques used in this example solves only part of the problems in the design of B_1. This approach was used to simplify the example and is not recommended. The designer must address both connectivity and interdependence problems during all phases of the design and analysis process. Using the design and analysis techniques, B_2 can be redesigned. Two examples of redesigning B_2 follow. The system designs, B_3 and B_4 , address both the feedback loop problems and the knowledge source interdependence problems. The specifications and analysis of B_3 and B_4 are included in Appendix B. The redesigned systems redistribute the functionality of the knowledge sources to achieve better design characteristics. Each of the designs is functionally equivalent at the system level.

The design specification for B_3 moves the computation of d_6, Is the Weapons Solution Improving?, from the Positional Assessment knowledge source to the Weapons Data Preprocessor knowledge source. The serial link between the Positional Assessment knowledge source and the Weapons Control Knowledge source is broken, and the two knowledge sources can execute concurrently.

The design specification for B_4 combines the functionality of the Positional Assessment knowledge source with the functionality of the Weapons Data Preprocessor knowledge source. This design has acceptable specialization and interdependence, but the serialization values show that the knowledge sources are serially linked and will execute in a pipeline fashion.

4.3.3 Blackboard System B_3 Analysis Results

The complete results of the connectivity analysis of B_3 are included in Appendix B. Table 4.13 contains the basic connectivity analysis results for each knowledge source in B_3. The data shows that no direct loops exist in the design.

Table 4.13 B_3 Connectivity Analysis Results

Knowledge Source	ks_1	ks_2	ks_3	ks_4
Ψ:	(d_1 d_2)	(d_3)	(d_4 d_6)	(d_5 d_7 d_8)
Φ:	(d_3 d_4 d_5 d_6)	(d_7)	(d_8)	(d_9)
Predecessors:	ϕ	(ks_1)	(ks_1)	(ks_1 ks_2 ks_3)
Successors:	(ks_2 ks_3 ks_4)	(ks_4)	(ks_4)	ϕ
Cardinality Ψ:	2	1	2	3
Cardinality Φ:	4	1	1	1

Table 4.14 contains the Output Overlap sets (Γ) for each knowledge source in B_3. $\Gamma = \phi$ for all of the knowledge sources.

Table 4.14 B_3 Output Overlaps: Γ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		ϕ	ϕ	ϕ
ks_2	ϕ		ϕ	ϕ
ks_3	ϕ	ϕ		ϕ
ks_4	ϕ	ϕ	ϕ	

Table 4.15 contains the Specialization Values (Ω) for each knowledge source. Since there is no output overlap ($\Gamma = \phi$ for all of the knowledge sources) the specialization values are 0.0. The knowledge sources are highly specialized in relation to each other.

Table 4.15 B_3 Specialization Values: Ω

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		0.0	0.0	0.0
ks_2	0.0		0.0	0.0
ks_3	0.0	0.0		0.0
ks_4	0.0	0.0	0.0	

Table 4.16 contains the Output to Input Connectivity sets (Λ) for pair of each knowledge sources.

Table 4.16 B_3 Output to Input Connectivity: Λ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		$\{d_3\}$	$\{d_4\ d_6\}$	$\{d_5\}$
ks_2	ϕ		ϕ	$\{d_7\}$
ks_3	ϕ	ϕ		$\{d_8\}$
ks_4	ϕ	ϕ	ϕ	

Table 4.17 contains the Interdependence Values (Π) for each pair of knowledge sources. For this design, interdependence values of 0.0 to 0.5 are considered acceptable. The data shows that ks_2 and ks_4 are functionally connected ($\Pi = 1.0$) and ks_3 and ks_4 are functionally connected ($\Pi = 1.0$).

Table 4.17 B_3 Interdependence Values: Π

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		0.25	0.5	0.25
ks_2	0.0		0.0	**1.0**
ks_3	0.0	0.0		**1.0**
ks_4	0.0	0.0	0.0	

Table 4.17 contains the Serialization Values (Σ) for each pair of knowledge sources. For this design, serialization values of 0.0 to 0.5 are considered acceptable. The data shows that ks_1 and ks_2 are tightly connected ($\Sigma = 1.0$) and ks_1 and ks_3 are tightly connected ($\Sigma = 1.0$). The high serialization values show that this design produces a system with two serial components. Ks_2's only input variable is an output from ks_1, so ks_2 can't execute until ks_1 executes. Ks_3's input variables are outputs from ks_1, so ks_3 can't execute until ks_1 executes.

Table 4.18 B_3 Serialization Values: Σ

Knowledge Source	ks_1	ks_2	ks_3	ks_4
ks_1		**1.0**	**1.0**	0.33
ks_2	0.0		0.0	0.33
ks_3	0.0	0.0		0.33
ks_4	0.0	0.0	0.0	

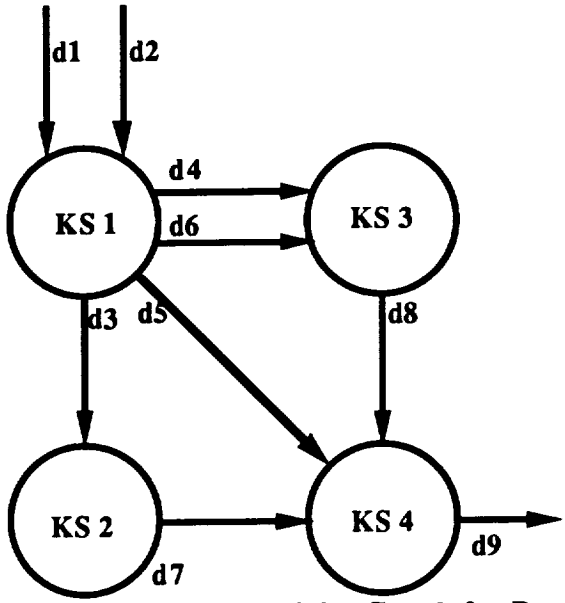

Figure 4.5 Connectivity Graph for B_3

4.3.3 Blackboard System B_4 Analysis Results

The complete results of the connectivity analysis of B_4 are included in Appendix B. Table 4.19 contains the basic connectivity analysis results for each knowledge source in B_4. No direct loops exist in the design.

Table 4.19 B_4 Connectivity Analysis Results

Knowledge Source	ks_1	ks_2	ks_3
Ψ:	$(d_1\ d_2)$	$(d_4\ d_6)$	$(d_3\ d_5\ d_8)$
Φ:	$(d_3\ d_4\ d_5\ d_6)$	(d_8)	(d_9)
Predecessors:	ϕ	(ks_1)	$(ks_1\ ks_2)$
Successors:	$(ks_2\ ks_3)$	(ks_3)	ϕ
Cardinality Ψ:	2	2	3
Cardinality Φ:	4	1	1

Table 4.20 contains the Output Overlap sets (Γ) for each knowledge source in B_4. $\Gamma = \phi$ for all of the knowledge sources.

Table 4.20 B_4 Output Overlap: Γ

Knowledge Source	ks_1	ks_2	ks_3
ks_1		ϕ	ϕ
ks_2	ϕ		ϕ
ks_3	ϕ	ϕ	

Table 4.21 contains the Specialization Values (Ω) for each knowledge source. Since there is no output overlap ($\Gamma = \phi$ for all of the knowledge sources) the specialization values are 0.0.

Table 4.21 B_4 Specialization Values: Ω

Knowledge Source	ks_1	ks_2	ks_3
ks_1		0.0	0.0
ks_2	0.0		0.0
ks_3	0.0	0.0	

Table 4.22 contains the Output to Input Connectivity sets (Λ) for pair of each knowledge sources.

Table 4.22 B_4 Output to Input Connectivity: Λ

Knowledge Source	ks_1	ks_2	ks_3
ks_1		$\{d_4\ d_6\}$	$\{d_3\ d_5\}$
ks_2	ϕ		$\{d_8\}$
ks_3	ϕ	ϕ	

Table 4.23 contains the Interdependence Values (Π) for each pair of knowledge sources. For this design, interdependence values of 0.0 to 0.5 are considered acceptable. The data shows that ks_2 and ks_3 are functionally connected ($\Pi = 1.0$).

Table 4.23 B_4 Interdependence Values: Π

Knowledge Source	ks_1	ks_2	ks_3
ks_1		0.5	0.5
ks_2	0.0		**1.0**
ks_3	0.0	0.0	

Table 4.24 contains the Serialization Values (Σ) for each pair of knowledge sources. For this design, serialization values of 0.0 to 0.5 are considered acceptable. The data shows that ks_1 and ks_2 are tightly connected ($\Sigma = 1.0$) and ks_1 and ks_3 are tightly connected ($\Sigma = 0.67$). The high serialization values show that this design produces a system with two serial components. Ks_2's input variables are outputs from ks_1, so ks_2 can't execute until ks_1 executes. Ks_3's input variables are outputs of ks_1, so ks_3 can't execute until ks_1 executes.

Table 4.24 B_4 Serialization Values: Σ

Knowledge Source	ks_1	ks_2	ks_3
ks_1		1.0	0.67
ks_2	0.0		0.33
ks_3	0.0	0.0	

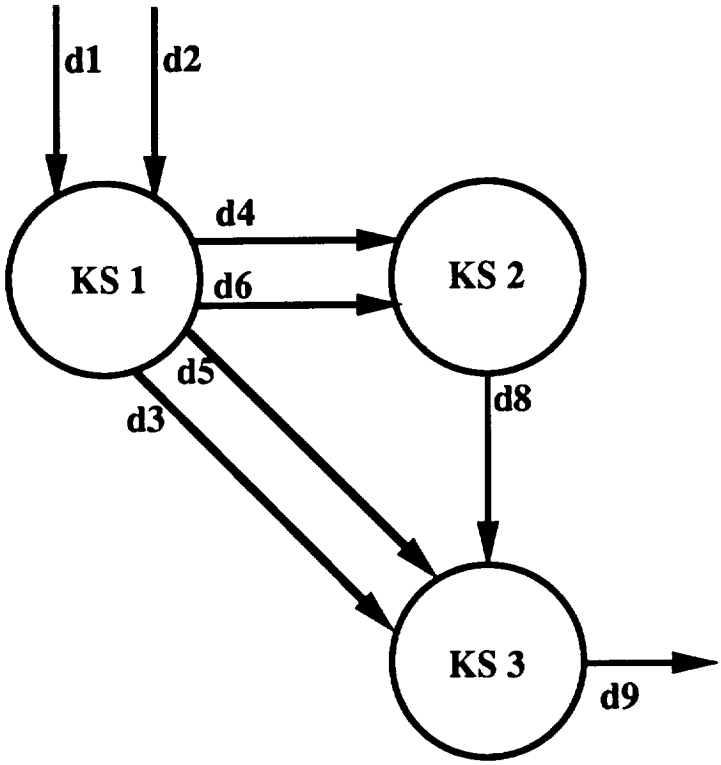

Figure 4.6 Connectivity Graph for B_4

4.4 Conclusions

These examples show the correct application of the design and analysis techniques. They also point out some of the tradeoffs a designer must make. All four of the systems discussed in this chapter are functionally equivalent, but each has its own set of performance characteristics. The designer must analyze each of the designs and determine which tradeoffs will allow them to meet their specific performance requirements.

System B_1 has feedback loop problems that make the design unacceptable. B_2 has no feedback loop problems, but it doesn't allow concurrent knowledge source execution. All of the knowledge sources are serially connected. B_3 has worse interdependence and serialization values for some knowledge sources, but it allows ks_2

and ks_3 to execute concurrently. B_4 has better interdependence and serialization values for some knowledge sources, but like B_1 and B_2 it does not allow concurrent knowledge source execution.

The example problems used here were chosen for their simplicity. Each of the examples was designed to show specific features of the design and analysis techniques on small, tractable problems. The design and analysis tools recognize the simplicity of the system and recommend collapsing the system into a single knowledge source. The power and effectiveness of the techniques will become clear in Chapter Eight, when the techniques are applied to more realistic problems.

The knowledge source connectivity analysis produces a measure of the interdependence between knowledge sources and a measure of the data transfers across the blackboard and through the communications network. From this analysis the designer can determine if the knowledge sources have been partitioned correctly and if the expected level of knowledge source specialization has been achieved. This type of analysis has been done by hand while developing several blackboard systems [8, 9, 10, 11].

Using these techniques a blackboard system design can be modified to gain the desired knowledge source size and specialization and the desired knowledge source output to input connectivity. These techniques are designed to function for generic blackboard systems expressed using the formal model for blackboard systems, and are not application dependent.

Endnotes for Chapter Four

1 Richard M. Hueschen and John W. McManus, 1988

2 John W. McManus and Kenneth H. Goodrich, 1989

3 John W. McManus and Kenneth H. Goodrich, 1990

4 McManus, 1990

5 John W. McManus and Kenneth H. Goodrich, 1989

6 Robert Tarjan: "Depth First Search and Linear Graph Algorithms". *SIAM Journal of Computing*, Vol 1, No. 2, June 1972. pp 146-160.

7 Robert Tarjan: "Enumeration of the Elementry Circuits of a Directed Graph". *SIAM Journal of Computing*, Vol 2, No. 3, September 1973. pp 146-160.

8 Richard M. Hueschen and John W. McManus, 1988

9 John W. McManus and Kenneth H. Goodrich, 1989

10 John W. McManus and Kenneth H. Goodrich, 1990

11 McManus, 1990

Chapter 5

A Simulation Model for Blackboard Systems

A formal Blackboard System Simulation model, based on the formal model for blackboard systems, was developed to evaluate proposed blackboard system designs before they are implemented. The Blackboard System Simulation Model is a serial, discrete event simulation. The simulation model is generic, and can be used to model any blackboard system that can be expressed using the formal blackboard system model.

5.1 Blackboard System Simulation Model

The Blackboard System Simulation Model uses the formal definition of a blackboard system as a basis. Unless specifically redefined in this section, all terms used in this chapter refer to the terms defined for the formal model. We can now define the components of the Blackboard System Simulation Model in further detail:

Definition 5-1: *Blackboard System Simulation Clock*

A *blackboard system simulation clock* , Δ, is an monotonically increasing integer variable.

♣

The blackboard system simulation clock contains the current simulation time.

Definition 5-2: *Blackboard System Simulation Event Queue*

A *blackboard system simulation event queue*, A, is an ordered queue that is used to store the currently executing knowledge sources.

♣

Elements in the event queue are ordered (smallest to largest) by completion time.

The blackboard system simulation model uses the three types of knowledge sources defined in the formal model with the addition of several features required to support the simulation model:

Definition 5-3: *Blackboard System Simulation Knowledge Source*

A *blackboard system simulation knowledge source*, ks_i, adds the following features: an Execution Delay, XD, and a Completion Time, CT.

♣

A knowledge source's execution delay is a constant or a discrete random variable representing the amount of time a knowledge source takes to execute. The knowledge source's completion time is the time (in relation to the global clock) at which the knowledge source will complete execution and update its output variables on the blackboard.

Definition 5-4: *Blackboard System Simulation Sensor*

A *blackboard system simulation sensor*, ks_i, is a specialized type of knowledge source that handles inputs from external sources. Each sensor includes an update interval, UR, and an activation time, AT.

♣

A sensor's update interval is a constant or a discrete random variable that represents how often the sensor's input variables are updated. A sensor is activated only when all of its input conditionals are true and the global clock is equal to the sensor's activation time. When activated, a sensor performs a read operation to copy all of its input data objects to its local input variables, computes an execution delay, XD, and adds that to the activation time, AT, to compute a completion time, CT.

$$CT = AT + XD \tag{5.1}$$

If an instance of the sensor is not currently executing, the sensor is placed in the event queue. If there is an instance of the sensor on the event queue, the sensor's completion time is computed as:

$$CT = CT \text{ (last instance on the queue)} + XD \tag{5.1.1}$$

and the sensor is placed in the event queue.

At execution a sensor performs its computation, places its updated output variables on the blackboard, and computes its next activation time by calculating a new update interval and adding it to the current clock value.

$$AT = \Delta + UR \tag{5.2}$$

The accuracy of the description of the sensor's computation and the accuracy of the execution delay variable directly affect the fidelity of the blackboard system simulation. Increasing the accuracy of these variables increases the accuracy of the simulation model.

Definition 5-5: *Blackboard System Simulation Actuator*

A *blackboard system simulation actuator*, ks_i, is a specialized type of knowledge source that uses blackboard data objects as inputs. Actuators do not update data objects on the blackboard, so $OV = \phi$.

♣

An actuator's execution delay is a constant or a discrete random variable. An actuator is activated only when all of its input conditionals are TRUE and all of its preconditions are TRUE. When activated the actuator performs a read operation to copy all of its input data objects to its local input variables, computes an execution delay, XD, and computes its completion time.

$$CT = \Delta + XD \tag{5.3}$$

If an instance of the actuator is not currently executing, the actuator is placed in the event queue. If there is an instance of the actuator on the event queue, the actuator's completion time is computed as:

$$CT = CT \text{ (last instance on the queue)} + XD \tag{5.3.1}$$

and actuator is placed in the event queue.

When executed an actuator performs its computation and updates its internal state variables. As with the sensors, the accuracy of the description of the computation performed by the actuator, and the accuracy of the execution delay variable directly

affect the fidelity of the blackboard system simulation. Increasing the accuracy of these variables increases the accuracy of the simulation model.

Definition 5-6: *Blackboard System Simulation Knowledge Processor*

A *knowledge processor*, ks_i, is a specialized type of knowledge source.

♣

A knowledge processor's execution delay is a constant or a discrete random variable. A blackboard system simulation knowledge processor is activated only when all of its input conditionals are TRUE and all of its preconditions are TRUE. When activated, the knowledge processor performs a read operation to copy all of its input data objects to tits local input variables, computes an execution delay, XD, and computes its completion time.

$$CT = \Delta + XD \tag{5.4}$$

If an instance of the knowledge processor is not currently executing, the knowledge processor is placed in the event queue. If there is an instance of the knowledge processor on the event queue, the knowledge processor's completion time is computed as:

$$CT = CT \text{ (last instance on the queue)} + XD \tag{5.4.1}$$

and knowledge processor is placed in the event queue.

When executed, a knowledge processor performs its computation and places its updated output variables on the blackboard. As with the sensors and actuators, the accuracy of the description of the computation performed by the knowledge processor and the accuracy of the execution delay variable directly affect the fidelity of the blackboard system simulation. Increasing the accuracy of these variables increases the accuracy of the simulation model.

If multiple knowledge sources are hosted on a single processor, a First-Come-First-Served protocol is used to order knowledge source execution at the processor level. Each knowledge source must include a processor identification tag that identifies which processor the knowledge source requires for execution. A Processor Utilization table is built to track the availability of each of the processors. For each processor in

the system, the Processor Utilization table contains the time at which the processor will become free and can accept a new task. All values in the Processor Utilization table are set to zero at system initialization. If the value in the Processor Utilization table is less than or equal to the global clock, the processor is available for use ("free"). When activated, a knowledge source checks the table to see if the required processor is free. If the processor is free, the knowledge source performs a read operation to copy all of its inputs to the local input variables, computes an execution delay, XD, computes its completion time, and updates the Processor Utilization table with its completion time.

$$CT = \Delta + XD \tag{5.5}$$

$$\text{Table value} = CT + 1 \tag{5.5.1}$$

If the processor is not free, the knowledge source performs a read operation to copy all of its inputs to the local input variables, computes an execution delay, XD, computes its completion time using equation 5.6, and updates the Processor Utilization table with its completion time.

$$CT = \text{Table value} + XD \tag{5.6}$$

$$\text{Table value} = CT + 1 \tag{5.6.1}$$

Multiple instances of a knowledge source executing concurrently are also treated differently. The input data for the knowledge source is not queued waiting for the currently executing instance of the knowledge source to complete execution. Instead, another instance of the knowledge source is created and activated. When activated, the new instance of the knowledge source performs a read operation to copy all of its inputs to the local input variables, computes an execution delay, XD, and computes its completion time.

$$CT = \Delta + XD \tag{5.7}$$

The effects of concurrently executing multiple instances of a knowledge source were presented in detail at the end of Chapter Three.

Definition 5-7: *Blackboard System Simulation Knowledge Source Activation*

A blackboard system simulation knowledge source is *activated* when all of the knowledge source's activation conditions have been met.

When a knowledge source is activated, the knowledge source performs a read operation to copy all of its input data objects from the blackboard to its local input variables, computes its completion time, CT, and is placed on the event queue.

♣

Definition 5-8: *Blackboard System Simulation Knowledge Source Execution*

A blackboard system simulation knowledge source is *executed* when the global clock, Δ, is equal to the knowledge source's completion time.

♣

When a knowledge source is executed, the knowledge source performs its computation and updates its output variables and places them on the blackboard. If several knowledge sources have the same completion time they are executed in the order they appear in the event queue.

Definition 5–9: *Blackboard System Simulation Model*

A *blackboard system simulation,* Σ, is a tuple $\langle \Delta, X, P, \beta, I_S, A \rangle$

- Δ is a global clock.
- X is a set of blackboard data objects,
  $X = \{d_1,\ldots,d_i\}$;
- P is the set of blackboard data object states:
  $P = V_1 \times V_2 \times \ldots \times V_i$
  where V_q is a set of all valid values for blackboard data object d_q
  and $q = 1..\ i$;
- β is the set of knowledge sources. $\beta = \{ks_1,\ldots, ks_j\}$; each knowledge source's domain is a subset of P, and its range is a subset of P;

- I_S is an *i*–vector describing the *i* initial values of the blackboard data objects, so $I_S \in P$;

- A is an ordered queue of all knowledge sources that are currently marked as executable.

♣

5.2 Description of the Simulation Model

The simulation system (Figure 5.1) executes an initialization process when it is activated:

- The blackboard clock is initialized to zero and the blackboard data objects are set to their initial states.

- All of the sensors are evaluated. If a sensor is ready to be activated its execution delay and completion time are computed and the sensor is placed on the event queue.

- Each knowledge source's input conditionals and preconditions are evaluated. If the knowledge source is ready to be activated its execution delay and completion time are computed, and the knowledge source is placed on the event queue.

- For each sensor in ß, if the sensors IC's are TRUE, and Equation 5.8 is TRUE, the sensor computes a CT value, new values for UR and AT, and the sensor is placed on the event queue.

$$AT <= CT_{\text{Head(Event queue)}} \quad \text{and} \quad AT >= \Delta \tag{5.8}$$

The first clause of Equation 5.6 tests to see if the activation time of the sensor is less than the next value of Δ ($CT_{\text{Head(Event queue)}}$ is used to set the next value of Δ). The second clause tests to insure that the sensor should fire after the current time. The result of the calculation is to activate sensors that have an AT between the current value of Δ an the next value of Δ.

The initialization process computes the initial activation times for all of the sensors. All knowledge sources that are ready to run are placed on the event queue. If the event queue is empty the sensor with the earliest activation time is activated. The global clock is updated to the sensor's activation time and the sensor is placed on the event queue. All of the sensors are tested to insure that none should be activated before the first element of the event queue is executed. Any sensors that need to be activated are activated in increasing order of activation time, and the global clock is updated to their activation time.

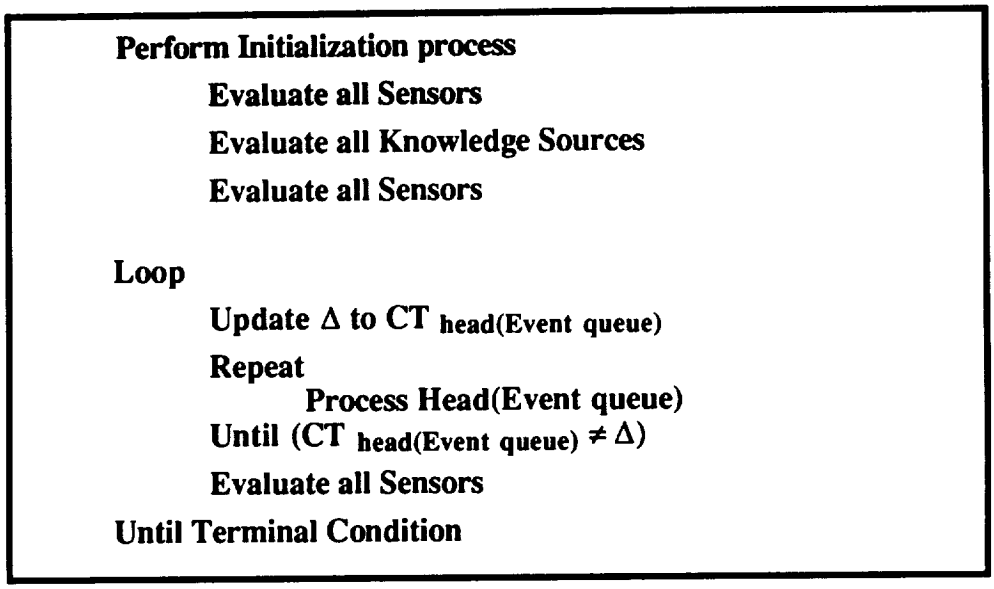

Figure 5.1 Simulation Execution Structure.

The simulation then enters the operate loop:

- The global clock is updated to the completion time of the first element in the event queue.
- The first knowledge source is removed from the event queue and executed. This step is repeated until all knowledge sources that should execute at this time have been processed.
- As described in the discussion of a knowledge processor, knowledge processors are placed on the event queue automatically when all of their IC's are TRUE.
- All sensors are tested to see if they are ready to be activated. For each sensor in ß, if all of the sensors IC's are TRUE, and Equation 5.8 is

TRUE, the sensor computes a CT value, new values for UR and AT, and the sensor is placed on the event queue.

The simulation system then loops back as shown in Figure 5.2. This loop is repeated until a "terminal condition" is signaled. The "Loop ... Until" construct is used to test for a terminal condition at the bottom of the loop. The "terminal condition" may be as simple as the event queue being empty or the clock reaching a predefined end time, or as complex a logical function as the specific application requires.

5.3 Conclusions

An example of the simulation of the blackboard system B_2 is included in Appendix B. The example shows each step on the simulation for the first execution cycle of the blackboard system. The blackboard system simulation model is a valid simulation of any blackboard system expressed using the formal model for blackboard systems. Appendix B discusses the unique conditions that may occur during the simulation of a blackboard system and how the simulation model addresses these special conditions.

The Simulation Model allows the blackboard systems designer to close the Design —> Simulate —> Analyze —> Implement loop. The Formal Model, the Blackboard System Design and Analysis Techniques, and the Simulation Model provide the blackboard system designer with a complete set of techniques for concurrent blackboard systems. Chapter Six present one of the many ways of implementing a set of tools that support the techniques described in Chapters Three, Four, and Five. These tools use the concepts developed in the past three chapters to develop a set of interactive tools for the design, simulation, and analysis of concurrent blackboard systems.

Endnotes for Chapter Five

Chapter 6

The Concurrent Object–Oriented Blackboard System.

The Concurrent Object–Oriented Blackboard System (COBS) system is an implementation of the techniques developed in Chapters Three through Five The basic premise of COBS is that a concurrent blackboard system will achieve the potential of the blackboard model of problem solving and outperform a conventionally designed serial blackboard system. COBS is a method for implementing concurrent blackboard system utilizing of a set of highly independent, highly specialized knowledge sources that cooperate using a shared object-oriented blackboard. COBS was developed using the formal model for blackboard systems presented in Chapter Three as a guideline, and is supported by a set of design and analysis tools, a simulation model, and a code generator. The design tools are implemented using the techniques described in Chapter Four, and the simulation model is implemented using the simulation model described in Chapter Five.

The use of a centralized control modules is one of the major bottlenecks in existing concurrent blackboard systems. Several systems[1, 2, 3, 4, 5] have proposed decentralizing the blackboard system's control module, or in some cases completely removing the control module, but no system has completely eliminated the serial, centralized control module. COBS removes the centralized control module, utilizing an object–oriented approach to implement the blackboard data objects. Daemons are attached to the blackboard data object slots and used to trigger the activation of the knowledge sources when the slot values are updated. This approach removes the need for a centralized control module. The transfer of data objects across the blackboard activates the knowledge sources.

6.1 Description of The COBS Architecture

COBS consists of the following elements: an object-oriented blackboard data structure, a set of highly specialized, highly independent knowledge sources, a set of blackboard handlers, and a daemon driven control structure. Blackboard control and knowledge source selection is achieved using daemons attached to the blackboard data objects. The daemons activate the blackboard handlers when data elements on the blackboard are updated. This daemon–driven control structure removes the need for a

centralized blackboard control mechanism and allows concurrent knowledge source execution. The blackboard handlers control knowledge source activation and provide the knowledge sources read/write access to the blackboard.

COBS is designed to execute on a heterogeneous computer network with a centralized shared memory parallel processor hosting the blackboard data structure. The blackboard data structure is a hierarchical object-oriented data structure. A set of daemon driven blackboard handlers are used to manage the blackboard data objects and the data transfers to and from the knowledge sources. Hosting the object–oriented blackboard and the blackboard handlers on the shared memory parallel processor removes the need for the centralized control structure found in most existing blackboard systems. The blackboard handlers have direct access to the blackboard data structure and can directly monitor the status of the blackboard data objects. Removing the centralized blackboard controller removes a major serial bottleneck in the blackboard system and results in increased system performance.

Specialized shared memory processors that can support the blackboard data structure are provided by several hardware vendors. Unfortunately I did not have such a machine available for my research. Since a shared memory machine was not available to test the COBS system, the blackboard handlers were multi-tasked on a single LISP machine with direct access to the object-oriented blackboard. Simulation models of the system executing on a shared memory system and on the LISP machine were built and used to predict the performance improvement that could be achieved if a shared memory machine was available.

An n-readers/one-writer protocol is used by the knowledge sources and blackboard handlers to guarantee blackboard consistency. All blackboard handlers use a two-phase locking protocol to guarantee knowledge source serializability. Knowledge sources that communicate directly with "child" processes use read/write synchronization to guarantee internal serializability. Direct operating system support is used to implement the required data object locking and queuing mechanisms. The operating system provides the software constructs required to define atomic operations and implement atomic blackboard read/write operations for the blackboard handlers. Primary deadlock detection is provided by the operating system.

C-2

6.1.1 The Object–Oriented Blackboard Data Structure

The blackboard supports a user defined set of blackboard data object types, and each data object type has a corresponding blackboard daemon type. The blackboard data structure consists of a dynamic set of blackboard data objects. The blackboard is a hierarchically partitioned, object-oriented data structure. Each blackboard data element has a static data object type, a dynamic value, and a dynamic activation identifier.

Each data element in the blackboard has a daemon attached to it. Each daemon has a fixed daemon type, a notification function, and a static list of knowledge sources to notify when the data object is updated. These daemons are used to signal the blackboard handlers when a blackboard data object has been updated. When a blackboard data object is updated, the associated daemon is activated. The daemon sends a message to each blackboard handler that monitors the blackboard data object. This message causes the input conditional associated with the blackboard data object to be updated to TRUE and all knowledge source preconditions that use the blackboard data object as input to be evaluated.

6.1.2 Blackboard Handlers

COBS implements a unique blackboard handler for each type of knowledge source. The blackboard handler acts as an interface between the knowledge source and the blackboard. A blackboard handler consists of a static input set describing the specific data objects or types of objects used as input by the blackboard handler's knowledge source, a static set of input conditionals, a static set describing the data objects or types of data objects output by the blackboard handler's knowledge source, a dynamic set of preconditions, a dynamic set of postconditions, and a queue used to store input variables. The following constraints are applied to the blackboard handlers:

1) All access of the blackboard is done using a two-phase locking protocol.

2) A n-readers/one-writer protocol is used to allow concurrent read operations.

The blackboard handlers reside on a single processor with direct memory access to the blackboard, in effect creating a shared blackboard data object similar to the blackboard implementation used in a shared memory blackboard system. When activated, the blackboard handler performs an atomic read operation to create the message to be sent to it's knowledge source. At knowledge source completion, the

handler tests the knowledge source's postconditions and if they hold, performs an atomic write operation to update the blackboard data objects.

Since this system provides a blackboard handler for each knowledge source instead of a blackboard server for a specific region on the blackboard, blackboard handler overloading should not occur. By separating the blackboard handlers from the knowledge sources, the cost of using atomic read/write operations has been reduced. The blackboard handlers are halted by the atomic operations, but the knowledge sources that are executing on the distributed network continue execution.

6.1.3 Knowledge Sources

COBS is designed to use a set of highly specialized, highly independent knowledge sources. Each knowledge source consists of a set of inputs, a set of outputs, a set of input conditionals, a set of preconditions, a set of postconditions, and the function performed by the knowledge source. Each knowledge source has a blackboard handler associated with it to handle all blackboard input and output operations. The knowledge source's input conditionals, preconditions and post conditions are handled by the knowledge source's blackboard handler. Knowledge sources execute as uninterruptable processes, they cannot be interrupted once they have been activated by the blackboard handlers.

The use of atomic read/write operations and the use of local knowledge source memory removes the requirement of locking blackboard objects or regions during knowledge source execution. Since all of the knowledge sources run as uninterruptable processes on distributed hardware, no context switching overhead is incurred.

Knowledge sources can dynamically create new generic blackboard data objects. The new generic data objects must be of a predefined blackboard data object type. The knowledge source creates the blackboard data object, initializes the object, and places the object on the blackboard. The data object inherits the proper type of daemon for its data object type, and all knowledge sources that use this type of generic data object can access the data object.. The creation of a new data object is an atomic operation.

A knowledge source's input conditionals are used to determine when a knowledge source's input data objects or input data types have been updated. The knowledge source's preconditions are used to place restrictions on when a knowledge source can be activated. A knowledge source cannot be activated until all of the knowledge

source's input conditionals are TRUE, and all of the knowledge source's preconditions are TRUE.

The use of postconditions allows a knowledge source to make sure the results of its computation are still valid, or that they are still required. The postconditions can also be used to keep conflicting results from being posted on the blackboard. For example, a knowledge source designed to determine the current tactical situation in a one-on-one air combat engagement posts on the blackboard that the current mode is evasive. This mode update causes the defensive systems knowledge source to activate. Before the defensive systems knowledge source completes execution, the situation assessment module receives new information and updates the mode to aggressive, activating the offensive systems knowledge source. When the defensive systems knowledge source is ready to post its results, it sees the mode has been updated to aggressive and terminates execution without updating the blackboard.

6.1.4 Knowledge Source Activation

The activation of a knowledge source is an atomic operation. The blackboard handlers for each type of knowledge source monitor the status of the input conditionals and preconditions for that type of knowledge source. When all of the input conditionals and preconditions are TRUE the knowledge source is activated. The following steps occur during a knowledge source activation:

1) Upon activation, the blackboard handler for a knowledge source performs an atomic read operation to build a local copy of all data required from the blackboard.

2) The blackboard handler resets all of the knowledge source's input conditionals to FALSE.

3) If the knowledge source is waiting, the blackboard handler constructs a data packet of the local data and sends the packet to the knowledge source. If the knowledge source is executing, the blackboard handler queues the data packet and waits until the knowledge source completes execution before sending the packet to the knowledge source.

6.1.5 Knowledge Source Execution

The execution of a knowledge source is an atomic operation. The knowledge source executes its computation, constructs a data packet containing its outputs, and sends the packet to the blackboard handler.

6.1.6 Knowledge Source Completion

The completion of a knowledge source is an atomic operation. Upon knowledge source completion, the blackboard handler for a knowledge source tests the knowledge source's postcondition. If the postconditions holds, the blackboard handler updates the blackboard data objects with the outputs of the knowledge source. If the postcondition fails the blackboard handler discards the knowledge source's outputs. If the handler has a data packet queued for the knowledge source, it is sent to the knowledge source.

6.2 Description of The COBS Design and Analysis Tools

The COBS design and analysis tools implement the techniques described in Chapter Four. The experience that I have gained developing Concurrent CLAWS, the Mode Control Panel[6] program, and the original Tactical Decision Generator systems [7, 8, 9] greatly influenced the design of the Knowledge Source Connectivity Analyzer. Using the design and analysis tools a concurrent object-oriented blackboard system that is faster, more efficient, and more powerful than existing systems can be developed. The use of the design and analysis tools provide the highly specialized, highly independent knowledge sources required for my concurrent blackboard systems to achieve their design goals. The Knowledge Source Connectivity Analyzer was developed using Common LISP to run on a Symbolics™ workstation. A listing of the Knowledge Source Connectivity Analyzer software is included in Appendix C.

6.2.1 Features of the Knowledge Source Analyzer

The Knowledge Source Connectivity Analyzer implements all of the features discussed in Chapter Four and produces a Blackboard System Specification File and a file containing the results of the connectivity analysis. The Blackboard System Specification File contains all of the information required to analyze the blackboard system design, the inputs required for the COBS Simulation System, and the inputs required for the Blackboard System Code Generator.

The knowledge source analyzer supports the following functions:

1) Add a Knowledge Source. This function adds a knowledge source to the blackboard system specification. The user is prompted for the type of the knowledge source, the list of input variables and the list of output variables. The system checks each input and output variable. If the variable has not been defined, the system prompts the user for the type and initial value and defines the blackboard data object. The knowledge source connectivity graph and adjacency lists are updated.

2) Clear the Blackboard Specification. This function clears the specification and resets the analyzer to its initial state.

3) Compute Circuits. This function uses the connectivity graph and an implementation of Tarjan's cycle detection algorithm to detect all feedback loops in the design.

4) Delete A Knowledge Source. This command prompts the user for the name of a knowledge source and deletes it from the current system specification. The knowledge source connectivity graph and adjacency lists are updated.

5) Edit a Blackboard Data Object. This command prompts the use for the name of a blackboard data object and then allows the user to modify the name, type and initial value of the data object.

6) Edit a Knowledge Source. This command prompts the use for the name of a knowledge source and then allows the user to modify the current attributes of the knowledge source. The user can modify the input and output variables of the knowledge source, change the knowledge source's preconditions and postconditions, and set the knowledge source's execution delay and update rate. The knowledge source connectivity graph and adjacency lists are updated.

7) Generate a Blackboard System Specification File. This command prompts the user for a file name to store the current blackboard

system specification. The list of knowledge sources and blackboard data objects is written to the file. A sample blackboard system specification file is included in Appendix C.

8) Generate a Blackboard System Analysis Output File. This command prompts the user for a file name to write the analysis of the current blackboard system specification. The list of knowledge sources and blackboard data objects is written to the file. The values of Λ, Ψ, Ω, Π and Σ for each knowledge source are also written to the file. A sample blackboard system output analysis file is included in Appendix C.

9) Load a Blackboard System Specification File. This command prompts the user for a blackboard system specification file name and loads the specification file. This overwrites the current state of the connectivity analyzer.

10) Quit. This command exits the connectivity analyzer. The current state of the connectivity analyzer is not saved.

11) Show Adjacency Lists. This command displays the adjacency list for each knowledge source in the system specification.

12) View a Blackboard Data Object. This command prompts the user for the name of a blackboard data object and displays the current values of the attributes of the selected data object.

13) View Knowledge Source. This command prompts the user for the name of a knowledge source and displays the current values of the attributes of the selected knowledge source.

14) View Λ. This command displays the output to input connectivity set for each pair of knowledge sources in the system specification.

15) View Γ. This command displays the output overlap set for each pair of knowledge sources in the system specification.

16) View Ω. This command displays the specialization values for each pair of knowledge sources in the system specification.

17) View Π. This command displays the interdependence values for each pair of knowledge sources in the system specification.

18) View Σ. This command displays the serialization values for each pair of knowledge sources in the system specification.

The results of the analysis of blackboard system specification B_1 and its blackboard system specification file are included at the end of Appendix C

6.3 Description of The COBS Simulation Model

A Blackboard System Simulation Model and a set of blackboard evaluation metrics have been developed to analyze the performance of blackboard systems. Verification and was performed using example blackboard system design specifications. The verification process will be presented later in the chapter The simulation model was refined using the example problems until the simulation system generated the proper simulation software and results. A copy of the simulation code generated for the example blackboard systems, and the results of the simulation runs are included in Appendix D. The blackboard simulation model was validated against the performance of several types of blackboard systems executing on a Intel ISPC 16 processor HyperCube™, a Symbolics 3650 workstation™, and a network of VAX™ and Symbolics workstations.

The COBS blackboard system simulation model is implemented according to the formal Blackboard System Simulation Model. The COBS Blackboard System Simulation Model was verified by comparing the software generated by the simulation system to the blackboard system specifications of several example systems. During initial system verification, discrepancies between a test system's specification and the generated software were used to refine the blackboard simulation system. The refinement process was iterated until the specifications and the software generated by the system converged.

The COBS Blackboard System Simulation Model was validated by comparing the performance predicted by the simulation model to the actual performance of several existing systems. Discrepancies between a system's predicted and actual performance were used to refine the blackboard simulation model, or if required, the blackboard system implementation. The refinement process was iterated until the predicted and actual performance values converged.

6.3.1 COBS Blackboard System Simulation System

The COBS Blackboard System Simulation System is a tool for generating simulations of blackboard systems. The simulations consist of three types of elements: the blackboard data structure, a set of knowledge source modules, and a ordered knowledge source event queue. The blackboard data structure contains the initial values of the blackboard data objects, the blackboard handlers and data object locks, and the global clock.

The simulation system uses a COBS Blackboard System Specification File as input. Based on the specification file, the system generates all software required for the simulation,except for the models of the knowledge source functionality. This includes: defining all required variables; defining all required flavors and methods for blackboard data objects; defining all of the blackboard handlers and the supporting functions for handling input conditionals, preconditions, and postconditions; defining all required blackboard data object locks; building and initializing all blackboard data objects; generating all code required by the blackboard system evaluation metrics. The code generated by the simulation system is, except for the code required for the evaluation metrics, identical to the code generated by the Blackboard System Code Generator. The simulation model uses the same blackboard data objects and locking procedures as the final version of the code, increasing the fidelity of the simulation model and increasing the reliability of its results.

The Blackboard System Simulation System supports the following functions:

1) Clear the Blackboard Specification. This function clears the specification and resets the analyzer to its initial state.

2) Generate Blackboard Simulation System. This function generates the software required for the blackboard simulation system.

3) Load a Blackboard System Specification File. This command prompts the user for a blackboard system specification file name and loads the specification file. This overwrites the current state of the connectivity analyzer.

4) Quit. This command exits the connectivity analyzer. The current state of the connectivity analyzer is not saved.

5) View a Blackboard Data Object. This command prompts the user for the name of a blackboard data object and displays the current values of the attributes of the data object.

6) View Knowledge Source. This command prompts the user for the name of a knowledge source and displays the current values of the attributes of the knowledge source.

The COBS Blackboard System Simulator is implemented in Common Lisp on a Symbolics™ workstation according to the simulation model for blackboard systems described in Chapter Five. A listing of the Lisp code for the Blackboard System Simulator is included in Appendix D.

6.3.2 Blackboard System Evaluation Metrics

Since no metrics for the evaluation of blackboard systems exist in the literature, I have developed several that are helpful in analyzing blackboard system performance. These metrics are automatically measured by the blackboard system simulation model.

1) The number of times a knowledge source is activated.

2) The number of shared data objects on the blackboard and the number of knowledge sources that access each data object.

3) Knowledge source output to input connectivity.

4) Execution time of the blackboard system.

5) Number of times data objects are locked and unlocked.

6) Number of times a process had to wait for locked data objects

The metrics measure knowledge source connectivity, knowledge source execution time, the number of shared data objects, data object locking, and other relevant performance data. These metrics are used to evaluate the system's measured performance against the performance predicted by the blackboard system simulation model of the system.

6.4 Description of The COBS Code Generator

The COBS Blackboard System Code Generator is a tool for generating all of the low level support software for concurrent blackboard systems. The code generator uses a COBS Blackboard System Specification File as input. Based on the specification file the system generates all software required for blackboard system, excluding the code for the knowledge sources. This includes: defining all required variables; defining all required flavors and methods for blackboard data objects; defining all of the blackboard handlers and the supporting functions for handling input conditionals, preconditions, and postconditions; defining all required blackboard data object locks, and building and initializing all blackboard data objects. The user provides the code for the knowledge sources. The code generator sets up a generic interface between the blackboard handlers and the knowledge sources and provides the user with a communication specification for each knowledge source.

The blackboard system code generator supports the following functions:

1) Clear the Blackboard Specification. This function clears the specification and resets the analyzer to its initial state.

2) Generate Blackboard System Code. This function generates the software required for the blackboard system.

3) Load a Blackboard System Specification File. This command prompts the user for a blackboard system specification file name and loads the specification file. This overwrites the current state of the connectivity analyzer.

4) Quit. This command exits the connectivity analyzer. The current state of the connectivity analyzer is not saved.

5) View a Blackboard Data Object. This command prompts the user for the name of a blackboard data object and displays the current values of the attributes of the data object.

6) View Knowledge Source. This command prompts the user for the name of a knowledge source and displays the current values of the attributes of the knowledge source.

The COBS Blackboard System Code Generator is implemented in Common Lisp on a Symbolics™ workstation . A listing of the Lisp code for the code generator and a copy of the code generated for blackboard system B_2 is included in Appendix E.

The COBS Blackboard System Code Generator currently produces software in Common Lisp using the Common Lisp Object System and the Common Lisp Interface Manager. The software produced is portable to any computer that supports the Common Lisp Standard. The Daemon-driven control structure has been tested a Symbolics 3560 ™ workstation and a Symbolics MacIvory™ workstation. On the Symbolics workstations the daemons execute serially, using operating system support to guarantee atomic operation. This is not a function of the code generator, but of the target architecture. The daemon specific code is concurrent by design and would require no modifications to execute concurrently. The concurrent execution of the daemons was simulated using the COBS simulation system, and a slight performance increase was detected. There are plans to test the daemon-driven control structure on an associative memory processor in the future. The proposed associative memory architecture will allow concurrent daemon execution. The move to a new target architecture may require minor changes to the code generator.

Endnotes for Chapter Six

1 Corkill, 1989.

2 Lesser and Corkill, 1983

3 Nii, Aiello, and Rice, 1988

4 Corkill, "Design Alternatives...", 1989

5 Jagannathan, 1989

6 Hueschen and McManus, 1988.

7 McManus and Goodrich, 1989

8 McManus and Goodrich, 1990

9 McManus, 1990

Chapter 7

The Paladin Tactical Decision Generation System

Testing the COBS system required a target application that was challenging and could provide existing performance data. Paladin, a Tactical Decision Generator for real-time air-to-air combat was selected. The Paladin system is an ongoing research project, and the system's designs and performance characteristics were used to evaluate the COBS system. Paladin provided a challenging set of design constraints and tested all of the Features of COBS. Two approaches to developing Paladin are presented in this chapter. The first approach implements Paladin using a heterogeneous computer network, implementing each knowledge source on the class of processor but suited to it. The second approach implemented a subset of Paladin on a 16 processor iPSC/2 HyperCube™.

Paladin is a real–time tactical decision generator for air combat engagements. Paladin uses specialized knowledge–based systems and other AI programming techniques to address the modern air combat environment and agile aircraft in a clear and concise manner. Paladin is designed to provide insight into both the tactical benefits and the costs of enhanced agility. The system was developed using the Lisp programming language on a specialized AI workstation. Paladin utilizes a set of air combat rules, an active throttle controller, and a situation assessment module that have been implemented as a set of highly specialized knowledge–based systems. The situation assessment module was developed to determine the tactical mode of operation (aggressive, defensive, neutral, evasive, or disengagement) used by Paladin at each decision point in the air combat engagement. Paladin uses situationally dependent modes of operation to more accurately represent the complex decision-making process of human pilots. This allows Paladin to adapt its tactics to the current situation and improves system performance.

7.1 The Tactical Guidance Research and Evaluation System

Modern air combat simulations require an intelligent system, called a Tactical Decision Generator (TDG), to select the combat maneuvers to perform throughout an air combat engagement. The simulation system must also have the ability to model agile aircraft. The system should have a modular software structure so that new weapons systems or aircraft subsystems (e.g., sensors or propulsion systems),

modifications to aircraft control systems, or changes to the aircraft configuration can be easily incorporated. In support of the study of superagile aircraft at NASA Langley Research Center (LaRC), a Tactical Guidance Research and Evaluation System (TiGRES) is being developed. The design and development of TiGRES as well as its relationship to other current air combat simulation systems is described in detail in references[1, 2, 3].

The TiGRES system is designed to allow researchers to develop and evaluate aircraft systems in a tactical environment. The three main components of TiGRES are a TDG, the Tactical Maneuver Simulator (TMS), and the Differential Maneuvering Simulator (DMS). Both the TMS and the DMS use a TDG as the automated intelligent opponent.

7.2 TiGRES Components

The TMS[4, 5] provides a high-fidelity batch air combat simulation environment for the development and testing of various guidance and control strategies. The researcher defines the initial conditions of the air combat engagement and the TMS then controls the trajectories and attitudes of the aircraft using simple trajectory commands, or through a tactical decision generation system. The main elements of the TMS are a high-fidelity, nonlinear six degree-of-freedom (d.o.f.) rigid-body aircraft dynamic model, including the control system; a TDG; and a user interface.

The DMS consists of two 40' diameter domes and one 20' diameter dome. The facility is intended for the real-time simulation of air combat engagements between piloted aircraft. By using a TDG to control one of the airplanes, it is possible to test a TDG against a human opponent. This feature allows the guidance logic to be evaluated against one or more unpredictable and adaptive human opponents.

7.3 The Paladin Software

Paladin is a knowledge-based TDG designed to provide insight into both the tactical benefits and the costs of superagility. The development of Paladin has been a multi-stage process using a baseline version of the Adaptive Maneuvering Logic[6] (AML) program as the starting point. Paladin uses the trial maneuver generation and evaluation concept outlined in the AML program[7] with several extensions. The original set of five to nine aircraft trial maneuvers used by AML has been replaced with four sets of positionally dependent trial maneuvers. Past research[8, 9] has shown that

the use of the positionally dependent sets of trial maneuvers improves overall system performance, allows Paladin to perform target acquisition and tracking more effectively, and improves Paladin's defensive and evasive maneuvering performance. Paladin uses an object-oriented programming approach to represent each aircraft in the simulation. Each aircraft object includes information on the current state of the aircraft's offensive systems (e.g., guns, missile systems, fire control radars, etc.), defensive systems (e.g., electronic counter–measures, chaff, etc.), and propulsion system. This state information is used to help guide Paladin's reasoning process.

Paladin utilizes modular software subroutines and specialized computer hardware. The separation of the aircraft simulation and decision logic components, and the use of highly specialized knowledge sources, allows each module or knowledge source to be designed and implemented using the hardware and programming techniques specifically suited for its function. The use of highly specialized and independent knowledge sources also provides for modular protection, confining the effect of an error occurring in a module at run-time to that module, or to a small set of neighboring modules in the program. The confining effect of the modular protection was used to aid in the design and debugging process. Each knowledge source was developed and tested independently before it was incorporated into Paladin.

The independence of the knowledge sources also increases the efficiency of Paladin by allowing knowledge sources to be distributed across a network of several heterogeneous processors. The network currently consists of a Symbolics 3650™ workstation, a Symbolics MacIvory™ workstation, and four Vax 3200™ class workstations. Communication between the distributed knowledge sources is achieved using customized DECNet–based Client/Server software developed in–house for TiGRES. This software allows for synchronization, communications, and data sharing between heterogeneous computers running the DECNet™ communications protocol. Each knowledge source requests all of the data required to perform its computation from the blackboard at the start of its execution cycle, and posts its results to the blackboard at the end of its execution cycle.

7.3.1 The Paladin Blackboard Control Structure.

The current implementation of Paladin uses a prototype version of the daemon-driven control structure. This control structure will be replaced with the

COBS daemon-driven control structure in later versions of Paladin. The move to the Final COBS system should not affect system performance.

7.3.2 The Paladin Inference Engine.

The Paladin knowledge sources use a custom inference engine that was designed to support real-time execution of knowledge–based systems. The inference engine uses a depth–first evaluation strategy to search the active rule–bases. Rule–bases can be partitioned, and the partitions are linked using meta–rules (rules used to guide the activation of rule–bases). Rule–bases are expressed in two formats: interpreted lists of condition action pairs used during the design stage, and compiled lists of in-line function definitions used in the final, real–time version of the system. The interpreted lists are used to develop and debug the initial versions of the rule–base. Most existing rule-based systems implement interpreted rule–bases. The use of the interpreted rules severely limits the execution performance of the inference engine, and restricts the real-time application of this type of system. To overcome this problem and allow real–time execution, the rule–base is "compiled" into a list of in–line function definitions. The compiled rule–bases execute approximately 90 to 100 times faster than the interpreted rules. The inference engine executes a representative test rule–base consisting of 40 rules in the interpreted format in 170 milliseconds. The inference engine executes the same rule–base in the compiled format in 1.9 milliseconds.

There is a trade–off between the length of the longest execution path in a rule-base and the execution time of a knowledge source. The shorter the execution path is, the faster the execution time. The rule–bases used by Paladin have been partitioned to increase system performance by grouping related rules into small partitions and using meta–rules to link the partitions. This partitioning decreases the number of rules that are active, and decreases the length of the worst–case execution path through the rule–base. The rule–base partitioning allows the designer to calculate the longest and shortest path through the rule–base and compute both a maximum and minimum knowledge source execution time. The knowledge source's maximum execution time can be used to insure that the system will meet real–time execution requirements. If the maximum execution time exceeds the allocated execution time, the designer may be able to repartition the rule–base until real–time execution requirements are achieved.

7.3.3 The Paladin Rule–Bases.

Paladin uses two rule–bases: a mode selection rule–base used by the Situation Assessment knowledge source, and a throttle control rule–base used by the Active Throttle Controller. The mode selection and throttle control rule–bases are discussed in detail in[10]. The mode selection rule–base consists of four partitions and contains nineteen rules. The shortest execution path in this rule–base results in a single rule being fired; the longest path results in twelve rules being fired. The throttle control rule–base consists of ten partitions and contains forty rules. The shortest execution path in this rule–base results in a two rules being fired; the longest path results in thirteen rules being fired.

7.4 KBS Modules of Paladin

The development of the TDG has been a multi-stage process. The current version of the TDG, Paladin, is a blackboard-based system written in LISP and FORTRAN that uses an object-oriented programming approach to represent the aircraft states and subsystems. Paladin uses objects to represent each aircraft in the simulation and the current state of the aircraft's offensive systems, defensive systems, and engines. This information is used by Paladin's knowledge sources to help guide the reasoning process.

7.4.1 Situation Assessment Module

Six modes of operation, shown in Table 7.1, have been incorporated in Paladin. As shown in Figure 7.1, the Situation Assessment knowledge source is executed before the maneuver scoring module. The Situation Assessment knowledge source uses the mode selection rule–base to determine the system's current mode of operation. This knowledge source is used to model a pilot's situational awareness and changing problem–solving strategies. Just as a pilot will recognize the difference between an aggressive situation and a evasive situation and react accordingly, the Situation Assessment knowledge source provides information allowing Paladin to adapt its problem–solving strategy based on the current situation. The determination of the current mode of operation is based on the aircraft's current mission, the current state of the aircraft's systems, the relative geometry between the aircraft and its opponent, and the opponent's instantaneous-intent (defined later). Each of the six modes of operations has a unique vector of scoring weights and a unique decision interval (shown in Table 7.1). The scoring weights for each mode of operation have been adjusted during the design and testing process to maximize Paladin's performance in that mode of

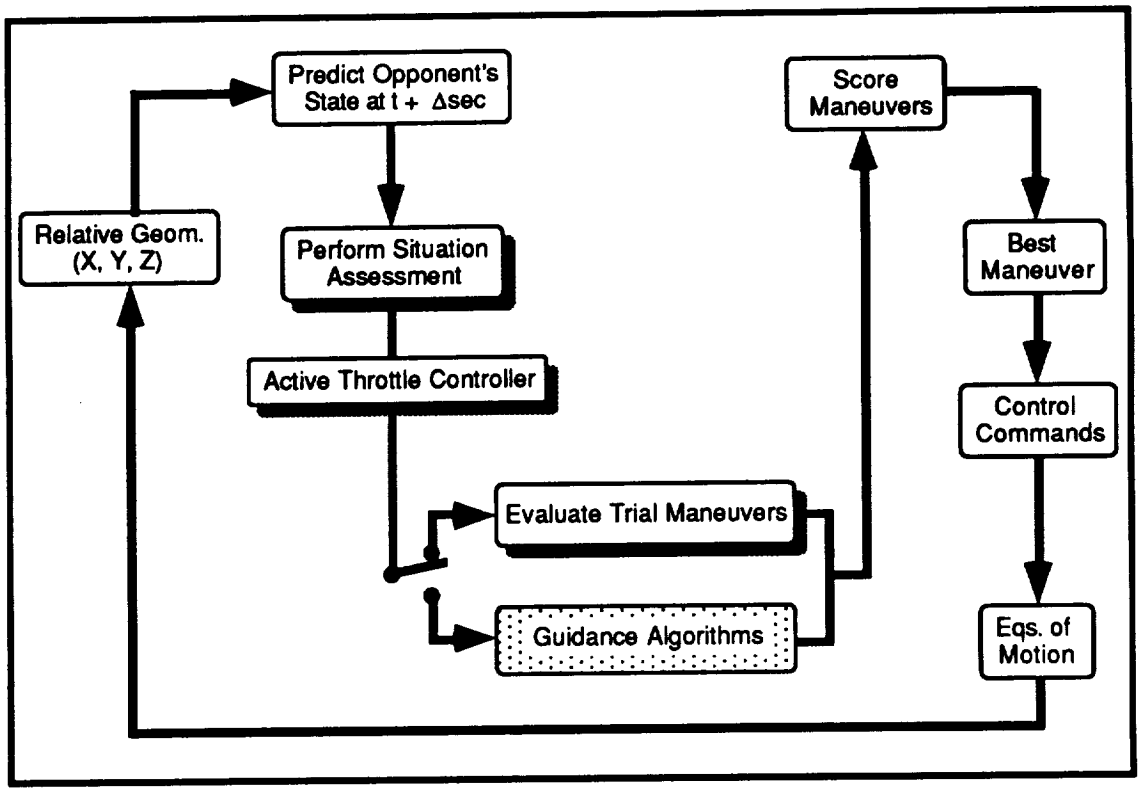

Figure 7.1. Schematic Of Paladin

operation. The testing procedures used to evaluate Paladin's performance are described in detail later in this chapter.

Table 7.1. Modes of Operation

Mode	Decision Interval
Aggressive	0.25 sec
Defensive	0.5 sec
Evasive	0.25 sec
Ground Avoidance	0.125 sec
Neutral	1.0 sec
Disengage	0.5 sec

The situation assessment knowledge source also determines the opponents instantaneous-intent. The opponent's instantaneous-intent is defined to be an estimation of the opponent's mode of operation at the current point in time based on Paladin's available sensor, positional, and geometric data. Currently, there is no attempt to use a history of instantaneous-intent to derive a long-term opponent intent.

7.4.2 Situation Assessment Data

To perform situation assessment, information on aircraft relative geometry and Paladin's system status is required. This information is available in the form of participant-specific data maintained by Paladin. All data relating to the Paladin aircraft as well as Paladin sensor data (e.g. the opponent's x, y, z position) are assumed to be known exactly. Other data required about the opponent must be estimated.

Paladin's current throttle position and altitude are parameters taken directly from the current state. Range is the magnitude of a vector connecting the centers of gravity of the aircraft. The Line-Of-Sight (LOS) angle is defined as the angle between the LOS vector and the ownship body x-axis (Figure 7.2); the deviation angle is defined as the angle between the LOS vector and the ownship velocity vector; and the LOS angle off is defined as the angle between the LOS vector and the opponent's body x-axis. The data used by the situation assessment module is shown in figure 7.2, and discussed in detail in [11]

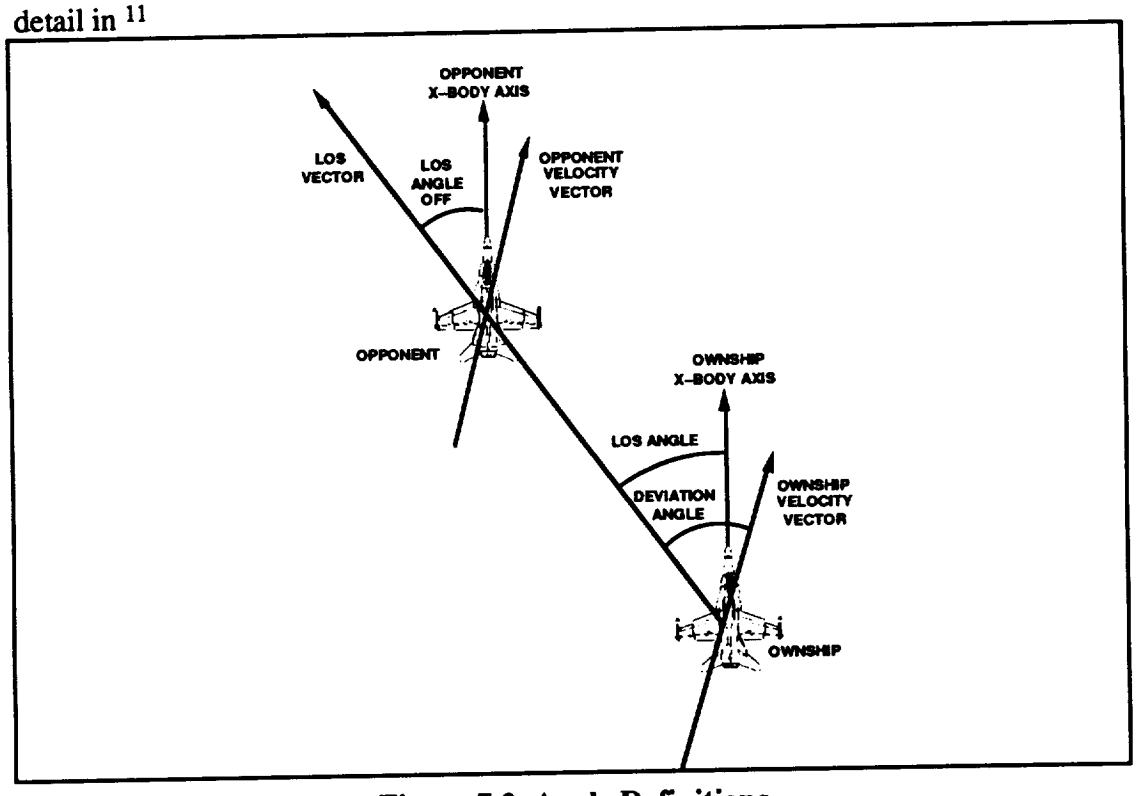

Figure 7.2. Angle Definitions

7.4.3 Active Throttle Controller

A rule-based Active Throttle Controller was developed to adjust the throttle setting based on the current mode of operation. The throttle controller is called at the start of

each decision interval and can set the throttle to any position between idle and full afterburner [0.0 = flight idle, ..1.0 = military power, ..2.0 = full afterburner]. The throttle controller uses the throttle control rule–base, the current mode of operation, and the relative geometry information to select either a target acquisition mode, a fine tracking mode, or a target or missile avoidance mode. Each mode has a set of specific throttle control rules that are used to maximize system performance in that mode.

7.4.4 Scoring Module

The Paladin Maneuver Scoring Module knowledge source is a FORTRAN subroutine. The scoring module uses a set of fuzzy logic questions[12] with responses ranging from [-1.0 = Negative, ..0.0 = Neutral, ..1.0 = Positive] and the mode-specific scoring weight vector selected by the situation assessment module to score each of the trial maneuvers. For each trial maneuver evaluated the predicted positions for both the opponent and the Paladin aircraft are computed. The position of the opponent is extrapolated using a quadratic curve fit based on the time history of the opponent aircraft's trajectory. The future position of the Paladin aircraft is determined by predicting the result of executing the control commands for each candidate trial maneuver.

Once the relative geometry between the future positions of the two aircraft is calculated, the score for the maneuver is determined by computing the responses to the fuzzy logic questions, applying the selected scoring weight vector, and then summing the results to generate a single numeric score. After all of the trial maneuvers have been evaluated, the highest scoring maneuver is selected and the associated control commands are executed.

7.5 Paladin Testing Procedures

Paladin is currently being tested in the TMS and the DMS.

7.5.1 TMS Testing

Paladin is currently being tested in the TMS using six d.o.f. aircraft dynamics. TMS testing is done in a non-real-time, batch mode environment against a baseline TDG. Each group of test conditions consists of 32 sets of initial aircraft conditions. The initial altitudes, airspeeds, and the separations between the two aircraft are adjusted to provide representative coverage of the within–visual–range air combat arena. The

largest initial aircraft separation currently being tested (5 nm) places the aircraft at the transition point between beyond-visual-range and within-visual-range air combat.

Four scoring metrics are currently used to evaluate each engagement. All metrics are computed at the aircraft simulation update rate of 32 times per second. The first metric computes the total time that each airplane has its weapons locked on the opponent, the probability that any weapons fired will hit the opponent, the distance between the opponents, the angle-off, and the deviation angle. The results are printed in a table format at the completion of each run. The scoring metrics are described in detail in [13].

These statistics are reviewed after each set of test runs and the data are used to tune the mode specific scoring weights and test the completeness of the knowledge bases. Although the statistics are helpful, no single statistic has been developed that can accurately measure the performance of an aircraft in the engagement. All engagements are ended when the PS for either aircraft is less than 0.30.

7.5.2 DMS Testing

A baseline version of Paladin is currently being tested in the DMS using a 5 d.o.f. aircraft model. The aircraft model lacks both the extra degree of freedom (lateral motion in body axes) as well as an accurate representation of the aircraft's rotational dynamics throughout the complete flight envelope. The baseline Paladin system, the Computerized Logic for Air Warfare Simulation (CLAWS) contains the situation assessment module, the active throttle controller, and a set of situationally dependent trial maneuvers. The simplified aircraft model is used to insure real-time performance in the DMS.

The development of CLAWS has made it possible to evaluate the tactical decision generation software against human pilots in the DMS in a realistic air combat environment. This capability has allowed experienced pilots to interact with the system and comment on its performance and suggest improvements. The pilots' comments and suggestions are the basis for changing the TMS experimental version of Paladin. These changes are tested and refined in the TMS before being included in the baseline system.

7.6 Concurrent Paladin

A concurrent version of the Paladin software, Cube_CLAWS, was developed as a distributed blackboard system in C[14,15]. The use of parallel distributed processing techniques allows the development of larger and more complete simulations than is currently possible using serial hardware and programming techniques. The independence of the Cube_CLAWS knowledge sources increases the efficiency of the system by allowing knowledge sources to be distributed across the hypercube. The knowledge sources communicate with the blackboard using a daemon-driven message passing system.

Cube_CLAWS is a distributed blackboard system designed to execute on a 16 processor iPSC/2 HyperCube™. Cube_CLAWS consists of a set of knowledge sources for each aircraft in the simulation. The set includes: a main knowledge source that contains the blackboard support software and aircraft model; a relative geometry knowledge source; a situation assessment knowledge source; and a maneuver evaluation knowledge source used to evaluate a set of eight prospective aircraft maneuvers. The daemon-driven control structure used for activating knowledge sources is embedded in the knowledge sources. The blackboard elements are passed as messages between the distributed knowledge sources and the blackboard data structure. Read/write synchronization is used to ensure blackboard consistency.

7.6.1 Software Description

The Cube_CLAWS software consists of six basic elements[16]. The host program initiates the blackboard system by passing the initial aircraft states to the two instances of the main knowledge source. The host program then goes into a loop to receive messages containing the updated aircraft states and writes them to the console.

The main knowledge source contains the aircraft modeling software and the required blackboard control elements. The main knowledge source is the "controlling" module for both aircraft in the engagement. The main knowledge source receives the initial aircraft state for the aircraft it will be controlling and initializes the variables used for inter-knowledge source communication. The main knowledge source then swaps aircraft state information with each main knowledge source for the other aircraft so that at the start of each time step all main knowledge sources know the position and appropriate state variables for all other aircraft in the engagement.

The relative geometry knowledge source is activated to compute the relative geometry between the two aircraft and then the situation assessment knowledge source is activated to determine the aircraft's current mode of operation. The maneuver evaluation knowledge source (Eval) then evaluates prospective aircraft maneuvers and returns the control commands for the highest rated maneuver to the main knowledge source. The maneuver is executed and the next cycle of the engagement begins.

The relative geometry knowledge source uses the current state of the aggressor and target aircraft stored on the blackboard. The LOS angle between the aggressor and the target, the closing rate, the range, and the angle-off are computed. The weapons solutions are computed, and if a gun or missile lock is achieved, the weapons systems are activated and the weapons lock times are incremented. The updated blackboard values are returned to the main knowledge source at the completion of the knowledge source execution.

The situation assessment knowledge source uses both the relative geometry data and aggressor's aircraft state data to compute the current tactical situation and update the aggressor's mode of operation. A set of eight tactical evaluation metrics are used to define the situation space. The situation assessment knowledge source uses a fuzzy logic based scoring scheme to evaluate the metrics and map the current situation into one of the modes of operation shown in Table 7.1. The situation assessment knowledge source also computes the opponents instantaneous-intent. The mode of operation is used to select a set of scoring weights that are used to generate a numeric "score" for the current maneuver. The score of a maneuver represents the computed tactical worth of the position being evaluated. The updated blackboard data elements are returned to the main knowledge source at completion of the knowledge source execution.

The evaluation knowledge source is used to evaluate the tactical value of candidate maneuvers. The knowledge source uses the current state of the aggressor and target aircraft from the blackboard. The Eval routine first predicts the future position of the opponent based on his last known position, flight path angle, speed, and heading. Eval then generates a set of eight candidate maneuvers based on the current mode of operation and generates the new position for the aggressor aircraft. These new positions and the projected position of the opponent are placed in blackboard data elements and the relative geometry knowledge source is activated to compute the relative geometry between the candidate positions and the predicted position of the

opponent. The results are placed in blackboard elements and the situation assessment knowledge source is activated to compute the mode of operation for each of the maneuvers and a tactical score for each position. The situation assessment results are placed in blackboard data elements. Eval selects the highest scoring maneuver and the required control commands are placed on the blackboard to be used by the main knowledge source.

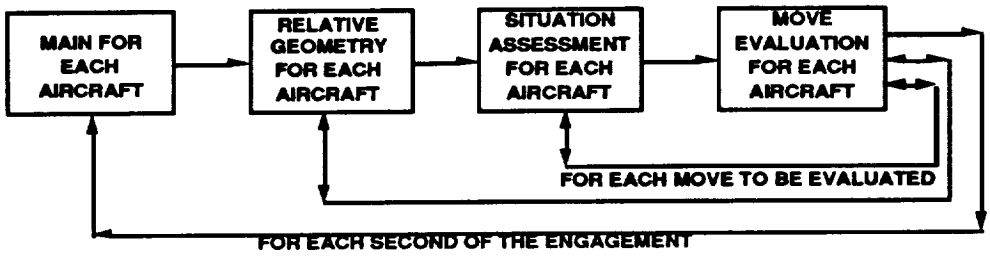

Figure 7.3. CLAWS Schematic

7.6.2 Cube_CLAWS Software Structure

Figure 7.3 shows a schematic of the "optimal serial" version of the Cube_CLAWS program. The optimal serial version is a highly optimized serial version of the CLAWS software. The main knowledge source is called for both aircraft, followed by a call to compute the relative geometry between the two aircraft and a call to perform the situation assessment. The move evaluation subroutine is then called and it calls an instance of the relative geometry and the situation assessment once for each prospective maneuver to be evaluated. This cycle is executed once for each second in the simulated engagement. Cube_CLAWS exploits the natural parallelism of the engagement by creating separate parallel execution paths for both aircraft in the engagement. The main knowledge sources of each of the aircraft synchronize at the start of each time step in the engagement to swap aircraft state data and then proceed down parallel execution paths.

The evaluation of trial maneuvers is also done in parallel. Multiple versions of the situation assessment and relative geometry knowledge sources are loaded onto processors of the HyperCube and are used to evaluate the candidate maneuvers. The Eval module generates the prospective maneuvers and then sends one maneuver to each available relative geometry knowledge source. When all of the maneuvers have been distributed and processed, the results are placed on the blackboard and the positions are distributed to the available situation assessment knowledge sources. The

positions are then evaluated and the set of control commands for the highest scoring maneuver is placed on the blackboard for the main knowledge source to execute.

It is important to note that although multiple versions of the relative geometry modules and the situation assessment modules are being executed in parallel, there is still an inherent serialization or pipelining between the relative geometry and the situation assessment modules. The relative geometry must be computed for a maneuver before the situation assessment module can begin execution. Figure 7.4 is a schematic of the current Cube_CLAWS software configuration.

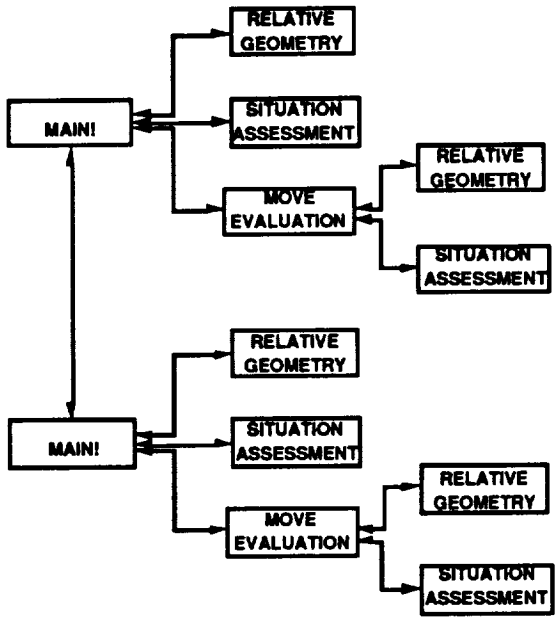

Figure 7.4. Cube_CLAWS Schematic

7.7 Evaluation of Cube_CLAWS Performance

A total of six test cases were used to evaluate the performance of Cube_CLAWS. The baseline case was the optimal serial version of Cube_CLAWS run on a single processor. The Cube_CLAWS software was then run in the configurations described in table 7.2, with the table entries representing the HyperCube processor executing the software. Configuration one (C1) loaded all of the processor software for both aircraft on a single processor. Configuration two (C2) loaded the software for aircraft one (A1) on processor 0 and the processor software for aircraft two (A2) on processor 1, creating parallel execution paths for the two aircraft. Configurations three through five (C3, C4, C5) loaded the processor software as described for configuration two, with

the addition of loading multiple versions of the situation assessment and relative geometry software.

Table 7.2. Software Test Configuration Processor Assignments

Test Version	Aircraft Identifier	Main Subroutine	Relative Geometry	Situation Assessment	Maneuver Evaluation
C1	A1	0	0	0	0
	A2	0	0	0	0
C2	A1	0	0	0	0
	A2	1	1	1	1
C3	A1	0	0,2,3	0,2,3	0
	A2	1	1,8,9	1,8,9	1
C4	A1	0	0,2,3,4,5	0,2,3,4,5	0
	A2	1	1,8,9,10,11	1,8,9,10,11	1
C5	A1	0	0,2,3,4,5,6,7,8,9	0,2,3,4,5,6,7,8,9	0
	A2	1	1,8,9,10,11,12,13,14,15	1,8,9,10,11,12,13,14,15	1

Speedup and efficiency computations were performed for the evaluation module and per second of the engagement. Speedup is a measure of how much faster a problem P is solved using N processors than when solved serially. The efficiency of a parallel algorithm is defined to be the speedup divided by the number of processors used.

Due to the size of the problem it was not possible to run the eight processor versions of Cube_CLAWS and evaluate all eight potential maneuvers without having some processors executing more than one task (processor overlap). In these cases the speedup without processor overlap was computed using the average execution speed of the nonoverlapping processors.

Figure 7.5 plots the expected speedup per second of simulated engagement (in the case of processor overlap) and the speedup actually achieved. Configuration one is used as the base case for these speedup calculations. These calculations measure the effect of splitting the task into separate execution paths for each aircraft. Note that the optimal serial implementation took approximately 23 percent less time to execute than configuration one. The speedup gained by splitting into separate paths for each aircraft is found by comparing the configuration one and the configuration two data. In this case the speedup achieved is almost linear, approximately 97 percent. There is a

distinct leveling off in speedup between configuration four and configuration five. Configuration five achieves much better speedup when no process overlap occurs.

Figure 7.6 shows that processor efficiency for the simulation decreases as additional processors are added. Much of the loss of speedup is due to the serial nature of the main knowledge source and the need for all main knowledge sources to synchronize at the start of each iteration. The additional processors used to compute the situation assessment and relative geometry are only assigned work by the maneuver evaluation knowledge source and do not speed up the execution of the aircraft models. The additional processors are idle except when evaluating trial maneuvers.

The computed speedups for the maneuver evaluation knowledge source (without processor overlap) are shown in figure 7.7. These calculations measure the effect of adding additional processors for the relative geometry and situation assessment knowledge sources on the execution of the Eval knowledge source. The speedup is close to linear for all cases except configuration five, the eight processor case. This data shows that adding additional processors has a large effect on the execution time of the maneuver evaluation subroutines.

The maneuver evaluation processor efficiency, shown in figure 7.8, is greater than 65 percent in all configurations tested. Configuration 5 shows significant improvement in both speedup and processor efficiency. The computed efficiency for configuration five is approximately 67.5 percent.

The sequential relationship between the situation assessment knowledge source and the relative geometry knowledge source reduces the benefit of adding additional processors in the current software configuration, and increases the number of messages that must be passed. The cost of passing these messages, both in time and operating system overhead, can be very large. The sequential relationship implies that the system may perform better if the two separate knowledge sources are combined to form a single knowledge source.

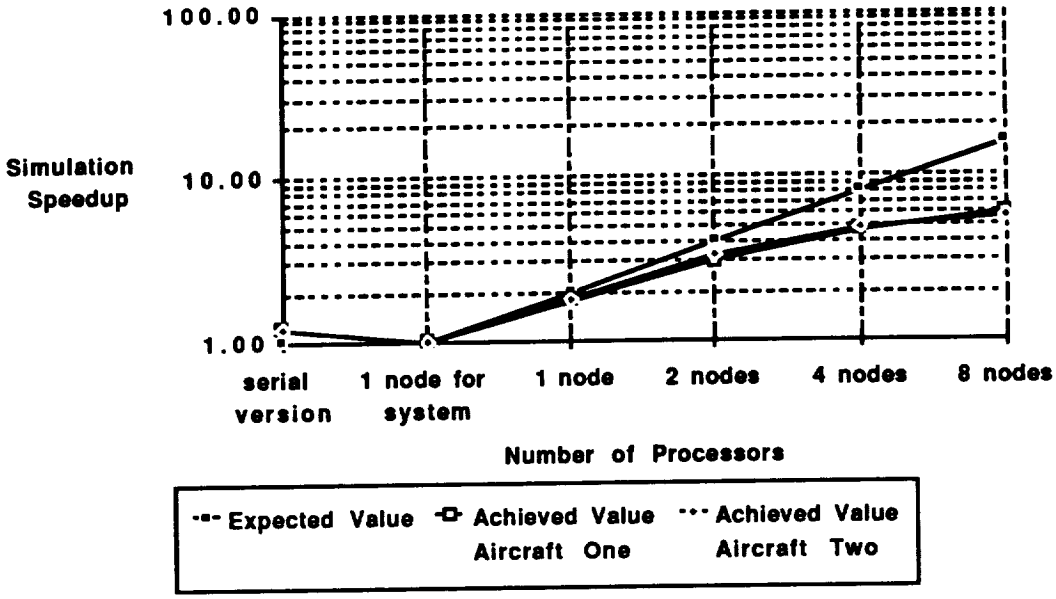

Figure 7.5. Simulation Speedup

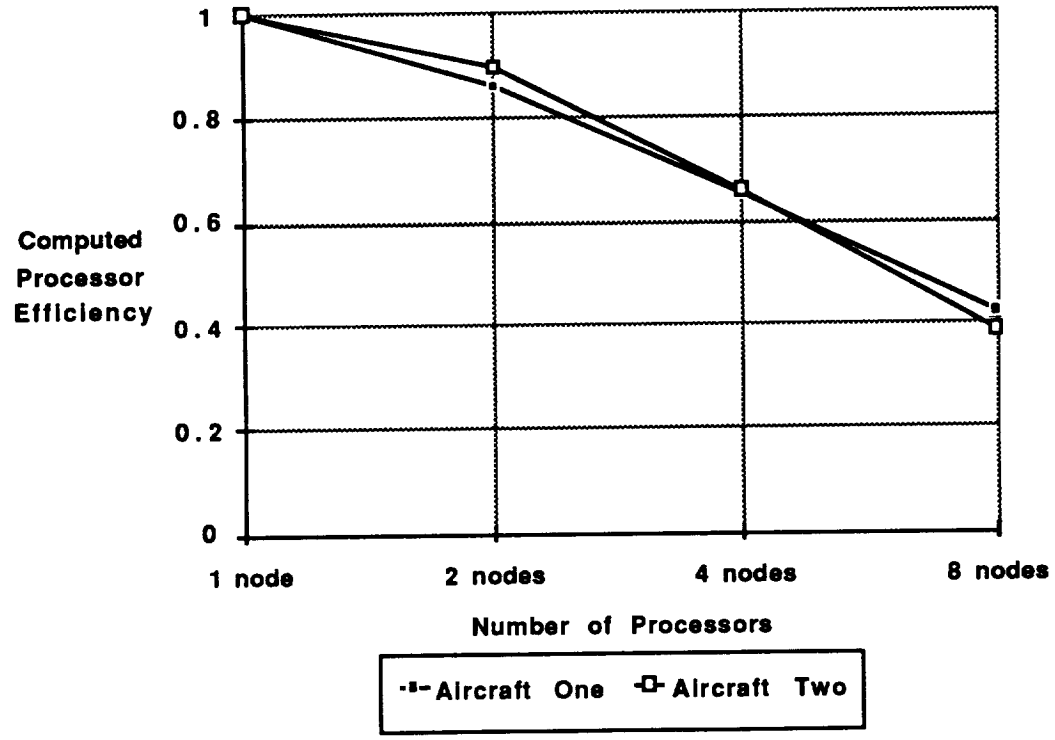

Figure 7.6. Processor Efficiency for Simulation.

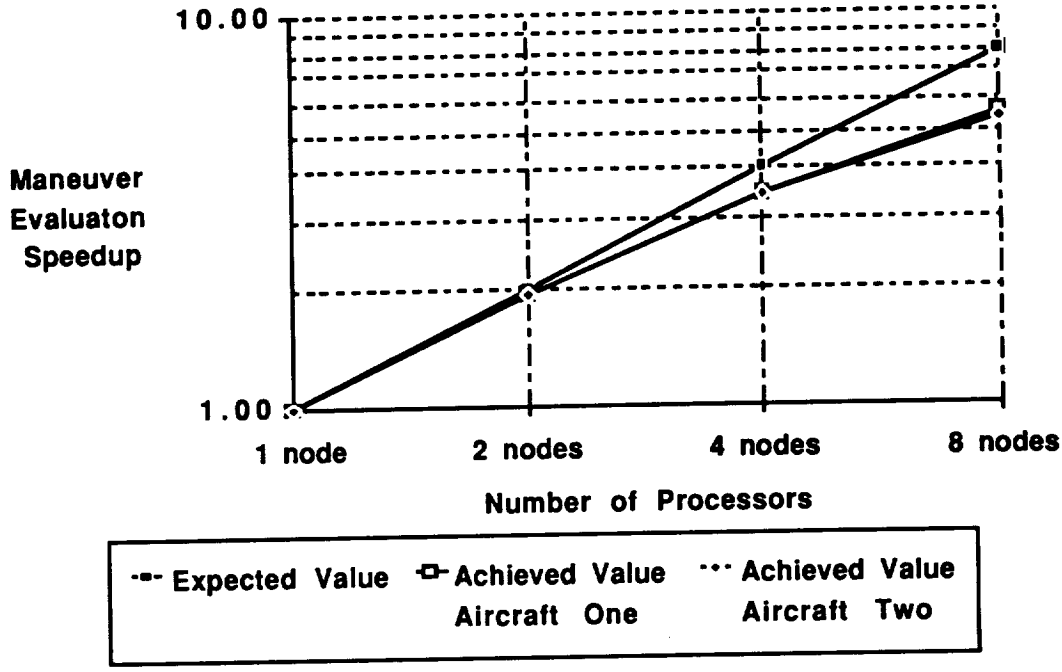

Figure 7.7. Maneuver Evaluation Speedup

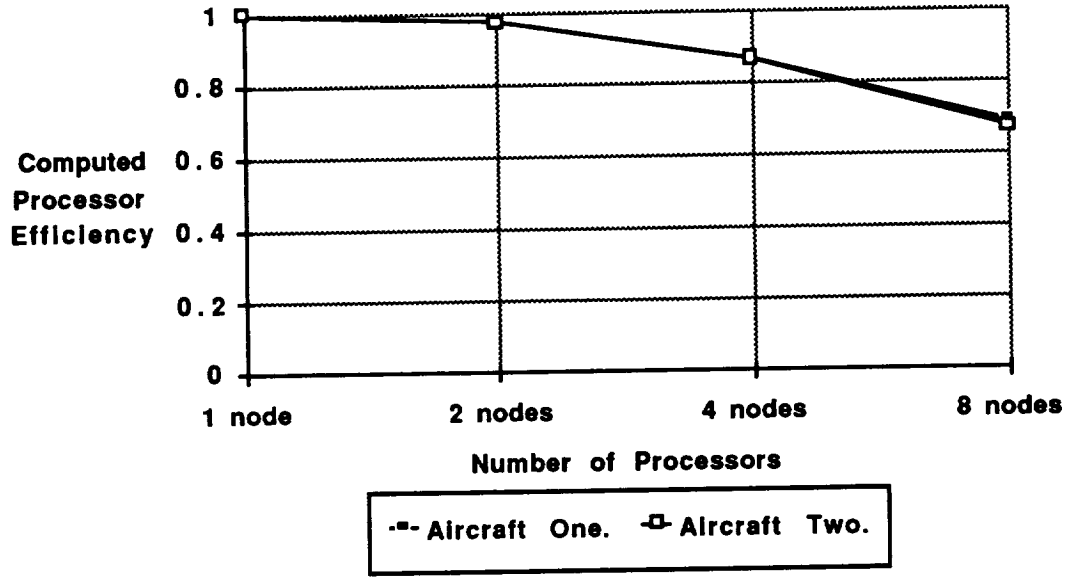

Figure 7.8. Processor Efficiency for Maneuver Evaluation Subroutine

7.8 Results

The speedup and efficiency data for the evaluation knowledge source are very promising and the overall speedup and efficiency data for the main knowledge source show that there is a clear advantage to splitting the problem into parallel execution paths for each aircraft. The data also highlighted some inefficiencies that may be corrected by redesigning parts of the system. Much of the loss of efficiency in the evaluation subroutine can be attributed to the serial link between the relative geometry knowledge source and the situation assessment knowledge source; and to the ratio of execution time for these relatively small knowledge sources to the time they required to send and receive messages. These knowledge sources can be combined into a single specialized knowledge source. This will reduce the evaluation knowledge sources message passing time, cut the number of messages required, and remove the synchronization requirement between the two separate relative geometry and situation assessment knowledge sources.

Cube_CLAWS provides a useful testbed to evaluate the development of a distributed blackboard system. This research shows that the complexity of developing specialized software on a distributed, message-passing architecture is not overwhelming, and reasonable speedups and processor efficiency can be achieved by a distributed blackboard system. The project highlights some of the costs of using a distributed approach to designing a blackboard system. Message passing costs, synchronization costs, and the cost of having multiple processes executing on a processor must be recognized during the system design phase so that their effect on the systems performance can be minimalized.

7.9 Conclusions

Paladin, a computerized air combat tactical decision generator, has been developed to study within-visual-range air combat engagements. The system incorporates modern airplane simulation techniques, sensors, and weapons systems. Paladin uses knowledge–based systems and Artificial Intelligence (AI) programming techniques to address air–to–air combat and agile aircraft in a clear and concise manner.

Paladin models aspects of the decision–making processes used by human pilots through the use of the Situation Assessment knowledge–based system and multiple

modes of operation. The use of distinct modes of operation allows Paladin to perform complex air–to–air combat tasks and generate sound tactical decisions in real–time.

Paladin presents an excellent opportunity to evaluate the use of AI programming techniques and knowledge-based systems in a real-time environment. Paladin clearly shows that the existing maneuver selection and scoring techniques cannot perform well in the modern tactical environment and are not well suited for evaluating agile aircraft. The use of the blackboard problem-solving model has allowed a complex tactical decision generation system to be developed that addresses the modern combat environment and agile aircraft. The ability to integrate Paladin into the DMS offers a unique opportunity to evaluate the performance of the AI-based Paladin software in a real-time tactical environment against human pilots.

Endnotes for Chapter Seven

1 Goodrich and McManus, 1989

2 McManus and Goodrich, 1989

3 Goodrich and McManus, 1990

4 Goodrich and McManus, 1989

5 Goodrich and McManus, 1990

6 G.H. Burgin, et al.: An Adaptive Maneuvering Logic Computer Program for the Simulation of One-on-One Air-to-Air Combat. Vol I and Vol II. NASA CR-2582, CR-2583, 1975.

7 Ibid.

8 McManus and Goodrich, 1989

9 McManus and Goodrich, 1990

10 John W. McManus, Alan Chappell, and P. Douglas Arbuckle. "Situation Assessment in the Paladin Tactical Decision Generation System" In Proceedings NATO–AGARD Guidance and Control Panel 53rd Symposium, October 22–25, 1991.

11 Ibid.

12 McManus, 1990

13 John W. McManus, Alan Chappell, and P. Douglas Arbuckle., 1991

14 McManus, 1989

15 McManus, 1990

16 McManus, 1989

Chapter 8

Validation of the COBS Simulation Model.

Two existing concurrent distributed blackboard systems were used to validate the performance of COBS. One system was selected to validate the performance of a message passing distributed processor implementation of COBS. The other system was selected to validate a shared memory implementation of COBS. The existing systems were developed using prototypes of the COBS daemon driven control structure. The functionality and performance of each of the systems is described in Chapter Seven. The existing systems were used to generate formal blackboard system design specifications. Each design specification was evaluated using the COBS design and analysis tools. The performance analysis metrics were used to measure the performance of the existing systems and the results were compared with the performance predicted by the COBS simulation model. The results of these tests were used to validate the performance of COBS simulation system. The analysis of the validation of the Cube_CLAWS system and the prototype Paladin system are presented in detail in this chapter.

8.1 Simulation of Cube_CLAWS

The Cube_CLAWS distributed message passing blackboard system software was used to develop a COBS design specification. Several minor modifications to the original design were made. The Maneuver Evaluation knowledge source in Cube_CLAWS was modified so that it included the functionality required from the Relative Geometry and Situation Assessment knowledge sources. The original analysis[1] of the Cube_CLAWS system determined that the functionality required from these knowledge sources should be combined to increase system performance and reduce network traffic. Four versions of the Cube_CLAWS system were developed and evaluated:

- Cube has a single copy of the Maneuver Evaluation knowledge source. The knowledge source evaluates all eight trial maneuvers and returns the score and input commands of the highest scoring maneuver.

- Cube2 has two copies of the Maneuver Evaluation knowledge source. The knowledge source evaluates four unique trial maneuvers and returns the score and input commands of the highest scoring maneuver.

- Cube4 has four copies of the Maneuver Evaluation knowledge source. Each knowledge source evaluates two unique trial maneuvers and returns the score and input commands of the highest scoring maneuver.

- Cube8 has eight copies of the Maneuver Evaluation knowledge source. Each knowledge source evaluates a unique trial maneuver and returns the score and input commands of that maneuver.

The knowledge source execution delays for each of the configurations tested are shown in Table 8.1 The execution delay values used were determined by timing the execution of the knowledge sources of Cube_CLAWS on a 16 processor HyperCube[2].

Table 8.1 Cube_CLAWS Knowledge Source Execution Times (msec.)

Knowledge Source	Cube	Cube2	Cube4	Cube8
Synchronize	1	1	1	1
Relative Geometry	10	10	10	10
Situation Assessment	3	3	3	3
Maneuver Evaluation	254	130	73	52
Dynamics	47	47	47	47
Total Execution Time	315	191	134	113

8.2 Validating Cube_CLAWS Simulation Results

Each of the four design specifications was simulated using the COBS simulation system. The results of the simulation runs have been condensed and are included in Appendix F. The results of all of the simulation runs outlined in the following sections show that the COBS produced software executed correctly and produced the expected results. The simulation results are then used to validate the COBS implementation of the formal model for blackboard systems

8.2.1 Analysis of the Cube Simulation Results

The simulation of the Cube design specifications produced results that match the performance the Cube_CLAWS system used to produce the CUBE specifications. All of the knowledge sources were activated and completed at the correct clock times, and the knowledge sources executed in the correct order. No unexpected knowledge source blocking or other anomalies in the daemon-driven control structure occured. Each of the blackboard data objects was accessed (both read and write access) the correct number of times.

The Maneuver Evaluation knowledge source speedup and processor efficiency values, Figures 8.1 and 8.2, are comparable to those produced by the baseline Cube_CLAWS software[3]. Figure 8.1 compares the Maneuver Evaluation knowledge source speedup results of the COBS Simulation with the "Achieved" HyperCube speedup values and the "Best" HyperCube speedup. The "Achieved" HyperCube speedup value was computed using the average of the actual execution times for all of the Maneuver Evaluation knowledge sources. The "Best" HyperCube value was computed using the minimum execution time for the Maneuver Evaluation knowledge source. As discussed in Chapter Seven, in the Cube8 design some processors had more than one knowledge source executing on the processor. The Maneuver Evaluation knowledge sources executing on "overlapped" processors had higher execution times.

The COBS simulation used the average execution time of the non-overlapping Maneuver Evaluation knowledge sources. The average execution delay was used in place of a random execution delay due to the small standard deviation of the execution delays. For all knowledge sources in the Cube_CLAWS system, the standard deviation of the execution delays was less than or equal to one millisecond. Since a one millisecond clock was used to measure the execution delays and the standard deviation is smaller than the resolution of the timing device, the average execution delay values are considered acceptable.

The simulation model correctly models all of the features found in the execution of the actual Cube_CLAWS system. The total execution times for the Cube simulations are comparable to those of the actual Cube_CLAWS system. The simulation results show that knowledge source speedup is close to linear and the processor efficiency is

high (97.5%) for the Cube2 design and the Cube4 design (86.5%). These values match the actual results of the Cube CLAWS system.

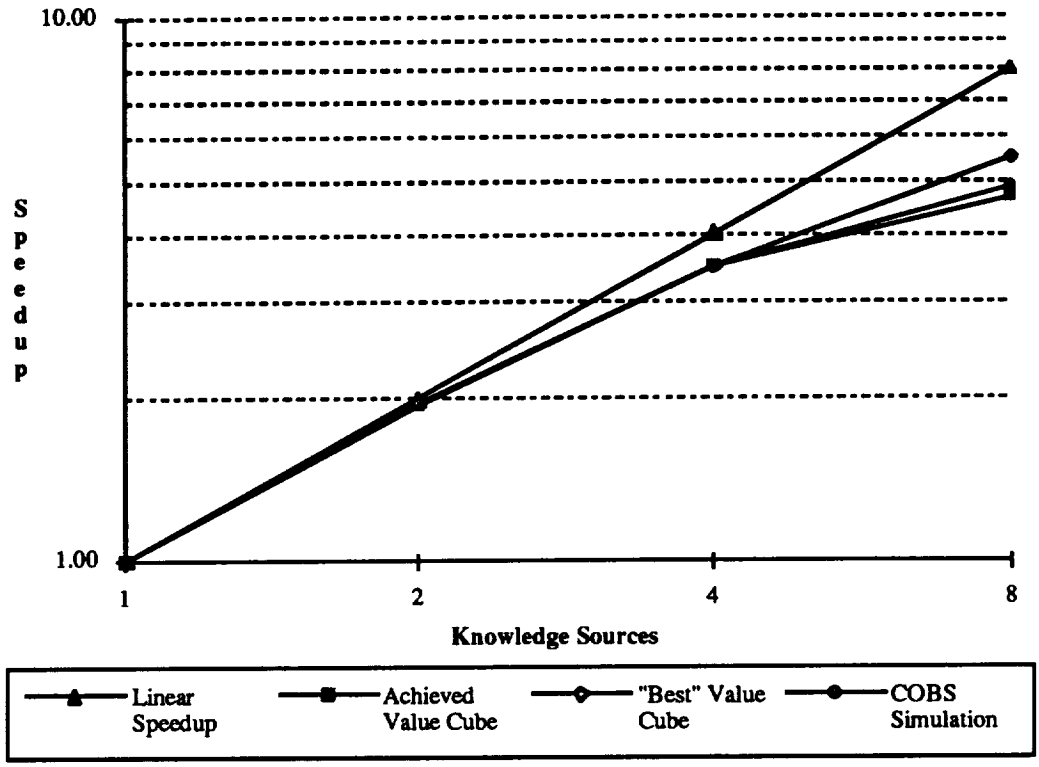

Figure 8.1. Maneuver Evaluation Knowledge Source Speedup

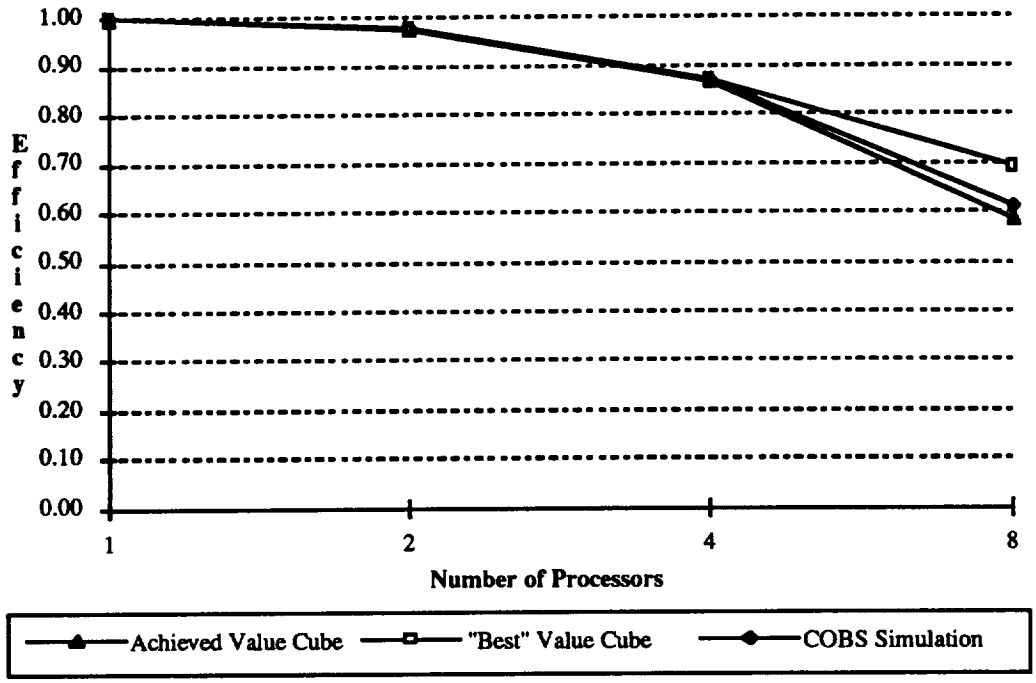

Figure 8.2. Knowledge Source Processor Efficiency

8.2.2 Conclusion

The simulation of the Cube design specifications produced results that validate the performance the COBS Cube design simulations. In each of the designs tested, the knowledge sources were activated and completed at the correct clock times, and the order of knowledge source execution matched the Cube_CLAWS execution order. No anomalies in the daemon-driven control structure occured, and each of the blackboard data objects was accessed the expected number of times, in the expected order. The integrity of the Blackboard data structure and system serializability was maintained. The Maneuver Evaluation knowledge source speedup and processor efficiency values produced by the COBS simulation are comparable to those produced by the baseline Cube_CLAWS software.

Table 8.2 Cube_CLAWS System Execution Times (msec.)

Nodes	Cube "Achieved"	Cube "Best"	COBS Cube
1	315	296	315
2	191	169	191
4	134	113	134
8	114	92	113

The simulation model correctly modeled all of the features found in the execution of the actual Cube_CLAWS system. The total execution times for the COBS simulations are comparable to those of the actual Cube_CLAWS system (Table 8.2). The system speedup and processor efficiency values for each of the simulation runs are comparable to those produced by the baseline Cube_CLAWS software. All differences in the overall system execution time, system speedup, and processor efficiency values are a result of using the average execution delays instead of the achieved execution delay values. These differences only occur in the Cube8 design where "non-overlapping" knowledge source average execution delays are used. If the "achieved" execution delay values are used the simulation results directly match the Cube_CLAWS analysis results.

8.3 Simulation of Paladin

The Paladin distributed blackboard system software was used to develop a COBS design specification. Paladin is implemented using a shared memory daemon-driven control structure. The size and complexity of the current Paladin specification make it impractical to present the full analysis of the Paladin design. The current implementation of Paladin contains twenty knowledge sources. The design analysis results file is over one hundred and twenty pages long, and the simulation software is over one hundred pages long. The results of the analysis of a simpler prototype containing ten knowledge sources are presented in this section. The prototype contains all of the functionality of the current Paladin system, but uses coarser grained knowledge sources. The functionality of each knowledge source in the COBS Paladin design is listed below:

- Dynamics is a full six degree of freedom simulation model of a modern high performance fighter aircraft. The model is implemented in the Advanced Continuous Simulation Language (ACSL)

- Main contains the software used to: extrapolate the position of the opponent at some future time; compute the relative geometry between the aircraft; and score the weapons solutions.

- Situation Assessment determines the current mode of operation and the opponent's instantaneous Intent.

- Throttle Control determines the required throttle position

- Maneuver Selection generates and evaluates the mode dependent sets of trial maneuvers. The highest scoring maneuver's input commands are translated and posted for the Dynamics knowledge source.

8.4 Validating the Paladin Simulation Results

Two Paladin design specifications were simulated using the COBS simulation system. The results of the simulation runs have been simplified and are included in Appendix F. The first design, COBS I, uses a single knowledge source to perform situation assessment and throttle control. The second design, COBS II, splits the functionality of the throttle controller and situation assessment into two separate knowledge sources. In both designs the Throttle Control and Situation Assessment knowledge sources are executing on a Symbolics™ 3650 workstation. All other knowledge sources are executing on Vax 3200™ workstations

The Paladin knowledge sources were executed 100,000 times on a Vax 3200™ workstation to determine an average execution delay. The Main, Throttle Control, Situation Assessment and Maneuver Evaluation knowledge sources were modified to execute their longest execution path, resulting in worst case execution delays. The use of worst case execution times ensures that the results, shown in Table 8.3, are conservative estimates.

The COBS simulation used the average execution time for all of Paladin's knowledge sources. The average execution delay was used in place of a random execution delay due to the small standard deviation of the execution delays. For all

knowledge sources in the Cube_CLAWS system, the standard deviation of the execution delays was less than twenty milliseconds. Since a ten millisecond clock was used to measure the execution delays and the standard deviation is almost equal to the resolution of the timing device, the average execution delay values are considered acceptable.

Table 8.3 Paladin Knowledge Source Execution Times (msec.)

Knowledge Source	COBS I	COBS II
Dynamics	180	180
Main	35	35
Throttle Control	26	16
Situation Assessment	0*	14**
Maneuver Evaluation	22*	22
Total Execution Time	263	253

*Implemented as a single Knowledge Source
**Executed concurrently with Throttle Control

The simulation of the Paladin design specifications produced results that match the performance the existing Paladin system used to produce the specifications. All of the knowledge sources were activated and completed at the correct clock times, and the knowledge sources executed in the correct order. No unexpected knowledge source blocking or other anomalies in the daemon-driven control structure occured. Each of the blackboard data objects was accessed (both read and write access) the correct number of times.

Table 8.4 Paladin System Execution Times (msec.)

Achieved	COBS I	COBS II
263	263	253

The simulation model correctly models all of the features found in the execution of the prototype Paladin system. The total execution times for the COBS simulations are equal to those of the actual Paladin system (Table 8.4).

8.4.2 Simulation Results

The simulation of the Paladin design specifications produced results that validate the performance the COBS Paladin design simulations. In each of the designs tested the knowledge sources were activated and completed at the correct clock times, and the order of knowledge sources execution matched the order of the knowledge sources in the prototype Paladin system. No anomalies in the daemon-driven control structure occured, and each of the blackboard data objects was accessed the expected number of times, and in the expected order. The integrity of the Blackboard data structure and serializability were maintained. The system speedup and processor efficiency values produced by the COBS simulation are equal to those produced by the baseline Paladin software.

The simulation model correctly modeled all of the features found in the execution of the actual Paladin system. The total execution times for the COBS simulations are comparable to those of the actual Paladin system. The system speedup and processor efficiency values for each of the simulation runs are comparable to those produced by the prototype Paladin software.

8.5 Conclusions

The analysis of the Cube_CLAWS and the prototype Paladin system validate the performance of the COBS simulation system. The simulations of both systems computed the expected results and accurately modeled the performance of the systems.

The COBS simulation system models the interaction between the knowledge sources, collisions and hotspots on the shared blackboard data object, and data transactions between the knowledge sources and the blackboard. The analysis provides important information on a design's expected performance, and how the performance can be improved. The validated simulation model allows the system designer to perform trade-off analysis using a low cost, high fidelity simulation system.

Endnotes for Chapter Eight

1 McManus, 1989.

2 Ibid.

3 Ibid.

Chapter 9

Utilizing the Concurrent Object–Oriented Blackboard System.

Two existing concurrent distributed blackboard systems were used to evaluate the performance of the COBS design, simulation, and analysis techniques. The analysis of the Cube_CLAWS system and the prototype Paladin system are presented in detail in this chapter. Each of the blackboard system designs analyzed has a unique set of characteristics and performance constraints. Based on the characteristics and performance constraints for each of the applications, a unique set of specialization, serialization, and interdependence values that are considered "acceptable" and "high" for that specific application are determined.

9.1 Analysis of the Cube_CLAWS Design Specifications

Each of the four design specifications presented in Chapter Eight were evaluated using the COBS design and analysis tools. The resulting analysis output files ranged in length from sixteen to eighty-four pages. The analysis files have been condensed and are included in Appendix F. The output overlap, output to input connectivity sets; and all cases where the specialization values, interdependence values, or serialization values are equal to zero have been dropped. Specialization values and serialization values that are greater that 0.5 are considered "high" for the Cube designs, values that are less that or equal to 0.5 are considered "acceptable". High interdependence values are expected for some knowledge sources due to the cooperative nature of the task the system is solving. The analysis of the Cube design specifications points out the importance of computing both the knowledge source interdependence value and knowledge source serialization value. The importance of trading-off functional connectivity versus serial connectivity is discused at the end of this section.

9.1.1 Analysis of the Cube Design Specification

The analysis of the Cube design specification shows that the system has no knowledge source pairs with high specialization values, twenty-six knowledge source pairs with high interdependence values ($\Pi > 0.5$), and no knowledge source pairs with high serialization values. The interdependence values show that eight sets of knowledge sources have direct functional connectivity ($\Pi = 1.0$). This is not

considered unacceptable since all of the functionally connected knowledge source pairs have acceptable ($\Sigma < 0.50$) serialization values. The Maneuver Evaluation knowledge source and Dynamics knowledge source are functionally connected in this design ($\Pi = 0.75$) but the serialization values are low ($\Sigma = 0.10$). If a single knowledge source is going to be used to evaluate trial maneuvers, the Maneuver Evaluation knowledge source and Dynamics knowledge source should be combined to decrease the functional connectivity values and network traffic and increase system efficiency.

9.1.2 Analysis of the Cube2 Design Specification

The analysis of the Cube2 design specification shows that the system has thirty knowledge source pairs with high interdependence values and two knowledge source pairs with high serialization values ($\Sigma = 0.67$). Sixteen sets of knowledge sources have direct functional connectivity. The increase in functionally connected processors is a result of adding knowledge sources to perform maneuver evaluation. None of the functionally connected knowledge source pairs have high serialization values. The Maneuver Evaluation knowledge source and Dynamics knowledge source are still functionally connected in this design ($\Pi = 0.85$), but the serialization values are staying low ($\Sigma = 0.19$) as additional maneuver evaluation knowledge sources are added. The benefits of combining the Maneuver Evaluation knowledge sources and Dynamics knowledge sources are decreasing due to the increased opportunity for concurrent Maneuver Evaluation knowledge source execution and system speedup.

9.1.3 Analysis of the Cube4 Design Specification

The analysis of the Cube4 design specification shows that the system has forty-six knowledge source pairs with high interdependence values and two knowledge source pairs with high serialization values. Twenty sets of knowledge sources have direct functional connectivity ($\Pi = 1.0$). This is not considered unacceptable since none of the functionally connected knowledge sources have high serialization values. The Maneuver Evaluation knowledge source and Dynamics knowledge source are still functionally connected in this design ($\Pi = 0.85$), but the serialization values are still low ($\Sigma = 0.19$). The benefits of combining the Maneuver Evaluation knowledge sources and Dynamics knowledge sources continue to decrease as concurrent Maneuver Evaluation knowledge source execution and the opportunity for improved system speedup increases.

9.1.4 Analysis of the Cube8 Design Specification

The analysis of the Cube8 design specification shows that the system has ninety knowledge source pairs with high interdependence values and two knowledge source pairs with high serialization values. The interdependence values show that thirty-six sets of knowledge sources have direct functional connectivity ($\Pi = 1.0$). This is not considered unacceptable since none of the functionally connected knowledge sources have high serialization values. The Maneuver Evaluation knowledge source and Dynamics knowledge source are still serially connected in this system ($\Sigma = 0.85$) but the serialization values are staying low ($\Sigma = 0.19$) as additional maneuver evaluation knowledge sources are added.

The benefits of combining the Maneuver Evaluation knowledge sources and Dynamics knowledge sources have decreased to the point that it may not be a good trade-off to combine the knowledge sources. The system speedup gained by the concurrent execution of the Maneuver Evaluation knowledge sources may outweigh the benefit of the decrease in functional connectivity and message passing that results from merging the two knowledge sources.

9.1.5 Design Specification Analysis Conclusions

The analysis of the four Cube design specifications shows some of the difficulties facing the concurrent blackboard system designer. Each design has a unique balance between knowledge source interdependence and serialization that will result in the "acceptable" system performance. As the complexity of the design specification analysis grows, evaluating the affect of design trade-offs is harder to determine.

The Cube examples point out some of the trade-offs facing the concurrent blackboard system designer. In the smaller Cube designs the benefits of combining the Maneuver Evaluation knowledge sources and Dynamics knowledge sources are clear. The combined knowledge source has better functional connectivity and serialization values, and the design reduces traffic on the blackboard and the communications network. These reductions result in reduced system execution times and increased system performance.

Adding Maneuver Evaluation knowledge sources to the Cube designs increases the opportunities for concurrent knowledge source execution. This increase in concurrent knowledge source execution must be balanced against the increased traffic on the

blackboard and the communications network. If the knowledge source execution times become smaller then the overhead required to pass data to and from the knowledge source, the design is not efficiently utilizing the available processors, is generating unproductive message passing overhead, and is generating unnecessary traffic on the communications network.

9.2 Analysis of the Cube Simulation Results

The Maneuver Evaluation knowledge source speedup and processor efficiency values, Figures 9.1 and 9.2, are comparable to those produced by the baseline Cube_CLAWS software[1]. Figure 9.1 compares the Maneuver Evaluation knowledge source speedup results of the COBS Simulation with the "Achieved" HyperCube speedup values and the "Best" HyperCube speedup. The simulation results show that knowledge source speedup is close to linear and the processor efficiency is high (97.5%) for the Cube2 design and the Cube4 design (86.5%). These values match the actual results of the Cube CLAWS system.

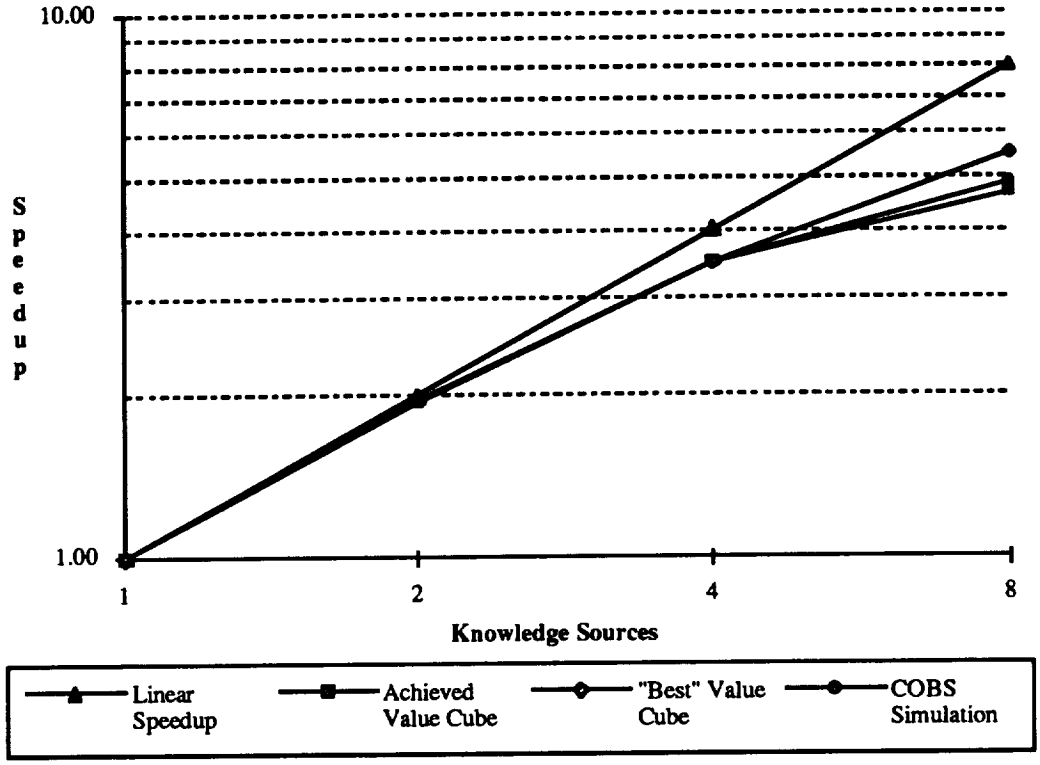

Figure 9.1. Maneuver Evaluation Knowledge Source Speedup

Knowledge source speedup and processor efficiency drop off sharply in the Cube8 configuration. The Cube8 design speedup is only 4.88 and processor efficiency has dropped to 61.0%. The drop-offs in speedup and efficiency are due to the way the knowledge sources evolve as Maneuver Evaluation knowledge sources are added to the design. Each Maneuver Evaluation knowledge source evaluates fewer maneuvers and becomes smaller as more are added to the system. This reduces the time each instance of the knowledge source requires to perform its computation, but the knowledge source's communication costs stay relatively constant. The Cube8 design approaches the point where the communications costs are greater than the time required to compute the knowledge source's function.

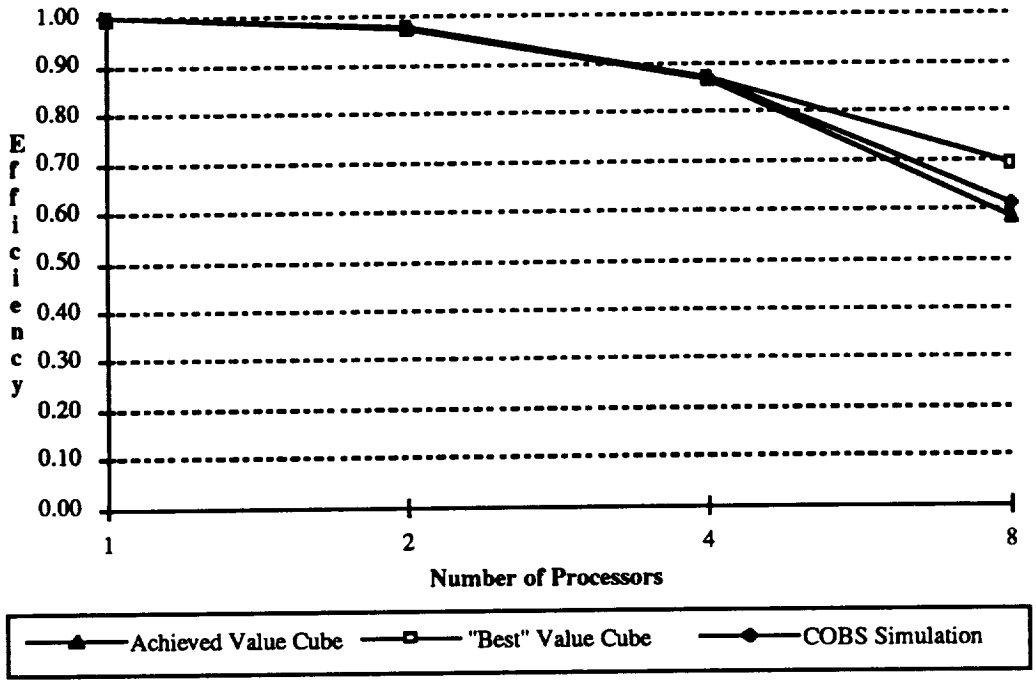

Figure 9.2. Knowledge Source Processor Efficiency

The speedup and processor efficiency values for each of the Cube designs, Figures 9.3 and 9.4, are comparable to those produced by the baseline Cube_CLAWS software[2]. The simulation results show that system speedup is acceptable and the processor efficiency is high (82.5%) for the Cube2 design and the Cube4 design (58.8%). Knowledge source speedup and processor efficiency drop off sharply in the Cube8 configuration. The Cube8 design speedup is only 2.79 and processor efficiency has dropped to 34.8%.

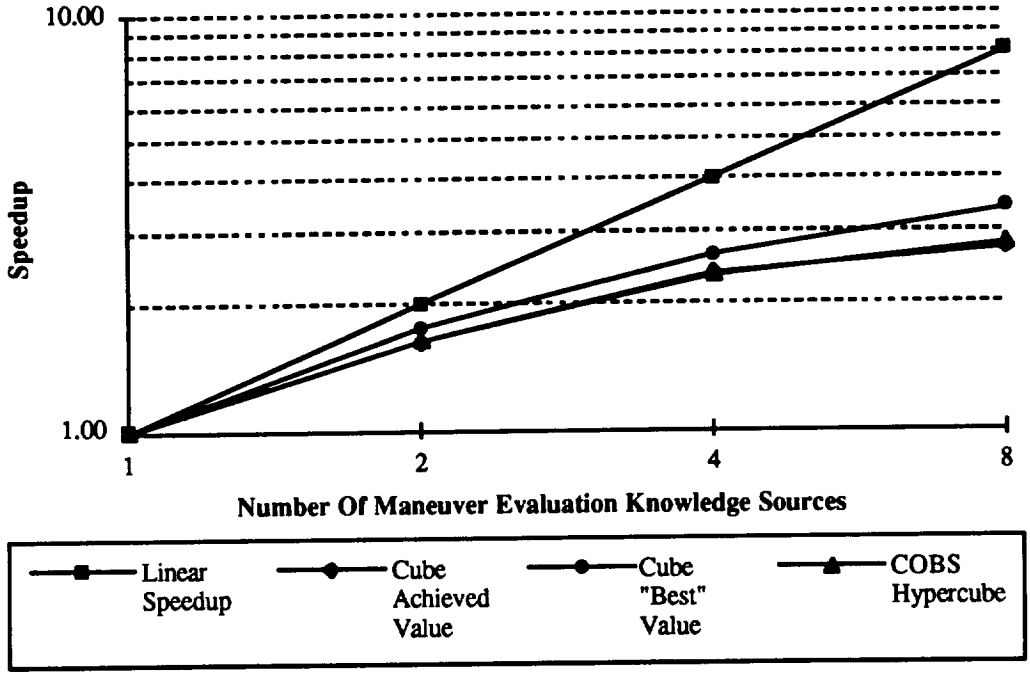

Figure 9.3. Cube Design Speedup

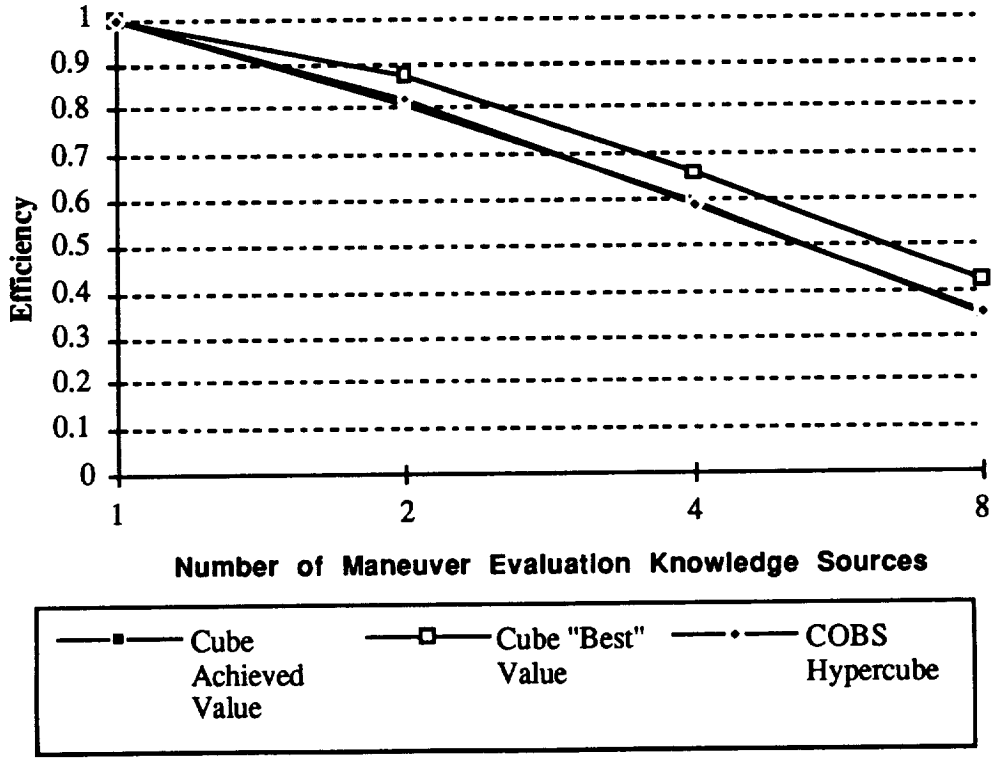

Figure 9.4. Cube Knowledge Source Efficiency

This drop-off is due to the way the problem evolves as Maneuver Evaluation processors are added to the design. Table 8.1 shows that as maneuver evaluation knowledge sources are added to the system, the time required to compute the next maneuver decreases. As the maneuver selection time decreases the total system execution time decreases; but more importantly, the percentage of the total execution time spent performing maneuver evaluation decreases. The effect of speeding up the maneuver evaluation knowledge source on the systems total execution time decreases as the other knowledge sources in the design become a larger percentage of the system's execution time.

9.3 Analysis of the Paladin Design Specifications

The prototype Paladin design specification was evaluated using the COBS design and analysis tools. The condensed analysis file is included in Appendix F. The output overlap, output to input connectivity sets; and all cases where the specialization values, interdependence values, or serialization values are equal to zero have been dropped. Specialization values and serialization values that are greater that 0.75 are considered "high" for these examples, specialization values and serialization values that are less than or equal to 0.75 are considered "acceptable". High interdependence and serialization values are expected for some of the knowledge sources due to the cooperative nature of the task the system is solving. Paladin is designed to allow knowledge-based systems to cooperate with a high fidelity aircraft simulation model. The problem was decomposed on a processor/task level, and the prototype utilizes large grain knowledge sources. As the prototype evolves, smaller more specialized knowledge sources are developed. The later designs have better functional connectivity and serialization values.

The analysis of the Paladin design specification shows that the system has no knowledge source pairs with high specialization values, twelve knowledge source pairs with high interdependence values, and ten knowledge source pairs with high serialization values. The interdependence values show that eight sets of knowledge sources have direct functional connectivity ($\Pi = 1.0$). As stated above, this is not considered unacceptable due to the design goals and strict design constraints. The knowledge source serialization values show that eight sets of knowledge sources have direct serial connectivity ($\Sigma = 1.0$).

The Paladin design specification points out several of the difficulties facing the concurrent blackboard system designer. The Paladin system has a strict set of design constraints. The Dynamic knowledge source is a large ACSL model and cannot be decomposed. The Situation Assessment and Throttle Control knowledge sources are implemented in LISP for real-time execution on specialized workstations. The designer must work within these constraints to develop the best system possible.

9.4 Evaluation of Paladin Simulation Results

Figure 9.5 compares the speedup results of the two COBS Paladin designs described in Chapter Eight. Figure 9.5 compares the processor efficiency values for the two Paladin designs The simulation results show that design speedup and processor efficiency are unacceptable for the COBS I and COBS II designs. The additional processors are not being applied to the most time consuming task in the process. The Dynamics knowledge source requires approximately 70% of the execution time in both designs. The most effective way to improve system speedup and processor efficiency is to reduce the execution time of the Dynamics knowledge source.

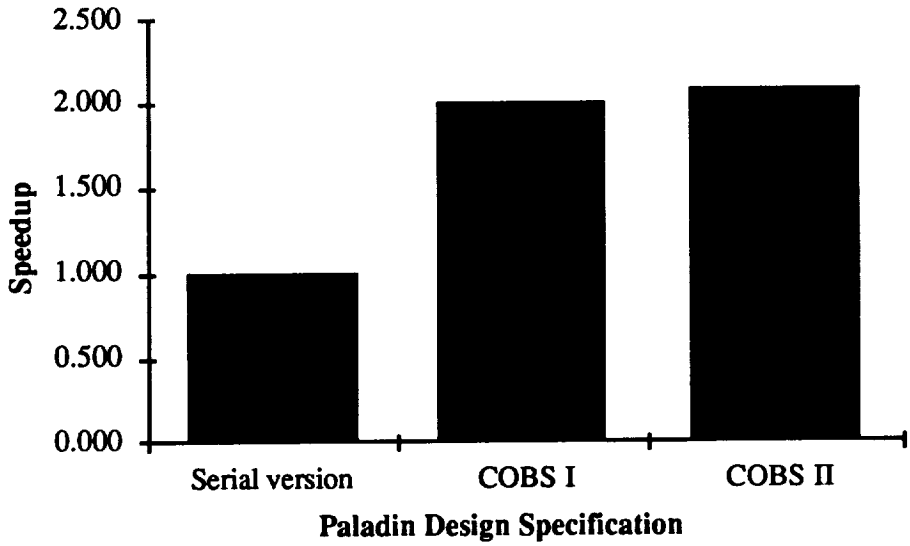

Figure 9.5 Paladin Speedup

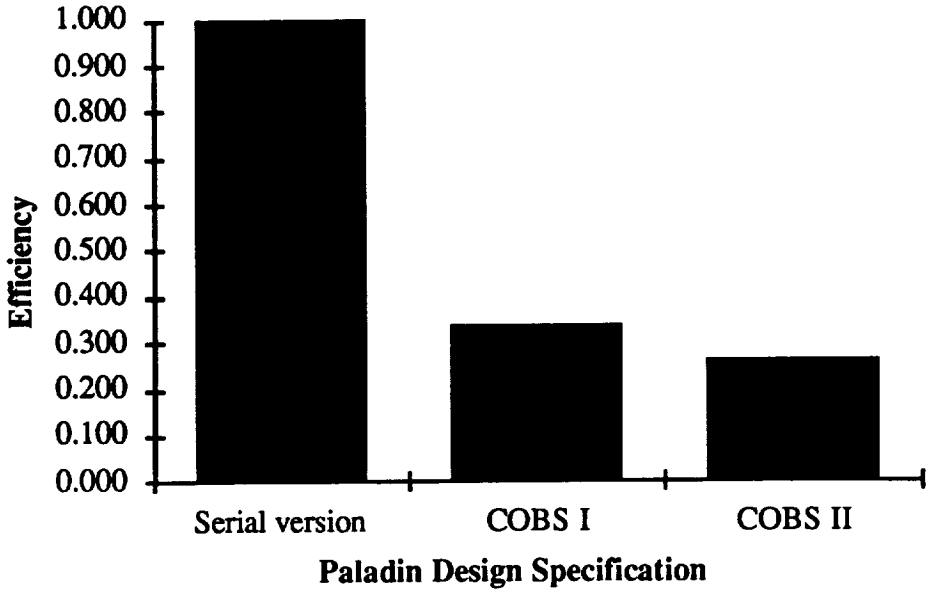

Figure 9.6 Processor Efficiency

9.5 Evaluation of Alternative Hardware Configurations

COBS provides the designer the ability to simulate a design specification utilizing the performance characteristics of a heterogeneous computer network. The simulation of the Cube design specifications described above used the characteristics of a 16 processor HyperCube[3]. A blackboard system design specification can be evaluated using the characteristics of any set of processors by modifying the execution delays of the knowledge sources to match the performance expected on the target processors. This allows the designer to evaluate a design's performance on a heterogeneous computer networks that may not be available.

9.5.1 Evaluation of Cube_CLAWS

Without simulating the proposed Cube designs it is hard to determine the trade-offs involved in combining the knowledge sources and what the potential payoffs are. The Cube design analysis points out that each of the proposed designs has a unique set of strengths and weaknesses. The Cube design has the best functional connectivity and serialization values. Each of the other Cube design provides an increasing level of concurrent knowledge source execution and potential for system speedup.

The designer must use the results of the design analysis, the simulation runs, and their knowledge of the specific problem being solved to determine which system best meets the design and performance constraints for the Cube system.

The Cube_CLAWS knowledge sources were executed across a set of processors to determine knowledge source execution delays. Each knowledge source was executed 100,000 times on each processor to determine an average execution delay on that processor. The knowledge sources were modified to execute their longest execution path, resulting in worst case execution delays. The use of worst case execution times ensures that the results, shown in Table 9.1, are conservative estimates. The cost of shipping a data packet between the Blackboard handler and a remote knowledge source was computed by shipping 20,000 varying sized packets across the EtherNet network used to connect the knowledge sources with the blackboard. The average time to build a packet, and send it to the remote knowledge source is 7 milliseconds.

Table 9.1 Processor Evaluation (msec.)

Processor	Synch.	Relative Geometry	Situation	Evaluation	Dynamics
HyperCube™	1.0	10.0	3.0	254.0	47.0
Vax II™	1.2	13.94	4.63	234.75	66.76
Vax 3200™	0.5	4.56	1.52	76.86	21.86
Sparc SLC™	0.3	3.10	1.03	52.14	14.83
Sparc II™	0.1	1.23	0.41	20.77	5.91
DS5000™	0.062	0.62	0.20	10.39	2.95

The processor evaluation data was used to evaluate the effect of placing the Dynamics knowledge source on different processors, and holding the performance of the other knowledge sources constant. The execution delay for the Dynamics knowledge sources was modified using the results of the processor evaluation tests and the simulations were run using the new execution delay data.

Evaluating alternative hardware systems allows the designer to measure the effect of moving knowledge sources to other processors in the heterogeneous computing network. The results of moving the Dynamics knowledge source are shown in figures 9.7 and 9.8. Figure 9.7 shows the projected speedup values for each design as the Dynamics knowledge source is moved to faster processors. Figure 9.8 shows the

projected processor efficiency values for each design as the Dynamics knowledge source is moved to faster processors. Moving the Dynamics knowledge source to the fastest available processor increases the Cube8 system speedup value and processor efficiency by 40.8%. The designer can use this type of "What If" simulation analysis to study the combined effects of changing the distribution of knowledge sources across the available processors and adding knowledge sources to the design.

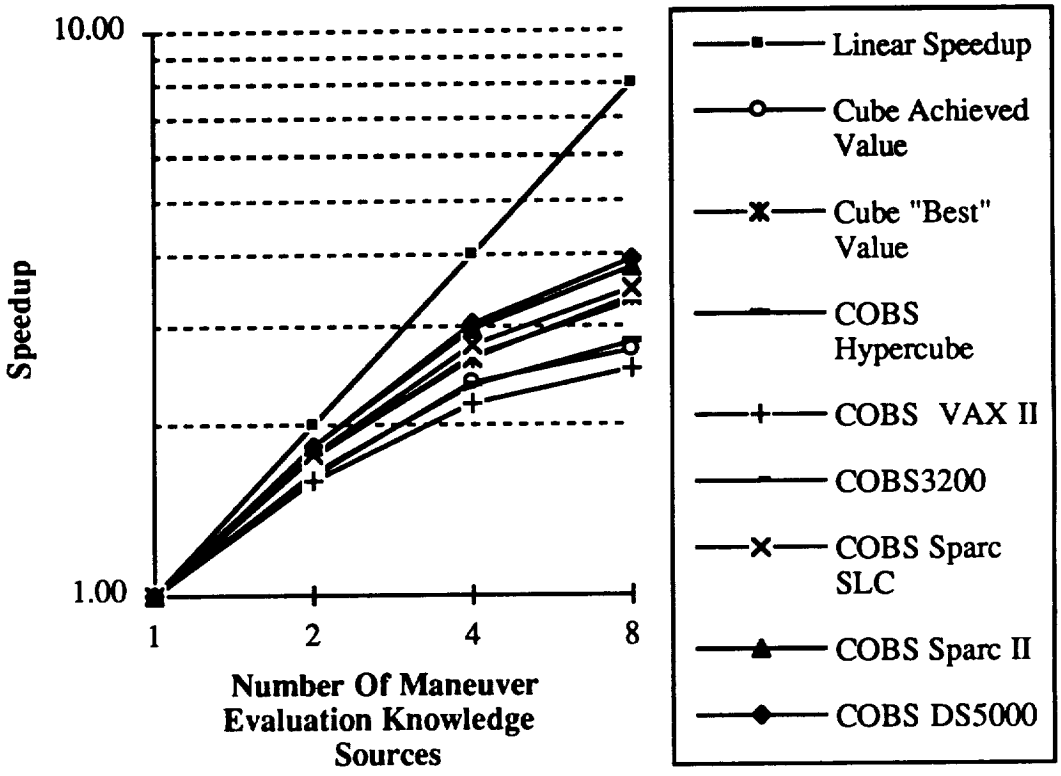

Figure 9.7. System Speedup

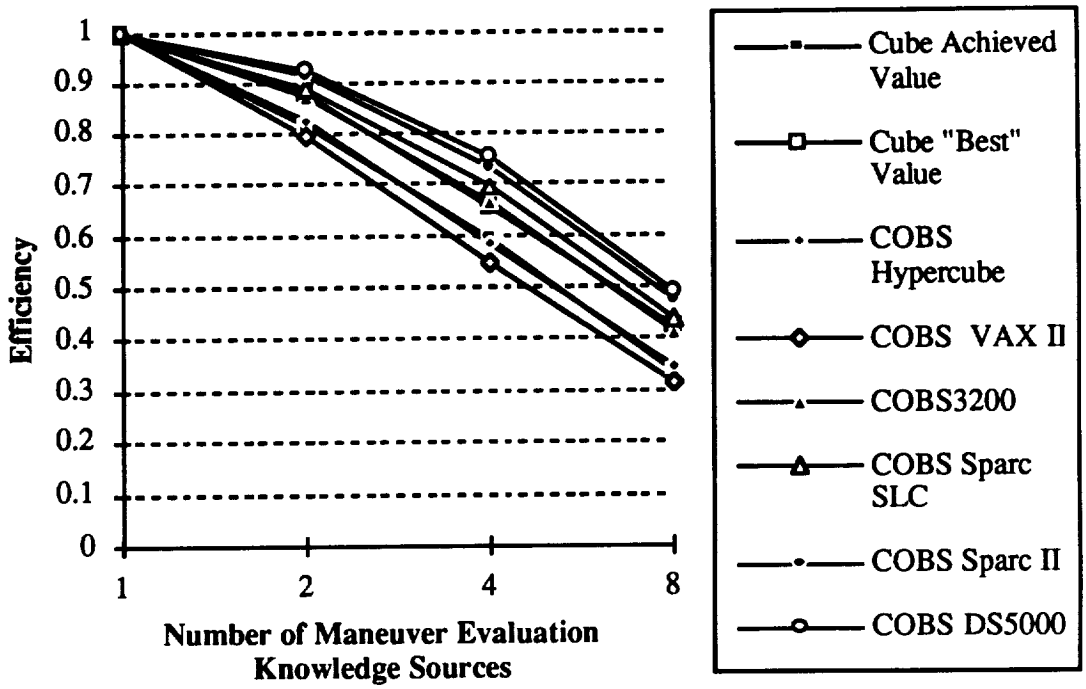

Figure 9.8. Processor Efficiency

9.5.2 Evaluation of Paladin

Speedup and processor efficiency for the Paladin designs can be improved by moving the Dynamics knowledge source to a faster processor. The execution time of the Dynamics knowledge source across a set of available processors are shown in Table 9.2. The results of the knowledge source timings presented in Table 8.2 were used to scale the execution times to the faster processors.

Table 9.2 Dynamics Knowledge Source Execution Times (msec.)

Processor	COBS I	COBS II
Vax 3200™	180	180
Sun Sparc SLC™	121	121
Sun Sparc II™	47	47
DEC DS5000™	23	23

Figures 9.9 and 9.10 show the effect of moving the Dynamics knowledge source to faster processors. The speedup values for both designs rise to acceptable levels. Using the DS5000 processor to host the Dynamics knowledge source, COBS I achieves a

speedup of 4.95 and COBS II achieves a speedup of 5.46. COBS I's processor efficiency rises to 82.6% and COBS II achieves a processor efficiency of 68.3%.

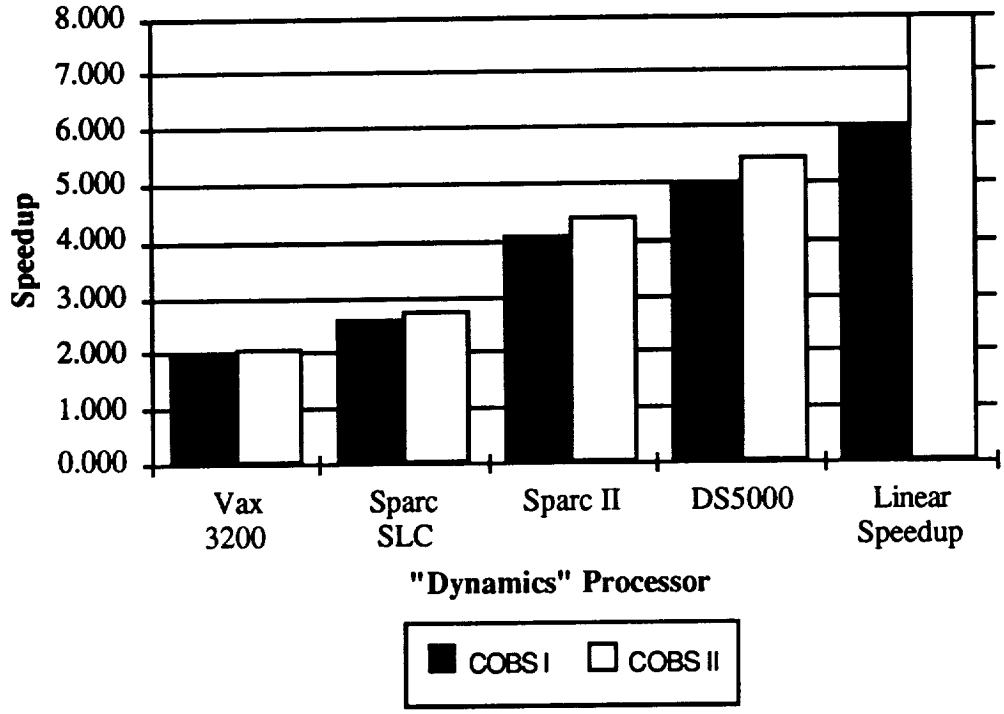

Figure 9.9 Paladin Speedup

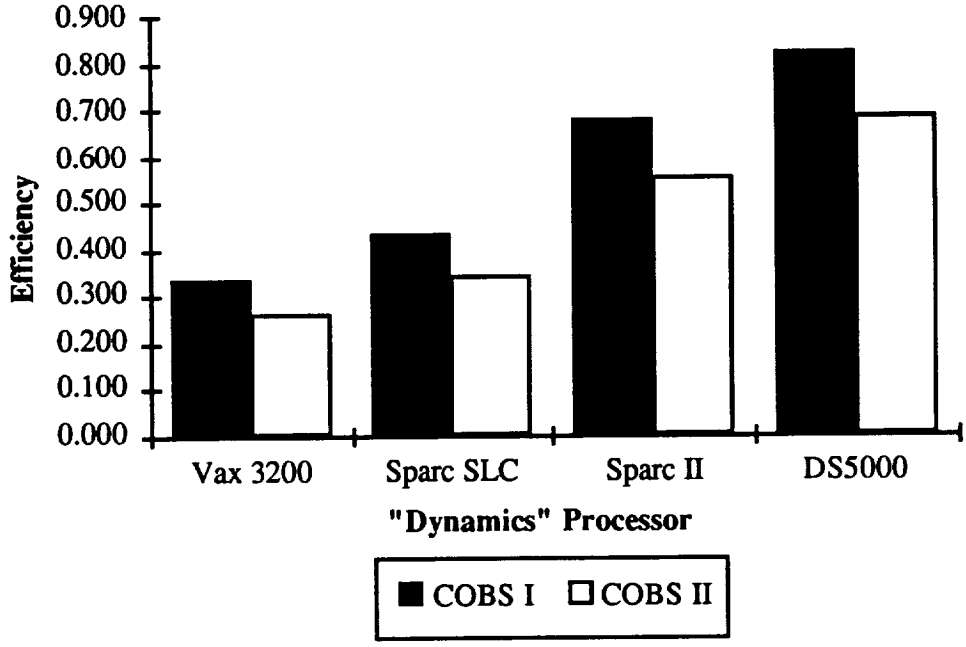

Figure 9.10 Processor Efficiency

This example shows effective use of a heterogeneous computing network and cooperation between knowledge-based software and conventional software. The use of the heterogeneous computing network allows each of the knowledge sources to execute on the most favorable processor. Moving the Dynamics knowledge source to a faster processor increases Paladin's speedup and efficiency values without requiring changes to the systems design.

The Paladin design makes effective use of the available resources, speeding up the "slower" knowledge sources without applying the "overkill syndrome" of running the whole system on the fastest available processors, regardless of the efficiency of the implementation or speedup achieved. Overkill syndrome can result in inefficient system implementations where many of the processors are idle a for large periods of time. Moving all of the knowledge sources to faster processors would increase the systems speedup when compared to the original implementation. This would reintroduce the processor efficiency problems of the original design, since all of the knowledge source would be executing proportionally faster. The designer needs to find the proper balance between system speedup and processor efficiency that meets the needs of their specific application.

9.6 Conclusions

The Cube system forces the designer to decide what performance versus processor efficiency trade-offs to make. The constraints on the design are very simple, and all of the knowledge sources can run on any of the available processors. The decision to move the Dynamics knowledge source to the DS5000™ is clear cut. The move increases design speedup and processor efficiency for all of the proposed designs. The decision of how many Maneuver Evaluation knowledge sources to implement is more difficult. The researcher must include real-time execution requirements and processor availability in the decision. The Cube4 design has good processor efficiency results, but may not meet the designers real-time requirements. If Cube4 doesn't meet the designers real-time execution requirements the faster, but less efficient, Cube8 design must be used.

The Paladin design has tighter constraints placed on it. The knowledge-based systems achieve their best performance executing on the specialized workstations, and the Dynamics knowledge source cannot be reduced into smaller components. The decision to split the Situation Assessment and Throttle Control knowledge sources is

complex. The researcher must include real-time execution requirements and processor availability in the decision. When the DS5000™ is used to host the Dynamics knowledge source, both COBS designs have good processor efficiency results. The slower COBS I design may not meet the systems real-time requirements. If COBS I doesn't meet the real-time execution requirements the faster, but less efficient, COBS II design must be used.

Endnotes for Chapter Nine

1 McManus,1989.

2 Ibid.

3 Ibid.

Chapter 10

Summary

Blackboard systems can be viewed as a set of independent knowledge sources working to solve a common problem. Each knowledge source independently monitors the global blackboard data structure and determines when it can advance the current solution of the problem. Concurrent blackboard systems show great promise for increasing the power of the blackboard problem-solving model. The blackboard problem-solving model has several inherent features that can be exploited to create efficient concurrent systems.

10.1 The Formal Model for Blackboard Systems

I have developed Formal Model for blackboard systems and a set of design and analysis techniques for concurrent blackboard systems that support the formal model. The Formal Model for Blackboard Systems provides the designer with a formal "Design -> Simulate -> Analyze -> Implement" process to develop concurrent blackboard systems. The formal model specifies the features and components a design must include to be considered a valid blackboard system design. The formal model requires a description of the blackboard data structure, the function computed by each knowledge source, and the knowledge source's input and output variables. The formal model for blackboard systems and the design and analysis techniques described in this dissertation are unique. This is the first use of a formal model and a set of structured design and analysis techniques for blackboard systems found in the literature. The formal model and the design, analysis, and simulation techniques provide blackboard system designers with the tools required to develop systems that realize the full power of the blackboard system problem-solving model. The design and analysis techniques I have developed for concurrent blackboard systems are generic, and can be applied to any concurrent processing problem.

10.1.1 Blackboard System Design and Analysis Techniques

Knowledge source connectivity analysis is a method for evaluating formal blackboard system design specifications. Connectivity analysis determines the data transfers between the knowledge sources and data migration across the blackboard. Knowledge source specialization, serialization, and knowledge source interdependence

values are calculated, and feedback loops in the blackboard system design are detected. These techniques evaluate a design specification before the blackboard system is developed. This allows the designer to address knowledge source interdependence problems, connectivity problems, and design feedback loops during the initial design process.

Knowledge source connectivity analysis produces a measure of the interdependence between knowledge sources and a measure of the data transfers across the blackboard and through the communications network. From this analysis the designer can determine if the knowledge sources have been partitioned correctly and if the required level of knowledge source specialization has been achieved. The design and analysis techniques are generic and are not application dependent.

10.1.2 Blackboard System Simulation Modeling

The validated Blackboard System Simulation model is used to evaluate proposed blackboard system designs before they are implemented. The simulation model is generic, and can be used to model any blackboard system that can be expressed using the formal blackboard system model. The simulation system has been applied to concurrent message-passing distributed blackboard systems, and concurrent shared memory blackboard systems.

The simulation system models the interaction between the knowledge sources, collisions and hotspots on the shared blackboard, and data transactions between the knowledge sources and the blackboard. The analysis provides important information on a designs expected performance, and how the performance can be improved. This allows the system designer to perform trade-off analysis using a low cost, high fidelity simulation system.

10.1.3 The Concurrent Object–Oriented Blackboard System

The Concurrent Object–Oriented Blackboard System (COBS) system is an implementation of the formal blackboard model and the blackboard system design, analysis, and simulation techniques. COBS is a method for implementing concurrent blackboard systems utilizing of a set of highly independent, highly specialized knowledge sources that cooperate using a shared object-oriented blackboard. COBS removes the centralized control module, instead implementing a set of object–oriented blackboard data objects. Blackboard control and knowledge source selection is

achieved using daemons attached to the blackboard data objects. The daemons activate the blackboard handlers when data elements on the blackboard are updated. The blackboard handlers control knowledge source activation and provide the knowledge sources read/write access to the blackboard.

COBS is designed to execute on a heterogeneous computer network with a centralized shared memory parallel processor hosting the blackboard data structure. Daemon driven blackboard handlers manage the blackboard data objects and the data transfers to and from the knowledge sources. The object–oriented blackboard and the knowledge source specific blackboard handlers remove the need for the centralized control structure. The blackboard handlers have direct access to the blackboard data structure and can directly monitor the status of the blackboard data objects. COBS uses an n-readers/one-writer protocol to guarantee blackboard consistency. All blackboard handlers use a two-phase locking protocol or read/write serialization to guarantee knowledge source serializability.

10.1.4 Validation of COBS

Two existing concurrent distributed blackboard systems were used to evaluate the performance of COBS. Formal blackboard system design specifications were developed using the existing systems. Each design specification was evaluated using the COBS design, analysis, and simulation tools, and the results were compared to the performance of the existing systems. The results of these tests were used to validate the performance of COBS. The simulations of both systems computed the correct results and accurately modeled the performance of the systems.

10.2 Future Research

Several interesting opportunities for continuing the refinement of the formal model and design, simulation and analysis techniques for concurrent blackboard systems exist. COBS will be used to develop the final version of the Paladin system. The final version of Paladin is more complex that the existing blackboard systems. This research will provide the information required to refine the formal model and the blackboard system design and analysis techniques.

Two other research projects currently plan to utilize the formal model to develop real-time concurrent blackboard systems:

The first project is a Autoland Progress Monitor (APM) system being developed at NASA Langley Research Center for testing in the Transport Systems Research Vehicle (TSRV). TSRV is a research B737 aircraft operated by the Advanced Transport Operating Systems (ATOPS) program. The autoland progress monitor is an knowledge-based control mode switching logic for the flight control system, and will provide feedback to the pilot as needed. This system requires redesigning portions of the the existing flight control system and integrating knowledge-based processing into system.

The second project is the Rotocraft Pilot's Associate Program. This project is an Army Aviation project managed by the Aviation Applied Technology Directorate at Ft. Eustis, Virginia. Discussion is underway to provide a version of the COBS system to be used to evaluate proposed Rotocraft Pilots Associate systems. The Rotocraft Pilot's Associate program is larger and more complex that any currently implemented blackboard systems, and has tight real-time execution constraints. The design and development of a system this size (approximately 100 knowledge sources) without a formal model and an automated set of design and analysis techniques is infeasible. The sheer size of the system and the complexity of the knowledge source interactions make hand analysis of this system impossible. COBS will be used to evaluate proposed software design specifications and proposed hardware architectures.

Discussions are also underway to implement the formal model and the daemon-driven control structure on a associative memory hardware configuration. The associative memory based implementation would be used for real-time embedded avionics systems applications. This research will apply the formal model to a third hardware architecture. Currently the formal model has been tested on a message passing distributed processor system and a shared memory distributed processing system.

The formal model and the design and analysis techniques provide concurrent blackboard systems designers with a unified design technique and a consistent set of design, simulation, and analysis tools. The power and usefulness of the the formal model and design and analysis techniques have been recognized by the blackboard systems community and are now being applied to a wide range of independent applications and hardware environments.

Glossary

Blackboard Model

The blackboard model consists of three major components

Knowledge Sources.

The knowledge needed to solve the problem is partitioned into separate and independent *knowledge sources.*

The blackboard data structure.

The problem solving state data are kept in a global database, the *blackboard.* Knowledge sources produce changes to the Blackboard that lead incrementally to a solution to the problem. The knowledge sources communicate and interact solely through the blackboard.

Control.

The knowledge sources respond opportunistically to changes to the blackboard. There is no control component specified in the blackboard model.

Blackboard Simulation Model

The blackboard simulation model is used to predict the performance of blackboard systems. The knowledge source distribution information and the results of the knowledge source connectivity analysis to build a simulation model of the system.

Expert System

Expert systems are computing systems that embody organized knowledge concerning some specific area of human expertise, sufficient to perform as a skillful and cost effective consultant. The rules and/or knowledge incorporated in the system have been extracted by consulting with an "expert" in the problem domain.

Explicit Input Variables

Explicit input variables specify a explicit blackboard data object that is used as the input variable to the knowledge source. The knowledge source can only use the specified blackboard data object as the input variable.

Explicit Knowledge Sources

Explicit Knowledge Sources have only explicit input variables.

Functionally Accurate, Cooperative FA/C

'Functionally Accurate' refers to the generation of acceptably accurate solutions without the requirement that all shared intermediate results be correct and consistent (semantically consistent). 'Cooperative' refers to the iterative, coroutine style of knowledge source interaction in the blackboard system. The hope of this approach is that much less communication is required to exchange the tentative results then the communication of all the raw data and processing results. Blocking and synchronization overhead can be reduced or eliminated resulting in increased parallelism and opportunistic problem solving.

Generic Input Variables

Generic input variables specify a class or type of blackboard data object that can be used as the input variable to a knowledge source. The use of generic input variables allows development of knowledge sources that function on a class of blackboard data objects.

Generic Knowledge Sources

Generic knowledge sources have only generic input variables.

Knowledge-Based System

These programs use a large amount of information about the domain under discussion to help understand the problem being solved. The knowledge is usually stored within the program using some knowledge representation scheme like logic, procedural semantics, semantic networks, frames, or objects.

Knowledge Source Connectivity Analyzer

The knowledge source connectivity analyzer is a tool used during the blackboard system design phase. The knowledge source connectivity analyzer uses the knowledge source distribution and the input and outputs of each knowledge source to develop a knowledge source connectivity graph between the knowledge sources. The connectivity data is used to determine knowledge source connectivity and knowledge source specialization.

Knowledge Source Organizer

The knowledge source organizer is a tool used during the blackboard system design phase. The knowledge source organizer is used to decompose the problem into its component parts and to aid in the initial selection of knowledge sources and the blackboard data structure.

Mixed Knowledge Sources

Mixed Knowledge Sources have both explicit and generic input variables.

Production Systems

A production system repeatedly looks for production rules whose left-hand sides (LHS) matches working memory (WM). A LHS matches WM if there is an assignment of values to its variables such that the patterns match. A production system has three main components: a working memory, a production memory, and a rule interpreter.

The working memory (WM) is a set of dat structures that represent the current state of the system.

The production memory (PM) is a set of rules, each consisting of a left-hand side (LHS) and a right-hand side (RHS).

The LHS is the condition side of each production rule. If the condition matches WM then the condition is satisfied and the rule is fired.

The RHS consists of a set of actions that are executed if the rule is fired.

The rule interpreter.

The rule interpreter applies the rules to the working memory. A PS interpreter works in a "recognize-act" cycle. The interpreter finds the appropriate production, *"recognition"*, and then executes the RHS, taking *"action."*

Conflict Resolution.

A conflict resolution strategy is incorporated in the rule interpreter to choose which rule to fire if more that one rule can be fired in a cycle.

Rule-Based Systems

A rule-based system repeatedly looks for rules whose left-hand sides (LHS) , or condition, evaluates to true. A rule-based system has three main components: a working memory, a rule-base, and a inference engine.

The working memory (WM) is a set of dat structures that represent the current state of the system.

The rule-base is a set of rules, each consisting of a left-hand side (**LHS**) and a right-hand side (**RHS**).

The **LHS** is the condition side of each production rule. If the condition evaluates to true then the condition is satisfied and the rule is fired.

The **RHS** consists of a set of actions that are executed if the rule is fired.

The inference engine applies the rules to the working memory. The inference engine works in a "recognize-act" cycle. The interpreter finds the appropriate production, *"recognition"*, and then executes the RHS, taking *"action."*

Conflict Resolution.

A conflict resolution strategy is incorporated in the rule interpreter to choose which rule to fire if more that one rule can be fired in a cycle.

Semantic Consistency

Semantic consistency requires that the data read in by a knowledge source does not change during the time in which the knowledge source performs its problem-solving activities and places its results onto the blackboard.

Serialization.

The **Serialization Principle** is the requirement that concurrent execution of multiple operations have the same effect as some serial execution of the operations. An execution that satisfies this principle is called serializable.

Appendix A Example Blackboard System Specifications

A.1 Blackboard System Specifications

Specification of the blackboard system B_1:

- $X = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9\}$;

- $P = \{V_1 \times V_2 \times V_3 \times V_4 \times V_5 \times V_6 \times V_7 \times V_8 \times V_9\}$;
  $V_1, .., V_9 = \{\{U\} \cup \Re\}$ U = Undefined, $\Re$ = the set of real numbers

- $\beta = \{ks_1, ks_2, ks_3, ks_4\}$;
  $ks_1 = \{IV = \{d_1, d_2\},$
  $IC = \{ic_1, ic_2\},$
  $F = \{d_3 = \max(d_1, d_2), d_4 = (d_1 * d_2), d_5 = (d_1 / d_2)\}$
  $OV = \{d_3, d_4, d_5\},$
  $PR = \phi,$
  $PT = \phi\}.$

  $ks_2 = \{IV = \{d_3, d_9\},$
  $IC = \{ic_3, ic_9\},$
  $F = \{d_6 = (d_3 / 4.10), d_7 = (d_9 * 0.41)\}$
  $OV = \{d_6, d_7\},$
  $PR = \{pr_1 = (d_4 \neq 7), pr_2 = (d_8 \neq pi)\},$
  $PT = \phi\}.$

  $ks_3 = \{IV = \{d_4, d_6\},$
  $IC = \{ic_4, ic_6\},$
  $F = \{d_8 = \max(d_4, d_6)\},$
  $OV = \{d_8\},$
  $PR = \{pr_2 = (d_8 \neq pi), pr_3 = (d_9 \neq 13.6)\},$
  $PT = \{pt_1 = (d_5 \neq U)\}.$

ks_4 = {IV = {d_5, d_7, d_8},
IC = {ic_5, ic_7, ic_8},
F = {d_9 = (max(d_5, d_7, d_8) / min(d_5, d_7, d_8))}
OV = {d_9},
PR = {pr_3 = ($d_9 \neq 13.6$), pr_4 = ($d_2 \neq 0.0$), pr_5 = ($d_8 \neq 0.0$)},
PT = ϕ.

- I_S = {$d_1 = 1$, $d_2 = 1$, $d_3 = U$, $d_4 = U$, $d_5 = U$, $d_6 = U$, $d_7 = U$, $d_8 = U$, $d_9 = U$}

- Θ ={$\langle ks_1, ks_2\rangle$, $\langle ks_1, ks_3\rangle$, $\langle ks_1, ks_4\rangle$, $\langle ks_2, ks_3\rangle$, $\langle ks_2, ks_4\rangle$, $\langle ks_3, ks_4\rangle$, $\langle ks_4, ks_2\rangle$}

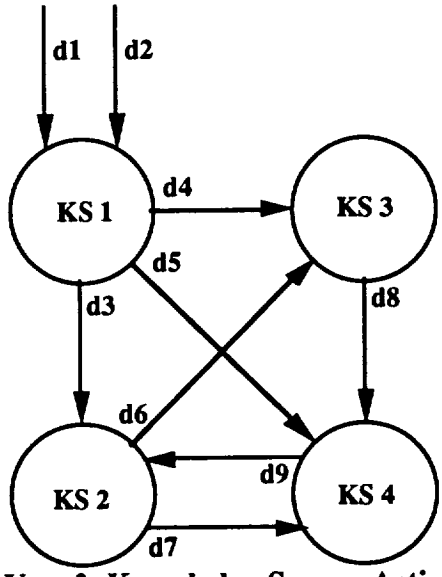

Figure A.1 Unsafe Knowledge Source Activation Graph

The blackboard system B_1 has two cycles. A direct *loop*, or closed path of length two, exists between ks_2 and ks_4. A closed path of length three exists between ks_2, ks_3, and ks_4.

Specification of the blackboard system B_2:

- $X = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9\}$;

- $P = \{V_1 \times V_2 \times V_3 \times V_4 \times V_5 \times V_6 \times V_7 \times V_8 \times V_9\}$;
  $V_1 = \{\{U\} \cup \Re\}$
  $V_2 = \{\{U\} \cup \Re\}$
  $V_3 = \{\{U\} \cup \Re\}$
  $V_4 = \{\{U\} \cup \Re\}$
  $V_5 = \{\{U\} \cup \Re\}$
  $V_6 = \{\{U\} \cup \Re\}$
  $V_7 = \{\{U\} \cup \Re\}$
  $V_8 = \{\{U\} \cup \Re\}$
  $V_9 = \{\{U\} \cup \Re\}$

- $\beta = \{ks_1, ks_2, ks_3, ks_4\}$;
  $ks_1 = \{IV = \{d_1, d_2\},$
  $IC = \{ic_1, ic_2\},$
  $F = \{d_3 = \max(d_1, d_2), d_4 = (d_1 * d_2), d_5 = (d_1 / d_2)\}$
  $OV = \{d_3, d_4, d_5\},$
  $PR = \phi,$
  $PT = \phi\}.$

  $ks_2 = \{IV = \{d_3\},$
  $IC = \{ic_3\},$
  $F = \{d_6 = (d_3 / 4.10), d_7 = (d_3 * 0.41)\}$
  $OV = \{d_6, d_7\},$
  $PR = \{pr_1 = (d_4 \neq 7)\},$
  $PT = \phi\}.$

  $ks_3 = \{IV = \{d_4, d_6\},$
  $IC = \{ic_4, ic_6\},$
  $F = \{d_8 = \max(d_4, d_6)\},$
  $OV = \{d_8\},$
  $PR = \{pr_2 = (d_8 \neq pi), pr_3 = (d_9 \neq 13.6)\},$
  $PT = \{pt_1 = (d_5 \neq U)\}.$

ks_4 = {IV = {d_5, d_7, d_8},
IC = {ic_5, ic_7, ic_8},
F = {d_9 = (max(d_5, d_7, d_8) / min(d_5, d_7, d_8))}
OV = {d_9},
PR = {pr_3 = ($d_9 \neq 13.6$), pr_4 = ($d_2 \neq 0.0$), pr_5 = ($d_8 \neq 0.0$)},
PT = ϕ.

- I_S = {$d_1 = 1$, $d_2 = 1$, $d_3 = U$, $d_4 = U$, $d_5 = U$, $d_6 = U$, $d_7 = U$, $d_8 = U$, $d_9 = U$}

- Θ ={$\langle ks_1, ks_2\rangle$, $\langle ks_1, ks_3\rangle$, $\langle ks_1, ks_4\rangle$, $\langle ks_2, ks_3\rangle$, $\langle ks_2, ks_4\rangle$, $\langle ks_3, ks_4\rangle$}

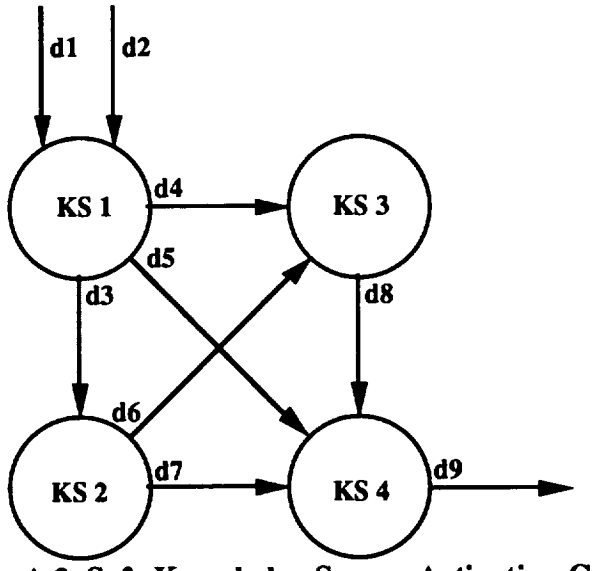

Figure A.2 Safe Knowledge Source Activation Graph

A.2 Blackboard System Execution Traces

The blackboard system B_1 is unsafe, and due to the initial values of the blackboard data objects specified in I_S, the system will deadlock after the execution of ks_1. The blackboard system B_2 is safe and will not deadlock . A trace of the blackboard system execution shows the state of the B_1 from system initialization to deadlock.

At initialization:

- I_S = {$d_1 = 1$, $d_2 = 1$, $d_3 = U$, $d_4 = U$, $d_5 = U$, $d_6 = U$, $d_7 = U$, $d_8 = U$, $d_9 = U$}

- IC = {ic_1 = T, ic_2 = T, ic_3 = F, ic_4 = F, ic_5 = F, ic_6 = F, ic_7 = F, ic_8 = F, ic_9 = F}
- PR = {pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T}
- PT = {pt_1 = F}

After execution of ks_1:

- I_s = {d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = U, d_7 = U, d_8 = U, d_9 = U}
- IC = {ic_1 = F, ic_2 = F, ic_3 = T, ic_4 = T, ic_5 = T, ic_6 = F, ic_7 = F, ic_8 = F, ic_9 = F}
- PR = {pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T}
- PT = {pt_1 = T}

B_1 is now deadlocked. Ks_2 cannot be activated until ks_4 is activated and computes a value for d_9 and sets ic_9 to TRUE. Ks_3 cannot be activated until ks_2 is activated and computes a value for d_6 and sets ic_6 to TRUE. Ks_4 cannot be activated until ks_2 is activated and computes a value for d_7 and sets ic_7 to TRUE, and ks_3 is activated and computes a value for d_8 and sets ic_8 to TRUE. Ks_2 is waiting for ks_4, ks_3 is waiting for ks_2, and ks_4 is waiting for ks_2 and ks_3.

By changing the initial values of the blackboard data objects specified in I_s we can show that fact that a system is unsafe does not imply that the system will deadlock. A trace of the blackboard system execution shows the state of the blackboard system B_1 from system initialization through the completion of the first execution cycle.

At initialization:

- I_s = {d_1 = 1, d_2 = 1, d_3 = U, d_4 = U, d_5 = U, d_6 = U, d_7 = U, d_8 = U, d_9 = 9.774}
- IC = {ic_1 = T, ic_2 = T, ic_3 = F, ic_4 = F, ic_5 = F, ic_6 = F, ic_7 = F, ic_8 = F, ic_9 = T}
- PR = {pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T}

- $PT = \{pt_1 = F\}$

After execution of ks_1:

- $I_s = \{d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = U, d_7 = U, d_8 = U, d_9 = 9.774\}$
- $IC = \{ic_1 = F, ic_2 = F, ic_3 = T, ic_4 = T, ic_5 = T, ic_6 = F, ic_7 = F, ic_8 = F, ic_9 = T\}$
- $PR = \{pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T\}$
- $PT = \{pt_1 = T\}$

Blackboard state after ks_2 is executed:

- $I_s = \{d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 0.24, d_7 = 0.41, d_8 = U, d_9 = 9.774\}$
- $IC = \{ic_1 = F, ic_2 = F, ic_3 = F, ic_4 = T, ic_5 = T, ic_6 = T, ic_7 = T, ic_8 = F, ic_9 = T\}$
- $PR = \{pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T\}$
- $PT = \{pt_1 = T\}$

Blackboard state after ks_3 is executed:

- $I_s = \{d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 0.24, d_7 = 0.41, d_8 = 1, d_9 = 9.774\}$
- $IC = \{ic_1 = F, ic_2 = F, ic_3 = F, ic_4 = F, ic_5 = T, ic_6 = F, ic_7 = T, ic_8 = T, ic_9 = T\}$
- $PR = \{pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T\}$
- $PT = \{pt_1 = T\}$

Blackboard state after ks_4 is executed:

- $I_s = \{d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 0.24, d_7 = 0.41, d_8 = 1, d_9 = 2.439\}$
- $IC = \{ic_1 = F, ic_2 = F, ic_3 = F, ic_4 = F, ic_5 = F, ic_6 = F, ic_7 = F, ic_8 = F, ic_9 = T\}$
- $PR = \{pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T\}$
- $PT = \{pt_1 = T\}$

B_1 has completed the first path through the execution cycle and has not deadlocked. The system will continue to execute safely as long as data arrives at the sensor, ks_1. It is important to realize that it is the initial state of the blackboard data objects that can cause an unsafe system to deadlock. An unsafe system design does not insure that a system will deadlock, the initial state of the blackboard data elements may or may not cause deadlock.

A trace of the blackboard system execution shows the state of the blackboard system B_2 from system initialization through the completion of the first execution cycle. B_2 is a safe blackboard system and does not deadlock.

At initialization:

- $I_s = \{d_1 = 1, d_2 = 1, d_3 = U, d_4 = U, d_5 = U, d_6 = U, d_7 = U, d_8 = U, d_9 = U\}$
- $IC = \{ic_1 = T, ic_2 = T, ic_3 = F, ic_4 = F, ic_5 = F, ic_6 = F, ic_7 = F, ic_8 = F, ic_9 = F\}$
- $PR = \{pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T\}$
- $PT = \{pt_1 = F\}$

After execution of ks_1:

- $I_s = \{d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = U, d_7 = U, d_8 = U, d_9 = U\}$

- IC = {ic_1 = F, ic_2 = F, ic_3 = T, ic_4 = T, ic_5 = T, ic_6 = F, ic_7 = F, ic_8 = F, ic_9 = F}
- PR = {pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T}
- PT = {pt_1 = T}

Blackboard state after ks_2 is executed:

- I_s = {d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 0.24, d_7 = 0.41, d_8 = U, d_9 = U}
- IC = {ic_1 = F, ic_2 = F, ic_3 = F, ic_4 = T, ic_5 = T, ic_6 = T, ic_7 = T, ic_8 = F, ic_9 = F}
- PR = {pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T}
- PT = {pt_1 = T}

Blackboard state after ks_3 is executed:

- I_s = {d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 0.24, d_7 = 0.41, d_8 = 1, d_9 = U}
- IC = {ic_1 = F, ic_2 = F, ic_3 = F, ic_4 = F, ic_5 = T, ic_6 = F, ic_7 = T, ic_8 = T, ic_9 = F}
- PR = {pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T}
- PT = {pt_1 = T}

Blackboard state after ks_4 is executed:

- I_s = {d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 0.24, d_7 = 0.41, d_8 = 1, d_9 = 2.439}
- IC = {ic_1 = F, ic_2 = F, ic_3 = F, ic_4 = F, ic_5 = F, ic_6 = F, ic_7 = F, ic_8 = F, ic_9 = T}
- PR = {pr_1 = T, pr_2 = T, pr_3 = T, pr_4 = T, pr_5 = T}

- $PT = \{pt_1 = T\}$

B_2 has completed the first loop through the execution cycle and has not deadlocked. The system will continue to execute safely as long as data arrives at the sensor, ks_1.

Appendix B Example Connectivity Analysis Results

B.1 Examples of Connectivity Analysis

This section demonstrates the application of the connectivity analysis techniques using the blackboard systems defined in Chapter Three as examples.

B.1.1 Blackboard System B_1 Analysis Results

Knowledge source: KS1
Type: SENSOR
Ψ: (D1 D2)
Φ: (D3 D4 D5)
Input conditionals: (KS1-D2 KS1-D1)
Preconditions: ϕ
Postconditions: ϕ
Depth: 0
Predecessors: ϕ
Successors: (KS4 KS3 KS2)
Execution time: 5
Cardinality of Ψ: 2
Cardinality of Φ: 3

Output Overlap
$\Gamma<KS_1KS_4>$ ϕ
$\Gamma<KS_1KS_3>$ ϕ
$\Gamma<KS_1KS_2>$ ϕ

Specialization Values
$\Omega<KS_1KS_4>$ 0.0
$\Omega<KS_1KS_3>$ 0.0
$\Omega<KS_1KS_2>$ 0.0

Output to Input Connectivity
$\Lambda<KS_1KS_4>$ (D5)
$\Lambda<KS_1KS_3>$ (D4)
$\Lambda<KS_1KS_2>$ (D3)

Interdependence Values
$\Pi<KS_1KS_4>$ 0.33333334
$\Pi<KS_1KS_3>$ 0.33333334
$\Pi<KS_1KS_2>$ 0.33333334

Serialization Value
$\Sigma<KS_1KS_4>$ 0.33333334
$\Sigma<KS_1KS_3>$ 0.5

$\Sigma<KS_1KS_2>$ 0.5

Knowledge source: KS2
Type: PROCESSOR
Ψ: (D3 D9)
Φ: (D6 D7)
Input conditionals: (KS2-D9 KS2-D3)
Preconditions: ((D4 $\neq$ 7)
(D8 $\neq$ PI))
Postconditions: ϕ
Depth: 1
Predecessors: (KS4 KS1)
Successors: (KS4 KS3)
Execution time: 4
Cardinality of Ψ: 2
Cardinality of Φ: 2

Output Overlap
$\Gamma<KS_2KS_4>$ ϕ
$\Gamma<KS_2KS_3>$ ϕ
$\Gamma<KS_2KS_1>$ ϕ

Specialization Values
$\Omega<KS_2KS_4>$ 0.0
$\Omega<KS_2KS_3>$ 0.0
$\Omega<KS_2KS_1>$ 0.0

Output to Input Connectivity
$\Lambda<KS_2KS_4>$ (D7)
$\Lambda<KS_2KS_3>$ (D6)
$\Lambda<KS_2KS_1>$ ϕ

Interdependence Values
$\Pi<KS_2KS_4>$ 0.5
$\Pi<KS_2KS_3>$ 0.5
$\Pi<KS_2KS_1>$ 0.0

Serialization Value
$\Sigma<KS_2KS_4>$ 0.33333334
$\Sigma<KS_2KS_3>$ 0.5
$\Sigma<KS_2KS_1>$ 0.0

Knowledge source: KS3
Type: PROCESSOR
Ψ: (D4 D6)
Φ: (D8)
Input conditionals: (KS3-D6 KS3-D4)
Preconditions: ((D8 $\neq$ PI)
(D9 $\neq$ 13.6))
Postconditions: ((D5 $\neq$ UNDEFINED))

Depth: 2
Predecessors: (KS2 KS1)
Successors: (KS4)
Execution time: 6
Cardinality of Ψ: 2
Cardinality of Φ: 1

Output Overlap
$\Gamma<KS_3KS_4>$ ϕ
$\Gamma<KS_3KS_2>$ ϕ
$\Gamma<KS_3KS_1>$ ϕ

Specialization Values
$\Omega<KS_3KS_4>$ 0.0
$\Omega<KS_3KS_2>$ 0.0
$\Omega<KS_3KS_1>$ 0.0

Output to Input Connectivity
$\Lambda<KS_3KS_4>$ (D8)
$\Lambda<KS_3KS_2>$ ϕ
$\Lambda<KS_3KS_1>$ ϕ

Interdependence Values
$\Pi<KS_3KS_4>$ 1.0
$\Pi<KS_3KS_2>$ 0.0
$\Pi<KS_3KS_1>$ 0.0

Serialization Value
$\Sigma<KS_3KS_4>$ 0.33333334
$\Sigma<KS_3KS_2>$ 0.0
$\Sigma<KS_3KS_1>$ 0.0

Knowledge source: KS4
Type: PROCESSOR
Ψ: (D5 D7 D8)
Φ: (D9)
Input conditionals: (KS4-D8 KS4-D7 KS4-D5)
Preconditions: ((D9 ≠ 13.6)
(D2 ≠ 0.0)
(D8 ≠ 0.0))
Postconditions: ϕ
Depth: 3
Predecessors: (KS3 KS2 KS1)
Successors: (KS2)
Execution time: 8
Cardinality of Ψ: 3
Cardinality of Φ: 1

Output Overlap
$\Gamma<KS_4KS_3>$ ϕ

$\Gamma<KS_4KS_2>$ ϕ
$\Gamma<KS_4KS_1>$ ϕ

Specialization Values
$\Omega<KS_4KS_3>$ 0.0
$\Omega<KS_4KS_2>$ 0.0
$\Omega<KS_4KS_1>$ 0.0

Output to Input Connectivity
$\Lambda<KS_4KS_3>$ ϕ
$\Lambda<KS_4KS_2>$ (D9)
$\Lambda<KS_4KS_1>$ ϕ

Interdependence Values
$\Pi<KS_4KS_3>$ 0.0
$\Pi<KS_4KS_2>$ 1.0
$\Pi<KS_4KS_1>$ 0.0

Serialization Value
$\Sigma<KS_4KS_3>$ 0.0
$\Sigma<KS_4KS_2>$ 0.5
$\Sigma<KS_4KS_1>$ 0.0

The blackboard system B_1 has two cycles. A direct *loop*, or closed path of length two, exists between ks_2 and ks_4. A closed path of length three exists between ks_2, ks_3, and ks_4.

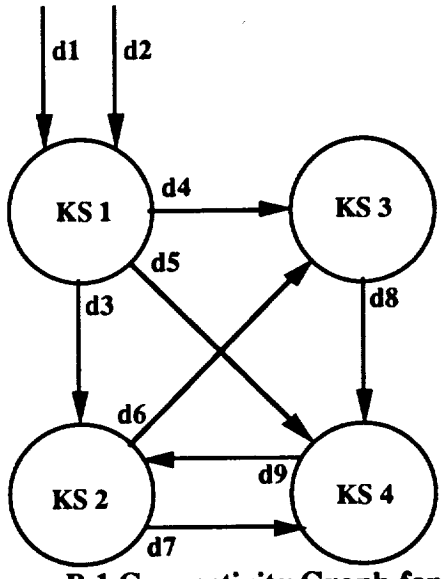

Figure B.1 Connectivity Graph for B_1

B.1.2 Blackboard System B_2 Analysis Results

Knowledge source: KS1

Type: SENSOR
Ψ: (D1 D2)
Φ: (D3 D4 D5)
Input conditionals: (KS1-D2 KS1-D1)
Preconditions: ϕ
Postconditions: ϕ
Depth: 0
Predecessors: ϕ
Successors: (KS4 KS3 KS2)
Execution time: 5
Cardinality of Ψ: 2
Cardinality of Φ: 3

Output Overlap
$\Gamma<KS_1KS_2>$ ϕ
$\Gamma<KS_1KS_3>$ ϕ
$\Gamma<KS_1KS_4>$ ϕ

Specialization Values
$\Omega<KS_1KS_2>$ 0.0
$\Omega<KS_1KS_3>$ 0.0
$\Omega<KS_1KS_4>$ 0.0

Output to Input Connectivity
Λ<KS1KS2> (D3)
Λ<KS1KS3> (D4)
Λ<KS1KS4> (D5)

Interdependence Values
$\Pi<KS_1KS_2>$ 0.33333334
$\Pi<KS_1KS_3>$ 0.33333334
$\Pi<KS_1KS_4>$ 0.33333334

Serialization Value
$\Sigma<KS_1KS_2>$ 1.0
$\Sigma<KS_1KS_3>$ 0.5
$\Sigma<KS_1KS_4>$ 0.33333334

Knowledge source: KS2
Type: PROCESSOR
Ψ: (D3)
Φ: (D6 D7)
Input conditionals: (KS2-D9 KS2-D3)
Preconditions: (D4 $\neq$ 7)
(D8 $\neq$ PI))
Postconditions: ϕ
Depth: 1
Predecessors: (KS4 KS1)
Successors: (KS4 KS3)
Execution time: 4

Cardinality of Ψ: 1
Cardinality of Φ: 2

Output Overlap
$\Gamma<KS_2KS_1>$ ϕ
$\Gamma<KS_2KS_3>$ ϕ
$\Gamma<KS_2KS_4>$ ϕ

Specialization Values
$\Omega<KS_2KS_1>$ 0.0
$\Omega<KS_2KS_3>$ 0.0
$\Omega<KS_2KS_4>$ 0.0

Output to Input Connectivity
Λ<KS2KS1> ϕ
Λ<KS2KS3> (D6)
Λ<KS2KS4> (D7)

Interdependence Values
$\Pi<KS_2KS_1>$ 0.0
$\Pi<KS_2KS_3>$ 0.5
$\Pi<KS_2KS_4>$ 0.5

Serialization Value
$\Sigma<KS_2KS_1>$ 0.0
$\Sigma<KS_2KS_3>$ 0.5
$\Sigma<KS_2KS_4>$ 0.33333334

Knowledge source: KS3
Type: PROCESSOR
Ψ: (D4 D6)
Φ: (D8)
Input conditionals: (KS3-D6 KS3-D4)
Preconditions: (D8 ≠ PI)
(D9 ≠ 13.6))
Postconditions: (D5 ≠ UNDEFINED)))
Depth: 2
Predecessors: (KS2 KS1)
Successors: (KS4)
Execution time: 6
Cardinality of Ψ: 2
Cardinality of Φ: 1

Output Overlap
$\Gamma<KS_3KS_1>$ ϕ
$\Gamma<KS_3KS_2>$ ϕ
$\Gamma<KS_3KS_4>$ ϕ

Specialization Values
$\Omega<KS_3KS_1>$ 0.0

$\Omega<KS_3KS_2>$ 0.0
$\Omega<KS_3KS_4>$ 0.0

Output to Input Connectivity
Λ<KS3KS1> ϕ
Λ<KS3KS2> ϕ
Λ<KS3KS4> (D8)

Interdependence Values
$\Pi<KS_3KS_1>$ 0.0
$\Pi<KS_3KS_2>$ 0.0
$\Pi<KS_3KS_4>$ 1.0

Serialization Value
$\Sigma<KS_3KS_1>$ 0.0
$\Sigma<KS_3KS_2>$ 0.0
$\Sigma<KS_3KS_4>$ 0.33333334

Knowledge source: KS4
Type: PROCESSOR
Ψ: (D5 D7 D8)
Φ: (D9)
Input conditionals: (KS4-D8 KS4-D7 KS4-D5)
Preconditions: (D9 $\neq$ 13.6)
(D2 $\neq$ 0.0)
(D8 $\neq$ 0.0))
Postconditions: ϕ
Depth: 3
Predecessors: (KS3 KS2 KS1)
Successors: (KS2)
Execution time: 8
Cardinality of Ψ: 3
Cardinality of Φ: 1

Output Overlap
$\Gamma<KS_4KS_1>$ ϕ
$\Gamma<KS_4KS_2>$ ϕ
$\Gamma<KS_4KS_3>$ ϕ

Output to Input Connectivity
Λ<KS4KS1> ϕ
Λ<KS4KS2> ϕ
Λ<KS4KS3> ϕ

Specialization Values
$\Omega<KS_4KS_1>$ 0.0
$\Omega<KS_4KS_2>$ 0.0
$\Omega<KS_4KS_3>$ 0.0

Interdependence Values

$\Pi<KS_4KS_1>$ 0.0
$\Pi<KS_4KS_2>$ 0.0
$\Pi<KS_4KS_3>$ 0.0

Serialization Value
$\Sigma<KS_4KS_1>$ 0.0
$\Sigma<KS_4KS_2>$ 0.0
$\Sigma<KS_4KS_3>$ 0.0

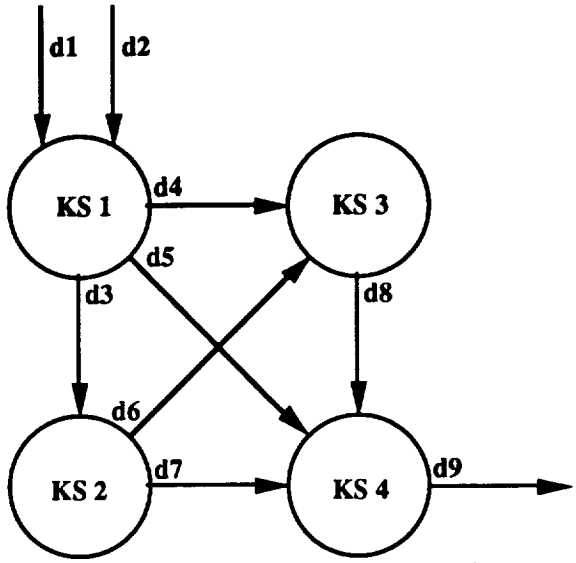

Figure B.2 Connectivity Graph for B_2

B.1.3 Blackboard System B_3 Analysis Results

Knowledge source: ks1
Type: sensor
Ψ: (D1 D2)
Φ: (D3 D4 D5 D6)
Input conditionals: (ks1-D2 ks1-D1)
Preconditions: ϕ
Postconditions: ϕ
Depth: 0
Predecessors: ϕ
Successors: (ks2 ks3 ks4)
Execution time: 4
Cardinality of Ψ: 2
Cardinality of Φ: 4

Output Overlap
$\Gamma<ks_1ks_2>$ ϕ
$\Gamma<ks_1ks_3>$ ϕ
$\Gamma<ks_1ks_4>$ ϕ

Specialization Values

$\Omega<ks_1 ks2>$ 0.0
$\Omega<ks_1 ks3>$ 0.0
$\Omega<ks_1 ks_4>$ 0.0

Output to Input Connectivity
$\Lambda<ks_1 ks_2>$ (D3)
$\Lambda<ks_1 ks_3>$ (D4 D6)
$\Lambda<ks_1 ks_4>$ (D5)

Interdependence Values
$\Pi<ks_1 ks2>$ 0.25
$\Pi<ks_1 ks3>$ 0.5
$\Pi<ks_1 ks4>$ 0.25

Serialization Value
$\Sigma<ks_1 ks_2>$ 1.0
$\Sigma<ks_1 ks_3>$ 1.0
$\Sigma<ks_1 ks_4>$ 0.33333334

Knowledge source: ks2
Type: processor
Ψ: (D3)
Φ: (D7)
Input conditionals: (ks2-D3)
Preconditions: ((D4 $\neq$ 7))
Postconditions: ϕ
Depth: 1
Predecessors: (ks1)
Successors: (ks4)
Execution time: 3
Cardinality of Ψ: 1
Cardinality of Φ: 1

Output Overlap
$\Gamma<ks_2 ks_1>$ ϕ
$\Gamma<ks_2 ks_3>$ ϕ
$\Gamma<ks_2 ks_4>$ ϕ

Specialization Values
$\Omega<ks_2 ks_1>$ 0.0
$\Omega<ks_2 ks_3>$ 0.0
$\Omega<ks_2 ks_4>$ 0.0

Output to Input Connectivity
$\Lambda<ks_2 ks_1>$ ϕ
$\Lambda<ks_2 ks_3>$ ϕ
$\Lambda<ks_2 ks_4>$ (D7)

Interdependence Values
$\Pi<ks_2 ks_1>$ 0.0

$\Pi<ks_2ks_3>$ 0.0
$\Pi<ks_2ks_4>$ 1.0

Serialization Value
$\Sigma<ks_2ks_1>$ 0.0
$\Sigma<ks_2ks_3>$ 0.0
$\Sigma<ks_2ks_4>$ 0.33333334

Knowledge source: ks3
Type: processor
Ψ: (D4 D6)
Φ: (D8)
Input conditionals: (ks3-D6 ks3-D4)
Preconditions: ((D8 $\neq$ PI)
(D9 $\neq$ 13.6))
Postconditions: ((D5 $\neq$ UNDEFINED))
Depth: 2
Predecessors: (ks1)
Successors: (ks4)
Execution time: 5
Cardinality of Ψ: 2
Cardinality of Φ: 1

Output Overlap
$\Gamma<ks_3ks_1>$ ϕ
$\Gamma<ks_3ks_2>$ ϕ
$\Gamma<ks_3ks_4>$ ϕ

Specialization Values
$\Omega<ks_3ks_1>$ 0.0
$\Omega<ks_3ks_2>$ 0.0
$\Omega<ks_3ks_4>$ 0.0

Output to Input Connectivity
$\Lambda<ks_3ks_1>$ ϕ
$\Lambda<ks_3ks_2>$ ϕ
$\Lambda<ks_3ks_4>$ (D8)

Interdependence Values
$\Pi<ks_3ks_1>$ 0.0
$\Pi<ks_3ks_2>$ 0.0
$\Pi<ks_3ks_4>$ 1.0

Serialization Value
$\Sigma<ks_3ks_1>$ 0.0
$\Sigma<ks_3ks_2>$ 0.0
$\Sigma<ks_3ks_4>$ 0.33333334

Knowledge source: ks4
Type: processor

Ψ: (D5 D7 D8)
Φ: (D9)
Input conditionals: (ks4-D8 ks4-D7 ks4-D5)
Preconditions: ((D9 $\neq$ 13.6)
(D2 $\neq$ 0.0)
(D8 $\neq$ 0.0))
Postconditions: f
Depth: 3
Predecessors: (ks1 ks2 ks3)
Successors: ϕ
Execution time: 3
Cardinality of Ψ: 3
Cardinality of Φ: 1

Output Overlap
Γ<ks_4ks_1> ϕ
Γ<ks_4ks_2> ϕ
Γ<ks_4ks_3> ϕ

Specialization Values
Ω<ks_4ks_1> 0.0
Ω<ks_4ks_2> 0.0
Ω<ks_4ks_3> 0.0

Output to Input Connectivity
Λ<ks_4ks_1> ϕ
Λ<ks_4ks_2> ϕ
Λ<ks_4ks_3> ϕ

Interdependence Values
Π<ks_4ks_1> 0.0
Π<ks_4ks_2> 0.0
Π<ks_4ks_3> 0.0

Serialization Value
Σ<ks_4ks_1> 0.0
Σ<ks_4ks_2> 0.0
Σ<ks_4ks_3> 0.0

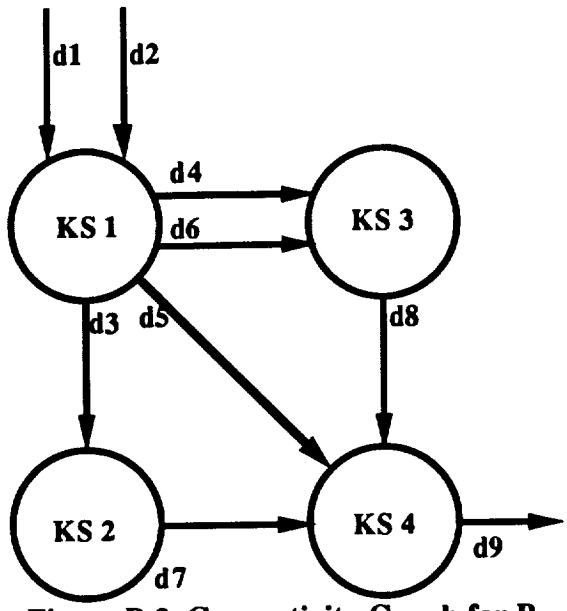

Figure B.3 Connectivity Graph for B_3

B.1.4 Blackboard System B_4 Analysis Results

Knowledge source: ks1
- Type: sensor
- Ψ: (D1 D2)
- Φ: (D3 D4 D5 D6)
- Input conditionals: (ks1-D2 ks1-D1)
- Preconditions: ϕ
- Postconditions: ϕ
- Depth: 0
- Predecessors: ϕ
- Successors: (ks2 ks3)
- Execution time: 4
- Cardinality of Ψ: 2
- Cardinality of Φ: 4

Output Overlap
- $\Gamma<ks_1 ks_2>$ ϕ
- $\Gamma<ks_1 ks_3>$ ϕ

Specialization Values
- $\Omega<ks_1 ks_2>$ 0.0
- $\Omega<ks_1 ks_3>$ 0.0

Output to Input Connectivity
- $\Lambda<ks_1 ks_2>$ (D4 D6)
- $\Lambda<ks_1 ks_3>$ (D3 D5)

Interdependence Values
- $\Pi<ks_1 ks_2>$ 0.5

$\Pi<ks_1 ks_3>$ 0.5

Serialization Value
$\Sigma<ks1 ks_2>$ 1.0
$\Sigma<ks1 ks_3>$ 0.6666667

Knowledge source: ks2
Type: processor
Ψ: (D4 D6)
Φ: (D8)
Input conditionals: (ks2-D6 ks2-D4)
Preconditions: ((D8 ≠ PI)
(D9 ≠ 13.6))
Postconditions: ((D5 ≠ UNDEFINED))
Depth: 1
Predecessors: (ks1)
Successors: (ks3)
Execution time: 4
Cardinality of Ψ: 2
Cardinality of Φ: 1

Output Overlap
$\Gamma<ks_2 ks_1>$ ϕ
$\Gamma<ks_2 ks_3>$ ϕ

Specialization Values
$\Omega<ks_2 ks_1>$ 0.0
$\Omega<ks_2 ks_3>$ 0.0

Output to Input Connectivity
$\Lambda<ks_2 ks_1>$ ϕ
$\Lambda<ks_2 ks_3>$ (D8)

Interdependence Values
$\Pi<ks_2 ks_1>$ 0.0
$\Pi<ks_2 ks_3>$ 1.0

Serialization Value
$\Sigma<ks2 ks_1>$ 0.0
$\Sigma<ks2 ks_3>$ 0.33333334

Knowledge source: ks3
Type: processor
Ψ: (D3 D5 D8)
Φ: (D9)
Input conditionals: (ks3-D8 ks3-D5 ks3-D3)
Preconditions: ((D4 ≠ 7)
(D2 ≠ 0.0)
(D8 ≠ 0.0)
(D9 ≠ 13.6))

Postconditions: ϕ
Depth: 2
Predecessors: (ks1 ks2)
Successors: ϕ
Execution time: 6
Cardinality of Ψ: 3
Cardinality of Φ: 1

Output Overlap
$\Gamma<ks_3 ks_1> \phi$
$\Gamma<ks_3 ks_2> \phi$

Specialization Values
$\Omega<ks_3 ks_1> 0.0$
$\Omega<ks_3 ks_2> 0.0$

Output to Input Connectivity
$\Lambda<ks_3 ks_1> \phi$
$\Lambda<ks_3 ks_2> \phi$

Interdependence Values
$\Pi<ks_3 ks_1> 0.0$
$\Pi<ks_3 ks_2> 0.0$

Serialization Value
$\Sigma<ks3 ks_1> 0.0$
$\Sigma<ks3 ks_2> 0.0$

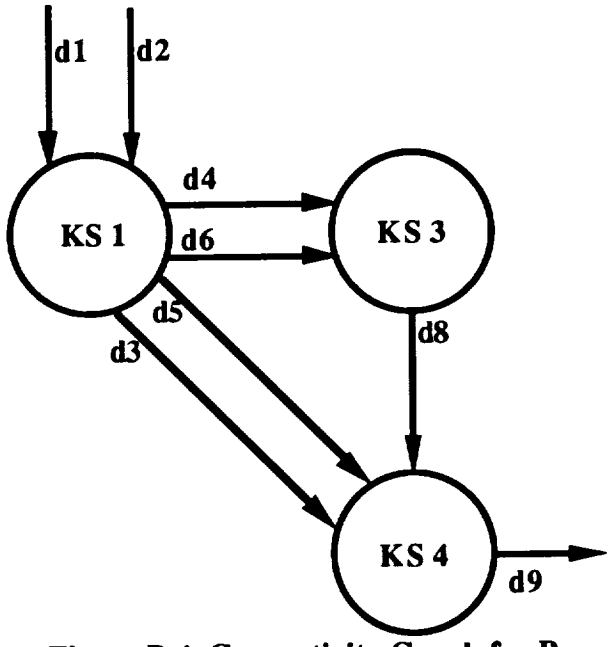

Figure B.4 Connectivity Graph for B_4

B.2 Example of a Simulation Run

The safe blackboard system B_2, described in Chapter Three, is used as an example.

B.2.1 Blackboard System Specification

Specification of the blackboard system B_2. :

$\Sigma =$

- $X = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9\}$;

- $P = \{V_1 \times V_2 \times V_3 \times V_4 \times V_5 \times V_6 \times V_7 \times V_8 \times V_9\}$;
  $V_1 = \{\{U\} \cup \Re\}$
  $V_2 = \{\{U\} \cup \Re\}$
  $V_3 = \{\{U\} \cup \Re\}$
  $V_4 = \{\{U\} \cup \Re\}$
  $V_5 = \{\{U\} \cup \Re\}$
  $V_6 = \{\{U\} \cup \Re\}$
  $V_7 = \{\{U\} \cup \Re\}$
  $V_8 = \{\{U\} \cup \Re\}$
  $V_9 = \{\{U\} \cup \Re\}$

- $\beta = \{ks_1, ks_2, ks_3, ks_4\}$;
  $ks_1 = \{IV = \{d_1, d_2\},$
  $IC = \{ic_1, ic_2\},$
  $UR = \{5 \ldots 135\},$
  $AT = UR,$
  $XD = \{5 \ldots 75\},$
  $CT = 0,$
  $F = \{d_3 = \max(d_1, d_2), d_4 = (d_1 * d_2), d_5 = (d_1 / d_2)\}$
  $OV = \{d_3, d_4, d_5\},$
  $PR = \phi,$
  $PT = \phi\}.$

ks_2 = {IV = {d_3},
IC = {ic_3},
XD = {15 ... 31},
CT = 0,
F = {$d_6 = (d_3 / 4.10)$, $d_7 = (d_3 * 0.41)$}
OV = {d_6, d_7},
PR = {$pr_1 = (d_4 \neq 7)$},
PT = ϕ}.

ks_3 = {IV = {d_4, d_6},
IC = {ic_4, ic_6},
XD = {25 ... 35},
CT = 0,
F = {$d_8 = \max(d_4,d_6)$},
OV = {d_8},
PR = {$pr_2 = (d_8 \neq pi)$, $pr_3 = (d_9 \neq 13.6)$},
PT = {$pt_1 = (d_5 \neq U)$}.

ks_4 = {IV = {d_5 ,d_7 ,d_8},
IC = {ic_5, ic_7, ic_8},
XD = {5 ... 55},
CT = 0,
F = {$d_9 = (\max(d_5, d_7, d_8) / \min(d_5, d_7, d_8))$}
OV = {d_9},
PR = {$pr_3 = (d_9 \neq 13.6)$, $pr_4 = (d_2 \neq 0.0)$, $pr_5 = (d_8 \neq 0.0)$},
PT = ϕ.

- I_S = {$d_1 = 1$, $d_2 = 1$, $d_3 = U$, $d_4 = U$, $d_5 = U$, $d_6 = U$, $d_7 = U$, $d_8 = U$, $d_9 = U$}

B.2.2 Blackboard System Simulation

At Initialization:

All sensors are evaluated. Sensor ks_1 is ready to execute (ic_1 = TRUE, ic_2 = TRUE). An execution delay and a completion time are computed for ks_1.

XD = 62

$$CT = \Delta + XD,\ CT = 23 + 62,\ CT = 85 \tag{B.1}$$

Ks_1 performs a read operation to read its input variables and is pushed onto the queue of executable knowledge sources. Δ is updated to the activation time of ks_1. Ks_1's input conditionals are reset to false. A new update rate and activation time are computed for sensor ks_1.

$$UR = 23$$
$$AT = UR + \Delta\ (\Delta = 0) = 23 \tag{B.2}$$

All knowledge sources are checked and none are ready to be activated.

$\Delta = 23$

$ks_1 = \{IV = \{d_1 = 1, d_2 = 1\},$
$IC = \{ic_1 = FALSE, ic_2 = FALSE\},$
$UR = \{23\},$
$AT = 23,$
$XD = \{62\},$
$CT = 85,$
$F = \{d_3 = \max(d_1, d_2), d_4 = (d_1 * d_2), d_5 = (d_1 / d_2)\}$
$OV = \{d_3, d_4, d_5\},$
$PR = \phi,$
$PT = \phi\}.$

$ks_2 = \{IV = \{d_3 = U\},$
$IC = \{ic_3 = FALSE\},$
$XD = \{15 \ldots 31\},$
$CT = 0,$
$F = \{d_6 = (d_3 / 4.10), d_7 = (d_3 * 0.41)\}$
$OV = \{d_6, d_7\},$
$PR = \{pr_1 = (d_4 \neq 7)\},$
$PT = \phi\}.$

ks_3 = {IV = {d_4 = U, d_6 = U},
IC = {ic_4 = FALSE, ic_6 = FALSE},
XD = {25 ... 35},
CT = 0,
F = {d_8 = max(d_4,d_6)},
OV = {d_8},
PR = {pr_2 = (d_8 ≠ pi), pr_3 = (d_9 ≠ 13.6)},
PT = {pt_1 = (d_5 ≠ U)}.

ks_4 = {IV = {d_5 = U, d_7 = U, d_8 = U},
IC = {ic_5 = FALSE, ic_7= FALSE, ic_8= FALSE},
XD = {5 ... 55},
CT = 0,
F = {d_9 = (max(d_5, d_7, d_8) / min(d_5, d_7, d_8))}
OV = {d_9},
PR = {pr_3 = (d_9 ≠ 13.6), pr_4 = (d_2 ≠ 0.0), pr_5 = (d_8 ≠ 0.0)},
PT = ϕ.

- X = {d_1 = 1, d_2 = 1, d_3 = U, d_4 = U, d_5 = U, d_6 = U, d_7 = U, d_8 = U, d_9 = U}

- A = {ks_1}

First Simulation Loop:

Ks_1 is removed from the event queue, Δ is set to the completion time of ks_1 and the function F is computed. Ks_1's output variables are updated on the blackboard.

$Ks_{2's}$ input conditionals and preconditions are TRUE, so ks_2 is activated. Ks_2 computes an execution delay and a completion time.

$$XD = 22$$
$$CT = \Delta + XD,\ CT = 85 + 22,\ CT = 107 \qquad (B.3)$$

Ks_2 performs a read operation to read its input variables and the input conditionals are reset to FALSE. Ks_2 is placed on the event queue.

Since ks_1 is a sensor new values for UR and AT are computed.

$$UR = 133$$
$$AT = UR + \Delta\ (\Delta = 85) = 218 \qquad (B.4)$$

All sensors are tested and none will be activated before the completion of ks_2.

$\Delta = 85$

$ks_1 = \{IV = \{d_1 = 1, d_2 = 1\},$
$IC = \{ic_1 = FALSE, ic_2 = FALSE\},$
$UR = \{133\},$
$AT = 218,$
$XD = \{0\},$
$CT = 0,$
$F = \{d_3 = \max(d_1, d_2), d_4 = (d_1 * d_2), d_5 = (d_1 / d_2)\}$
$OV = \{d_3, = 1\ d_4 = 1, d_5 = 1\},$
$PR = \phi,$
$PT = \phi\}.$

$ks_2 = \{IV = \{d_3 = 1\},$
$IC = \{ic_3 = FALSE\},$
$XD = \{22\},$
$CT = 107,$
$F = \{d_6 = (d_3 / 4.10), d_7 = (d_3 * 0.41)\}$
$OV = \{d_6, d_7\},$
$PR = \{pr_1 = (d_4 \neq 7)\},$
$PT = \phi\}.$

$ks_3 = \{IV = \{d_4 = 1, d_6 = U\},$
$IC = \{ic_4 = TRUE, ic_6 = FALSE\},$
$XD = \{25 \ldots 35\},$
$CT = 0,$
$F = \{d_8 = \max(d_4, d_6)\},$
$OV = \{d_8\},$
$PR = \{pr_2 = (d_8 \neq pi), pr_3 = (d_9 \neq 13.6)\},$
$PT = \{pt_1 = (d_5 \neq U)\}.$

ks_4 = {IV = {$d_5 = 1$, $d_7 = U$, $d_8 = U$},
IC = {ic_5 = TRUE, ic_7= FALSE, ic_8= FALSE},
XD = {5 ... 55},
CT = 0,
F = {$d_9 = (\max(d_5, d_7, d_8) / \min(d_5, d_7, d_8))$}
OV = {d_9},
PR = {$pr_3 = (d_9 \neq 13.6)$, $pr_4 = (d_2 \neq 0.0)$, $pr_5 = (d_8 \neq 0.0)$},
PT = ϕ.

- X = {$d_1 = 1$, $d_2 = 1$, $d_3 = 1$, $d_4 = 1$, $d_5 = 1$, $d_6 = U$, $d_7 = U$, $d_8 = U$, $d_9 = U$}
- A = {ks_2}

Second Simulation Loop:

Ks_2 is removed from the event queue. Δ is set to the completion time of ks_2 and the function F is computed. Ks_2's output variables are updated on the blackboard.

Ks_3's input conditionals and preconditions are TRUE, so ks_3 is activated. Ks_3 computes an execution delay and a completion time.

$$XD = 27$$
$$CT = \Delta + XD,\ CT = 107 + 27,\ CT = 134 \qquad \text{(B.5)}$$

Ks_3 performs a read operation to read its input variables and the input conditionals are reset to FALSE. Ks_3 is placed on the event queue. All sensors are tested and none will be activated before the completion of ks_3.

$$\Delta = 107$$

ks_1 = {IV = {$d_1 = 1, d_2 = 1$},
IC = {ic_1 = FALSE, ic_2 = FALSE},
UR = {23},
AT = 23,
XD = {62},
CT = 85,
F = {$d_3 = \max(d_1, d_2)$, $d_4 = (d_1 * d_2)$, $d_5 = (d_1 / d_2)$}
OV = {d_3, = 1 $d_4 = 1$, $d_5 = 1$},
PR = ϕ,
PT = ϕ}.

ks_2 = {IV = {$d_3 = 1$},
IC = {ic_3 = FALSE},
XD = {22},
CT = 107,
F = {$d_6 = (d_3 / 4.10)$, $d_7 = (d_3 * 0.41)$}
OV = {$d_6 = 2.43$, $d_7 = 0.41$},
PR = {$pr_1 = (d_4 \neq 7)$},
PT = ϕ}.

ks_3 = {IV = {$d_4 = 1$, $d_6 = 2.43$},
IC = {ic_4 = FALSE, ic_6 = FALSE},
XD = {27},
CT = 134,
F = {$d_8 = \max(d_4, d_6)$},
OV = {d_8},
PR = {$pr_2 = (d_8 \neq pi)$, $pr_3 = (d_9 \neq 13.6)$},
PT = {$pt_1 = (d_5 \neq U)$}.

ks_4 = {IV = {$d_5 = 1$, $d_7 = 0.41$, d_8 = U},
IC = {ic_5 = TRUE, ic_7= TRUE, ic_8= FALSE},
XD = {5 ... 55},
CT = 0,
F = {$d_9 = (\max(d_5, d_7, d_8) / \min(d_5, d_7, d_8))$}
OV = {d_9},
PR = {$pr_3 = (d_9 \neq 13.6)$, $pr_4 = (d_2 \neq 0.0)$, $pr_5 = (d_8 \neq 0.0)$},
PT = ϕ.

C3

- $X = \{d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 2.43, d_7 = 0.41, d_8 = U, d_9 = U\}$
- $A = \{ks_3\}$

Third Simulation Loop:

Ks_3 is removed from the event queue. Δ is set to the completion time of ks_3 and the function F is computed. Ks_3's output variables are updated on the blackboard.

Ks_4's input conditionals and preconditions are TRUE, so ks_4 is activated. Ks_4 computes an execution delay and a completion time.

$$XD = 52$$
$$CT = \Delta + XD,\ CT = 134 + 52,\ CT = 186 \qquad \text{(B.6)}$$

Ks_4 performs a read operation to read its input variables and the input conditionals are reset to FALSE. Ks_4 is placed on the event queue. All sensors are tested and none will be activated before the completion of ks_4.

$\Delta = 134$
$ks_1 = \{IV = \{d_1 = 1, d_2 = 1\},$
$IC = \{ic_1 = FALSE, ic_2 = FALSE\},$
$UR = \{133\},$
$AT = 218,$
$XD = \{62\},$
$CT = 85,$
$F = \{d_3 = \max(d_1, d_2), d_4 = (d_1 * d_2), d_5 = (d_1 / d_2)\}$
$OV = \{d_3, = 1\ d_4 = 1, d_5 = 1\},$
$PR = \phi,$
$PT = \phi\}.$

ks_2 = {IV = {$d_3 = 1$},
IC = {ic_3 = FALSE},
XD = {22},
CT = 107,
F = {$d_6 = (d_3 / 4.10)$, $d_7 = (d_3 * 0.41)$}
OV = {$d_6 = 2.43$, $d_7 = 0.41$},
PR = {$pr_1 = (d_4 \neq 7)$},
PT = ϕ}.

ks_3 = {IV = {$d_4 = 1$, $d_6 = 2.43$},
IC = {ic_4 = FALSE, ic_6 = FALSE},
XD = {27},
CT = 134,
F = {$d_8 = \max(d_4,d_6)$},
OV = {$d_8 = 1$},
PR = {$pr_2 = (d_8 \neq pi)$, $pr_3 = (d_9 \neq 13.6)$},
PT = {$pt_1 = (d_5 \neq U)$}.

ks_4 = {IV = {$d_5 = 1$, $d_7 = 0.41$, $d_8 = 1$},
IC = {ic_5 = FALSE, ic_7= FALSE, ic_8= FALSE},
XD = {52},
CT = 186,
F = {$d_9 = (\max(d_5, d_7, d_8) / \min(d_5, d_7, d_8))$}
OV = {d_9},
PR = {$pr_3 = (d_9 \neq 13.6)$, $pr_4 = (d_2 \neq 0.0)$, $pr_5 = (d_8 \neq 0.0)$},
PT = ϕ.

- X = {$d_1 = 1$, $d_2 = 1$, $d_3 = 1$, $d_4 = 1$, $d_5 = 1$, $d_6 = 2.43$, $d_7 = 0.41$, $d_8 = 1$, $d_9 = U$}
- A = {ks_4}

Fourth Simulation Loop:

Ks_4 is removed from the event queue. Δ is set to the completion time of ks_4 and the function F is computed. Ks_4's output variables are updated on the blackboard.

All sensors are tested and none will be activated before the completion of ks_4.

$\Delta = 186$

$ks_1 = \{IV = \{d_1 = 1, d_2 = 1\},$
$IC = \{ic_1 = FALSE, ic_2 = FALSE\},$
$UR = \{133\},$
$AT = 218,$
$XD = \{62\},$
$CT = 85,$
$F = \{d_3 = \max(d_1, d_2), d_4 = (d_1 * d_2), d_5 = (d_1 / d_2)\}$
$OV = \{d_3, = 1\ d_4 = 1, d_5 = 1\},$
$PR = \phi,$
$PT = \phi\}.$

$ks_2 = \{IV = \{d_3 = 1\},$
$IC = \{ic_3 = FALSE\},$
$XD = \{22\},$
$CT = 107,$
$F = \{d_6 = (d_3 / 4.10), d_7 = (d_3 * 0.41)\}$
$OV = \{d_6 = 2.43, d_7 = 0.41\},$
$PR = \{pr_1 = (d_4 \neq 7)\},$
$PT = \phi\}.$

$ks_3 = \{IV = \{d_4 = 1, d_6 = 2.43\},$
$IC = \{ic_4 = FALSE, ic_6 = FALSE\},$
$XD = \{27\},$
$CT = 134,$
$F = \{d_8 = \max(d_4, d_6)\},$
$OV = \{d_8 = 1\},$
$PR = \{pr_2 = (d_8 \neq pi), pr_3 = (d_9 \neq 13.6)\},$
$PT = \{pt_1 = (d_5 \neq U)\}.$

ks_4 = {IV = {d_5 = 1, d_7 = 0.41, d_8 = 1},
IC = {ic_5 = FALSE, ic_7= FALSE, ic_8= FALSE},
XD = {52},
CT = 186,
F = {d_9 = (max(d_5, d_7, d_8) / min(d_5, d_7, d_8))}
OV = {d_9 = 2.43},
PR = {pr_3 = ($d_9 \neq 13.6$), pr_4 = ($d_2 \neq 0.0$), pr_5 = ($d_8 \neq 0.0$)},
PT = ϕ.

- X = {d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 2.43, d_7 = 0.41, d_8 = 1, d_9 =2.43}
- A = {ϕ}

At this time the event queue is empty. The sensor ks_1 is ready to be activated at $\Delta = 218$. Data arrives at the sensor, ks_1's input conditionals are set to TRUE, and the global clock is set to ks_1's activation time

Sensor ks_1 is ready to execute (ic_1 = TRUE, ic_2 = TRUE). An execution delay and a completion time are computed for ks_1.

$$XD = 71$$
$$CT = \Delta + XD,\ CT = 218 + 71\ CT = 289 \qquad (B.7)$$

Ks_1 performs a read operation to read its input variables, is pushed onto the queue of executable knowledge sources, and ks_1's input conditionals are reset to false.

$\Delta = 218$
ks_1 = {IV = {d_1 = 7,d_2 = 4},
IC = {ic_1 = FALSE, ic_2 = FALSE},
UR = {133},
AT = 218,
XD = {41},
CT = 289,
F = {d_3 = max(d_1, d_2), d_4 = (d_1 * d_2), d_5 = (d_1 / d_2)}
OV = {d_3, = 1 d_4 = 1, d_5 = 1},
PR = ϕ,
PT = ϕ}.

ks_2 = {IV = {$d_3 = 1$},
IC = {ic_3 = FALSE},
XD = {22},
CT = 107,
F = {$d_6 = (d_3 / 4.10)$, $d_7 = (d_3 * 0.41)$}
OV = {$d_6 = 2.43$, $d_7 = 0.41$},
PR = {$pr_1 = (d_4 \neq 7)$},
PT = ϕ}.

ks_3 = {IV = {$d_4 = 1$, $d_6 = 2.43$},
IC = {ic_4 = FALSE, ic_6 = FALSE},
XD = {27},
CT = 134,
F = {$d_8 = \max(d_4, d_6)$},
OV = {$d_8 = 1$},
PR = {$pr_2 = (d_8 \neq pi)$, $pr_3 = (d_9 \neq 13.6)$},
PT = {$pt_1 = (d_5 \neq U)$}.

ks_4 = {IV = {$d_5 = 1$, $d_7 = 0.41$, $d_8 = 1$},
IC = {ic_5 = FALSE, ic_7= FALSE, ic_8= FALSE},
XD = {52},
CT = 186,
F = {$d_9 = (\max(d_5, d_7, d_8) / \min(d_5, d_7, d_8))$}
OV = {$d_9 = 2.43$},
PR = {$pr_3 = (d_9 \neq 13.6)$, $pr_4 = (d_2 \neq 0.0)$, $pr_5 = (d_8 \neq 0.0)$},
PT = ϕ.

- X = {$d_1 = 7$, $d_2 = 4$, $d_3 = 1$, $d_4 = 1$, $d_5 = 1$, $d_6 = 2.43$, $d_7 = 0.41$, $d_8 = 1$, $d_9 = 2.43$}
- A = {ks_1}

The simulation has completed its first execution loop through the blackboard system. This loop will continue until data containing a shutdown message arrives at the sensor ks_1 or ks_1 processes all of the inputs in its data stream.

Appendix C

This appendix contains a listing of the Knowledge Source Organizer LISP code and the Knowledge Source Connectivity Analysis Lisp Code.

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: kstools; Base: 10 -*-
;;;*****************************************************************
;;;
*
;;; Program: Blackboard System Design Tools
;;;        Knowledge Source Input/Output Connectivity Analyzer
;;;
;;;*********************************
;;; define the program framework *
;;;*********************************

(DW:DEFINE-PROGRAM-FRAMEWORK  BLACKBOARD

;;;************************
;;; define the top level *
;;; program framework    *
;;;************************

  :top-level (my-top-level)

;;;************************************
;;; define the program selection key *
;;;************************************

  :SELECT-KEY
  #\k

;;;*********************************
;;; define the program framework *
;;;*********************************

  :COMMAND-DEFINER
  T

;;;************************************
;;; define the program command table *
;;;************************************

  :COMMAND-TABLE
  (:INHERIT-FROM
  '("accept-values-pane" "colon full command" "standard arguments"
    "input editor compatibility" "standard scrolling")
  :KBD-ACCELERATOR-P 'T)
```

```
;;;*************************************
;;; define the program state variables *
;;; and their initial values           *
;;;*************************************

 :STATE-VARIABLES
 ()

;;;*******************************
;;; define the program windows *
;;;*******************************

 :PANES
 ((PANE-1 :TITLE :REDISPLAY-STRING "Blackboard System Design Tools"
       :HEIGHT-IN-LINES 1
       :REDISPLAY-AFTER-COMMANDS NIL)
  (PANE-4 :display
       :margin-components '((dw:margin-borders)
                    (dw:margin-label :string "Graphic Display Window"
                                 :margin :top
                                 :box :inside
                                 :box-thickness 2
                                 :style (:fix :bold :large)
                                 :centered-p t)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-scroll-bar)) )
  (PANE-3 :listener
       :HEIGHT-IN-LINES 35
       :margin-components '((dw:margin-borders)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-label :string "Listener Window"
                                 :margin :top
                                 :box :inside
                                 :box-thickness 2
                                 :style (:fix :bold :large)
                                 :centered-p t)
                    (dw:margin-scroll-bar)) )
  (PANE-6 :display
       :margin-components '((dw:margin-borders)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-label :string "List of Knowledge Sources"
                                 :margin :top
                                 :box :inside
                                 :box-thickness 2
                                 :centered-p t
                                 :style (:fix :bold :large))
                    (dw:margin-scroll-bar)) )
  (PANE-5 :COMMAND-MENU
```

```
        :CENTER-P T
        :MENU-LEVEL
        :TOP-LEVEL
        :margin-components '((dw:margin-borders)
                             (dw:margin-white-borders :thickness 4)
                             (dw:margin-label :string "Commands"
                                              :margin :top
                                              :box :inside
                                              :box-thickness 2
                                              :style (:fix :bold :large)
                                              :centered-p t)) )
  )


;;;*****************************************
;;; define the window layout configuration *
;;;*****************************************

  :CONFIGURATIONS
  '((DW::MAIN
     (:LAYOUT (DW::MAIN :COLUMN PANE-1 PANE-4 ROW-1)
      (ROW-1 :ROW PANE-3 PANE-6 PANE-5))
     (:SIZES (DW::MAIN (PANE-1 1 :LINES)
                       :THEN (PANE-4 :EVEN) (ROW-1 :EVEN))
      (ROW-1 (PANE-3 30 :LINES) (PANE-6 20 :LINES)
       (PANE-5 :ASK-WINDOW SELF
               :SIZE-FOR-PANE PANE-5) :THEN)))))

;;;*******************************
;;; define any needed variables *
;;;*******************************

  (defvar *data-out* nil)        ;;; Output Data File
  (defvar *data-in* nil)         ;;; Input Data File
  (defvar beta '())              ;;; List Of Knowledge Sources
  (defvar ks-list '())           ;;; List Of Knowledge Source Names
  (defvar data-object-list '())  ;;; List Of Blackboard Data Object Names
  (defvar d/o-list '())          ;;; List Of Blackboard Data Object(s)
  (defvar ks-in '())             ;;; Knowledge Source selected from menu
  (defvar do-in '())             ;;; Blackboard Data object selected from menu
  (defvar ks-name-in '())
  (defvar ks-edit '())
  (defvar do-edit '())
  (defvar data-in1 '())
  (defvar gamma-list '())
  (defvar lambda-ks-list '())
  (defvar omega-list '())
  (defvar pie-list '())
  (defvar sigma-list '())
```

```
(defvar ks_count 0)
(defvar sensor_count 0)
(defvar undefined '"undefined")


(defvar data-in)
(defvar temp)
(defvar ix)
(defvar in-psi)
(defvar in-phi)
(defvar in-pre)
(defvar in-post)
(defvar in-exec)
(defvar in-urate)
(defvar in-name)
(defvar in-value)
(defvar in-type)
(defvar in-in_list)
(defvar in-out_list)

(defvar pointstack '())
(defvar markstack '())
(defvar mark '())
(defvar adj-list '())
(defvar flag nil)
(defvar u)
(defvar connected nil)
(defvar *ks-spec* nil)
(defvar *do-spec* nil)

(defvar *pane1*)
(defvar *pane3*)
(defvar *pane4*)
(defvar *pane5*)
(defvar *pane6*)

;;;*********************************************
;;; define the variables required to build a  *
;;; pop up window to select knowledge sources *
;;;*********************************************

(defvar geometry-list (list 2))

(defvar *ks-menu* (tv:make-window 'tv:pop-up-menu
                        ':label '(:string "select a knowledge source"
                                :character-style (:swiss :italic :normal))
;;                      ':geometry geometry-list
                        ':borders 4))
```

```
(defvar *do-menu* (tv:make-window 'tv:pop-up-menu
                    ':label '(:string "select a Blackboard Data Object"
                          :character-style (:swiss :italic :normal))
                    ':borders 4))

;;;*****************************************
;;; define any needed presentation types. *
;;;*****************************************

(define-presentation-type ks-type ()
  :abbreviation-for `string)

(define-presentation-type output-file-name ()
  :abbreviation-for `string)

(define-presentation-type input-file-name ()
  :abbreviation-for `string)

(define-presentation-type input-conditional-list ()
  :abbreviation-for `list)

(define-presentation-type precondition-list ()
  :abbreviation-for `list)

(define-presentation-type postcondition-list ()
  :abbreviation-for `list)

(define-presentation-type input-variable-list ()
  :abbreviation-for `list)

(define-presentation-type output-variable-list ()
  :abbreviation-for `list)

(define-presentation-type rotation-about-y-axis-in-degrees ((limit))
  :abbreviation-for `((integer 0,limit)))

(define-presentation-type ks-name (())
  :no-deftype t
  :printer ((ks-name stream)
            (format stream " ~% knowledge source ~a" ks-name))
  :parser ((stream)
        (accept 'string :stream stream
                    :prompt "enter a Knowledge Source name")) )

;;;************************************************
;;; define area for instances for specialized GC. *
;;;************************************************

(defvar *instance-area*
```

```
    (make-area :name '*instance-area*
               :gc ':dynamic) )

;;;**************************************
;;;define Flavor for a Knowledge Source *
;;;**************************************

(defflavor base_ks ((name " " 'ks-name)
                    (type " " 'ks-type)
                    (input-conditionals '() 'input-conditional-list)
                    (preconditions '() 'precondition-list)
                    (postconditions '() 'postcondition-list)
                    (psi '() 'input-variable-list)
                    (phi '() 'output-variable-ist)
                    (execution_time 0.0 'number)
                    (depth 0 'number)
                    (successor_list '() 'list)
                    (predecessor_list '() 'list)
                    (update_rate 0.0 'number)
                    (psi_card 0 'integer)
                    (phi_card 0 'integer))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type base_ks ())
  :no-deftype t
  :printer ((base_ks stream)
            (format stream " ~% knowledge source: ~a ~
                  ~& knowledge source type: ~a ~
                  ~& input variables: ~a ~
                  ~& output variables: ~a ~
                  ~& input conditionals: ~d ~
                  ~& preconditions: ~d ~
                  ~& postconditions: ~d ~
                  ~& depth: ~d ~
                  ~& predecessors: ~d ~
                  ~& successors: ~d ~
                  ~& knowledge source execution time: ~d ~
                  ~& cardinality of input variable set: ~d ~
                  ~& cardinality of output variable set: ~d ~%"
                  (get-name base_ks) (get-type base_ks)
                  (get-psi base_ks) (get-phi base_ks)
                  (get-input-conditionals base_ks)
                  (get-preconditions base_ks) (get-postconditions base_ks)
                  (get-depth base_ks) (get-predecessor_list base_ks)
                  (get-successor_list base_ks)
                  (get-execution_time base_ks)
```

```
                (get-psi_card base_ks) (get-phi_card base_ks)) ) )

;;;*************************************
;;;define methods for a knowledge source *
;;;*************************************

(defmethod (show base_ks)
 ()
(format *data-out* " ~% knowledge source: ~a ~
                ~& knowledge source type: ~a ~
                ~& input variables: ~a ~
                ~& output variables: ~a ~
                ~& input conditionals: ~d ~
                ~& preconditions: ~d ~
                ~& postconditions: ~d ~
                ~& depth: ~d ~
                ~& predecessors: ~d ~
                ~& successors: ~d ~
                ~& knowledge source execution time: ~d ~
                ~& cardinality of input variable set: ~d ~
                ~& cardinality of output variable set: ~d ~%"
        name type psi phi input-conditionals preconditions postconditions
        depth predecessor_list successor_list execution_time
        psi_card  phi_card)
)

(defmethod (display base_ks)
 ()
 (format *data-out* " ~% ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~%"
       name type psi phi input-conditionals preconditions postconditions
       depth predecessor_list successor_list execution_time
       psi_card  phi_card) )

(defmethod (compute_cardinality base_ks)
 ()
  (setf psi_card (length psi)
      phi_card (length phi)) )
```

```
;;;*****************************************
;;;
;;;Define flavor for a Blackboard Data Object *
;;;*****************************************
;;;

(defflavor blackboard_data_object ((name nil)
                                   (type      "" " 'string)
                                   (value     'undefined)
                                   (ic_list   '() 'list)
                                   (lock      '() 'list)
                                   (input_list '() 'list)
                                   (output_list '() 'list))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type blackboard_data_object (())
  :no-deftype t
  :printer ((blackboard_data_object stream)
            (format stream "~& Blackboard Data Object Name: ~a ~
                            ~& Data Object Type: ~a ~
                            ~& Data Object Value: ~a ~
                            ~& Data Object Lock: ~a ~
                            ~& Used as an Input by: ~a ~
                            ~& Input Conditionals: ~a ~
                            ~& Output by: ~a ~%"
                    (get-name blackboard_data_object)
                    (get-type blackboard_data_object)
                    (get-value blackboard_data_object)
                    (get-lock blackboard_data_object)
                    (get-input_list blackboard_data_object)
                    (get-ic_list blackboard_data_object)
                    (get-output_list blackboard_data_object)) ) )

(defmethod (display blackboard_data_object)
  ()
  (format *data-out* " ~% ~a ~
                       ~& ~a ~
                       ~& ~a ~
                       ~& ~a ~
                       ~& ~a ~
                       ~& ~a ~
                       ~& ~a ~%"
          name type value lock input_list ic_list output_list) )

;;;*********************************
;;;
;;;define flavor for a Gamma-List *
;;;*********************************
;;;
```

```
(defflavor gamma ((name nil)
                  (type '"gamma")
                  (cardinality 0)
                  (value '()))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type gamma ())
  :no-deftype t
  :printer ((gamma stream)
            (format stream "gamma<~a> ~a ~t cardinality = ~d ~%"
                  (get-name gamma) (get-value gamma) (get-cardinality gamma))) )

;;;*************************************************
;;;define functions and methods for a gamma-list *
;;;*************************************************

(defun create_gamma (ksj ksk)
  (pushnew (make-instance 'gamma
                  :area  *instance-area*
                  :name  (build_name ksj ksk)
                  :value '())
       gamma-list)
  (pushnew (make-instance 'omega
                  :area  *instance-area*
                  :name  (build_name ksj ksk)
                  :value '())
       omega-list)

  (make_omega (car omega-list)
          (make_gamma (car gamma-list) (get-phi ksj) (get-phi ksk))
          (get-phi_card ksj) (get-phi_card ksk)) )

(defmethod (make_gamma gamma)
  (phij phik)
   (setf value (intersection phij phik)
       cardinality (length value)) )

(defmethod (display gamma)
  ()
  (cond
    (value (format *data-out* "gamma<~a> ~a ~T cardinality = ~d ~%"
                    name value cardinality))
    (t t))
  )
```

```
;;;*******************************
;;;define Flavor for Lambda-ks-list *
;;;*******************************
;;;

(defflavor lambda-ks ((name nil)
                      (type '"lambda-ks")
                      (cardinality 0)
                      (value '()))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type lambda-ks ())
  :no-deftype t
  :printer ((lambda-ks stream)
            (format stream "lambda<~a> ~a ~t cardinality = ~d ~%"
                    (get-name lambda-ks) (get-value lambda-ks) (get-cardinality lambda-ks)))
)

;;;************************************************
;;;define functions and methods for a lambda-ks-list *
;;;************************************************
;;;

(defun create_lambda-ks (ksj ksk)
  (pushnew (make-instance 'lambda-ks
                          :area  *instance-area*
                          :name  (build_name ksj ksk)
                          :value '())
           lambda-ks-list)
  (pushnew (make-instance 'pie
                          :area  *instance-area*
                          :name  (build_name ksj ksk)
                          :value '())
           pie-list)
  (make_pi (car pie-list)
           (make_lambda-ks (car lambda-ks-list) (get-phi ksj) (get-psi ksk))
           (get-phi_card ksj))

  (pushnew (make-instance 'sigma
                          :area  *instance-area*
                          :name  (build_name ksj ksk)
                          :value '())
           sigma-list)
  (make_sigma (car sigma-list)
              (get-cardinality (car lambda-ks-list)) (get-psi_card ksk)) )

(defmethod (make_lambda-ks lambda-ks)
  (phij psik)
```

```
  (setf value (intersection phij psik)
      cardinality (length value)) )

(defmethod (display lambda-ks)
 ()
 (cond
  (value (format *data-out* "lambda<~a> ~a ~T cardinality = ~d ~%"
                 name value cardinality))
  (t t))
 )

;;;**********************************
;;;define Flavor for a omega-list *
;;;**********************************

(defflavor omega ((name nil)
                  (type '"omega")
                  (value 0))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type omega (())
  :no-deftype t
  :printer ((omega stream)
            (format stream "omega<~a> ~f ~%"
                (get-name omega) (get-value omega))) )

;;;************************************************
;;;define functions and methods for a omega-list *
;;;************************************************

(defmethod (make_omega omega)
 (cardinality_gamma card_phij card_phik)
  (let ((min_value (min card_phij card_phik)))
  (cond
    ((= min_value 0) (setf value 0))
    ((= cardinality_gamma 0) (setf value 0))
    (t (setf value (/ cardinality_gamma min_value))))) )

(defmethod (display omega)
 ()
 (cond
  ((= value 0) t)
  (t (format *data-out* "omega<~a> ~f ~%"
             name value)))
 )
```

```
;;;*****************************
;;;define Flavor for a Pi-list *
;;;*****************************
;;;

(defflavor pie ((name nil)
          (type '"phi")
          (value 0))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type pie (())
  :no-deftype t
  :printer ((pie stream)
        (format stream "pi<~a> ~f ~%"
            (get-name pie) (get-value pie))) )

;;;*******************************************
;;;define functions and methods for a Pi-list *
;;;*******************************************
;;;

(defmethod (make_pi pie)
          (cardinality_lambda-ks card_phij)
 (cond
  ((= card_phij 0) (setf value 0))
  ((= cardinality_lambda-ks 0) (setf value 0))
  (t (setf value (/ cardinality_lambda-ks card_phij)))) )

(defmethod (display pie)
 ()
 (cond
  ((= value 0) t)
  (t (format *data-out* "pi<~a> ~f ~%"
          name value)))
 )

;;;*********************************
;;;
;;; Define Flavor for a Sigma-list *
;;;*********************************
;;;

(defflavor sigma ((name nil)
          (type '"sigma")
          (value 0))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)
```

```
(define-presentation-type sigma (())
  :no-deftype t
  :printer ((sigma stream)
          (format stream "sigma<~a> ~f ~%"
               (get-name sigma) (get-value sigma))) )

;;;*********************************
;;;* Define functions and methods *
;;;* for a Sigma-list             *
;;;*********************************

(defmethod (make_sigma sigma)
          (cardinality_lambda-ks card_psik)
  (cond
   ((= card_psik 0) (setf value 0))
   ((= cardinality_lambda-ks 0) (setf value 0))
   (t (setf value (/ cardinality_lambda-ks card_psik))))
  )

(defmethod (display sigma)
          ()
  (cond
   ((= value 0) t)
   (t (format *data-out* "sigma<~a> ~f ~%"
              name value)))
  )

;;;*************************************************************
;;;define functions to build component names from KS names. *
;;;*************************************************************

(defun build_name (ksj ksk)
  (string-append (get-name ksj) (get-name ksk)) )

(defun build_ks_name ()
  (setf ks_count (1+ ks_count))
  (string-append "ks" (princ-to-string ks_count)) )

(defun display-ks-list ()
  (send *pane6* :clear-window)
  (present "Current List of Knowledge Sources" 'string :stream *pane6*)
  (loop for x in ks-list
       do (present x 'ks-name :stream *pane6*)) )

;;;*******************************
;;; define blackboard commands *
;;;*******************************
;;; Delete a KS Command *
;;;**********************
```

```
(define-blackboard-command (delete_ks :menu-accelerator
                                      "Delete a Knowledge Source"
                                      :keyboard-accelerator #\d)
                    ()
  (setf ks-in nil)
  (select-ks-name)

  (setf ks-list (remove ks-in ks-list))

  (loop for x in beta
      when (equal ks-in (get-name x))
       do (setf beta (remove x beta)))

  (send *pane6* :clear-window)
  (display-ks-list)
    )

;;;*********************
;;; View a KS Command *
;;;*********************

(define-blackboard-command (display_ks :menu-accelerator
                                       "View a Knowledge Source"
                                       :keyboard-accelerator #\v)
                    ()
  (setf ks-in nil)
  (select-ks-name)

  (send *pane4* :clear-window)
  (loop for x in beta
      when (equal ks-in (get-name x))
       do (present x 'base_ks :stream *pane4*)
         )
  )

;;;*********************************
;;; View a BB data object Command *
;;;*********************************

(define-blackboard-command (display_do :menu-accelerator
                                       "View a Blackboard Data Object"
                                       :keyboard-accelerator #\5)
                    ()
  (setf data-in nil)
  (select-data-object-name)

  (send *pane4* :clear-window)
  (loop for x in d/o-list
```

```
      when (equal data-in1 (get-name x))
       do (present x 'blackboard_data_object :stream *pane4*))
  )

;;;*******************************
;;; Show Adjacency List Command *
;;;*******************************

(define-blackboard-command (show_adj :menu-accelerator
                                     "Show Adjacency List"
                                     :keyboard-accelerator #\s)
                    ()
  (send *pane4* :clear-window)
  (format *pane4* " Adjacency List ~&")
  (dotimes (i ks_count)
   (get-adj  i (aref adj-list i)))
  )

(defun get-adj (i adj-list)
 (cond
  ((NOT adj-list) t)
  (t (format *pane4* "Knowledge Source: ~a, Adjacency List: ~a ~&"
             (get-name (nth i (reverse beta)))
             (get-name (nth (car adj-list) (reverse beta))))
    (get-adj i (cdr adj-list))) )
  )

;;;******************
;;; Add KS Command *
;;;******************

(define-blackboard-command (add_ks :menu-accelerator "Add a Knowledge Source"
                     :keyboard-accelerator #\a)
                    ()
  (pushnew (make-instance  'base_ks
                  :area   *instance-area*
                  :depth  ks_count
                  :name   (build_ks_name)
                  :type   (accept 'ks-type)
                  :psi    (accept 'input-variable-list)
                  :phi    (accept 'output-variable-list))
     beta)

  (pushnew (get-name (car beta)) ks-list)
  (update-blackboard-data-objects (car beta))
  (setf (get-psi_card (car beta)) (length (get-psi (car beta)))
     (get-phi_card (car beta)) (length (get-phi (car beta))) )
  (build_ic (car beta) (get-psi (car beta)))
  (send *pane6* :clear-window)
```

```
  (display-ks-list)
  )

(defun build_ic (ks-in bb-data-list)
  (cond
   ((NOT bb-data-list) t)
   (t (pushnew (build_ic-name (get-name ks-in) (car bb-data-list))
               (get-input-conditionals ks-in))
      (build_ic ks-in (cdr bb-data-list))) )
  )

(defun build_ic-name (ks-name bb-data)
  (string-append ks-name "-" bb-data) )

;;;*****************************
;;;Function to update the set *
;;;of blackboard data objects *
;;;*****************************

(defun update-blackboard-data-objects (ks-in)
 (let ((ks-name-in (get-name ks-in)))
  (mapcar 'test-blackboard-in-data-objects (get-psi ks-in))
  (mapcar 'test-blackboard-out-data-objects (get-phi ks-in))) )

(defun test-blackboard-in-data-objects (data-in)
 (cond
  ((member data-in data-object-list) (update-inputs data-in))
  (t (add-bbdata-object data-in)
     (pushnew (string-append ks-name-in "-" (get-name (car d/o-list)))
              (get-ic_list (car d/o-list)) )
     (setf data-object-list (union (list data-in) data-object-list))
     (setf (get-input_list (car d/o-list)) (union (list ks-name-in)
                                                  (get-input_list (car d/o-list))))) )
 )

(defun test-blackboard-out-data-objects (data-in)
  (cond
   ((member data-in data-object-list) (update-outputs data-in))
   (t (add-bbdata-object data-in)
      (setf data-object-list (union (list data-in) data-object-list))
      (setf (get-output_list (car d/o-list)) (union (list ks-name-in)
                                          (get-output_list (car d/o-list))))) )
  )

(defun update-inputs (data-in)

  (loop for x in d/o-list
      when (equal data-in (get-name x))
       do (setf do-edit x))
```

```
 (setf (get-input_list do-edit) (union (list ks-name-in)
                                (get-input_list do-edit)))
 (pushnew (string-append ks-name-in "-" (get-name do-edit)) (get-ic_list do-edit) )
 (setf (get-ic_list do-edit) (remove-duplicates (get-ic_list do-edit) :test #'string-equal))
 )

(defun update-outputs (data-in)

  (loop for x in d/o-list
      when (equal data-in (get-name x))
       do (setf do-edit x))

  (setf (get-output_list do-edit) (union (list ks-name-in)
                                (get-output_list do-edit)))
  )

;;;*********************
;;; Edit a KS Command *
;;;*********************

(define-blackboard-command (edit_ks :menu-accelerator "Edit a Knowledge Source"
                                    :keyboard-accelerator #\e)
                    ()
  (setf ks-in nil)
  (select-ks-name)

  (loop for x in beta
      when (equal ks-in (get-name x))
       do (setf ks-edit x))

  (setf in-psi (get-psi ks-edit)
      in-phi (get-phi ks-edit)
      in-pre (get-preconditions ks-edit)
      in-post (get-postconditions ks-edit)
      in-exec (get-execution_time ks-edit)
      in-urate (get-update_rate ks-edit)
      )

    (tv:choose-variable-values
     '((in-psi "Input Variables" :expression)
       (in-phi "Output Variables" :expression)
       (in-pre "Preconditions" :expression)
       (in-post "Postconditions"  :expression)
       (in-exec "Execution Rate"  :expression)
       (in-urate "Update Rate"  :expression))
     ':label "Knowledge Source Editor")

    (setf (get-psi ks-edit) in-psi
```

```
        (get-phi ks-edit) in-phi
        (get-preconditions ks-edit) in-pre
        (get-postconditions ks-edit) in-post
        (get-execution_time ks-edit) in-exec
        (get-update_rate ks-edit) in-urate
        (get-psi_card ks-edit) (length in-psi)
        (get-phi_card ks-edit) (length in-phi) )

    (update-blackboard-data-objects ks-edit)
    (setf (get-psi_card ks-edit) (length (get-psi ks-edit))
          (get-phi_card ks-edit) (length (get-phi ks-edit)) )
    (setf (get-input-conditionals ks-edit) '())
    (build_ic ks-edit (get-psi ks-edit))
    (send *pane6* :clear-window)
    (display-ks-list)
  )

;;;*****************************************
;;;Function to add a blackboard data object *
;;;*****************************************

(defun add-bbdata-object (name)
  (pushnew (make-instance  'blackboard_data_object
                    :area      *instance-area*
                    :name       name
                    :value     'undefined
                    :lock      (string-append name "-lock")
                    :type      nil
                    :ic_list   '()
                    :input_list  '()
                    :output_list '())
        d/o-list)

  (setf in-name (get-name (car d/o-list))
        in-type (get-type (car d/o-list))
        in-value (get-value (car d/o-list)) )

  (tv:choose-variable-values
    '((in-name "Data Object Name" :expression)
      (in-type "Data Object Type"  :expression)
      (in-value "Data Object Value"  :expression))
    ':label "Knowledge Source Editor")

  (setf (get-name (car d/o-list)) in-name
        (get-type (car d/o-list)) in-type
        (get-value (car d/o-list)) in-value)
  )

;;;**************************
```

```
;;; Load a BB Specification *
;;;*************************

(define-blackboard-command (load_spec :menu-accelerator
                                      "Load a Blackboard Specification"
                                      :keyboard-accelerator #\l)
                    ()
;;; open the specification file

 (setf *data-in* (open (fs:merge-pathnames (accept 'input-file-name))
                  :direction :input))

;;; Read in the Number of Knowledge Sources in the Specification file

 (init_system)
 (setf *ks-spec* (read *data-in*))

 (dotimes (i *ks-spec*)
  (pushnew (make-instance  'base_ks
                    :name (read *data-in*)
                    :type (read *data-in*)
                    :psi (read *data-in*)
                    :phi (read *data-in*)
                    :input-conditionals (read *data-in*)
                    :preconditions (read *data-in*)
                    :postconditions (read *data-in*)
                    :depth (read *data-in*)
                    :predecessor_list (read *data-in*)
                    :successor_list (read *data-in*)
                    :execution_time (read *data-in*)
                    :psi_card (read *data-in*)
                    :phi_card (read *data-in*))
     beta)

  (pushnew (get-name (car beta)) ks-list)
  )

 (setf *do-spec* (read *data-in*))

 (dotimes (i *do-spec*)
  (pushnew (make-instance  'blackboard_data_object
                           :area       *instance-area*
                           :name        (read *data-in*)
                           :type       (read *data-in*)
                           :value       (read *data-in*)
                           :lock       (read *data-in*)
                           :input_list  (read *data-in*)
                           :ic_list    (read *data-in*)
                           :output_list (read *data-in*))
```

```
          d/o-list)
   (pushnew (get-name (car d/o-list)) data-object-list)
)


;;; close the specification file

  (setf ks_count *ks-spec*)
  (close *data-in*)
  (send *pane6* :clear-window)
  (display-ks-list)
  )

;;;***************
;;; Quit Command *
;;;***************

(define-blackboard-command (quit_bb :menu-accelerator
                                    "Quit"
                                    :keyboard-accelerator #\q)
                   ()
  (send *pane1* :clear-window)
  (send *pane3* :clear-window)
  (send *pane4* :clear-window)
  (send *pane5* :clear-window)
  (send *pane6* :clear-window)
  (send dw:*program-frame* :clear-window)
  (send *pane1* :bury)
  (send *pane3* :bury)
  (send *pane4* :bury)
  (send *pane5* :bury)
  (send *pane6* :bury)
  (send dw:*program-frame* :bury)
  (process-abort *current-process* :all t))

;;;**************************
;;; Generate Output Command *
;;;**************************

(define-blackboard-command (generate_output :menu-accelerator
                                            "Generate System Output File"
                                            :keyboard-accelerator #\g)
  ()
 (setf *data-out* (open (fs:merge-pathnames (accept 'output-file-name))
                  :direction :output
                  :if-exists :new-version))
 (format *data-out* "Set of Knowledge Sources~%")
 (mapcar 'show beta)
 (format *data-out* "~%Gamma~%")
```

```
 (mapcar 'display gamma-list)
 (format *data-out* "~%Lambda-ks~%")
 (mapcar 'display lambda-ks-list)
 (format *data-out* "~%Omega~%")
 (mapcar 'display omega-list)
 (format *data-out* "~%Pi~%")
 (mapcar 'display pie-list)
 (format *data-out* "~%Sigma~%")
 (mapcar 'display sigma-list)
 (close *data-out*)
 (format t "~%"))

;;;*****************************
;;; Generate Data File Command *
;;;*****************************

(define-blackboard-command (generate_bb_spec :menu-accelerator
                                              "Generate BlackBoard Specification"
                                              :keyboard-accelerator #\4)
  ()
 (setf *data-out* (open (fs:merge-pathnames (accept 'output-file-name))
                  :direction :output
                  :if-exists :new-version))
 (format *data-out* "~& ~d" (length beta))
 (mapcar 'display beta)
 (format *data-out* "~% ~d" (length d/o-list))
 (mapcar 'display d/o-list)
 (close *data-out*)
 (format t "~%"))

;;;**************************
;;; Compute Circuit Command *
;;;**************************

(define-blackboard-command (Compute_circuit :menu-accelerator
                                            "Compute Circuit(s)"
                                            :keyboard-accelerator #\c)
  ()
 (setf flag nil
       pointstack '()
       markstack '())
 (send *pane4* :clear-window)
 (dotimes (i ks_count)
  (backtrack i i)
  (print (list "mark " markstack))
  (loop until (NOT markstack)
       do (setf u (pop markstack)
                (aref mark u) nil)
          (print u))
```

```
   )
  (format t "~%"))

;;;*************************************
;;;define functions to compute circuits *
;;;*************************************

(defun backtrack (s k)
 (setf flag nil)
 (pushnew k pointstack)
 (setf (aref mark k) t)
 (pushnew k markstack)
 (setf temp (aref adj-list k))

 (loop until (NOT temp)
     do (cond
;            ((< (car temp) s)
;              (pop temp))

            ((= (car temp) s)
             (setf flag t)
             (show-circuit s k pointstack))

            ((NOT (aref mark (car temp)))
             (backtrack s (car temp))
             (setf flag (or flag nil))))

          (pop temp))

 (cond (flag (setf ix (pop markstack)
                (aref mark ix) nil)
          (loop until (= ix k)
              do (setf ix (pop markstack)
                       (aref mark ix) nil)))
     (t t))
 (setf ix (pop pointstack))
)

;;;************************************************
;;; define function to display elementry circuits *
;;;************************************************

(defun show-circuit (s k pstack)
 (print (list "popstack "pstack))
  (loop until (or (equal (car pstack) s) (NOT pstack))
      do (pop pstack))

  (format *pane4* "Elementry Circuit ~&")
```

```
  (loop until (or (equal (car pstack) k) (NOT pstack))
      do (format *pane4* "~a " (get-name (nth (car pstack) (reverse beta))))
        (pop pstack))

  (format *pane4* "~a ~&" (get-name (nth k (reverse beta))))
  )

;;;*****************************
;;; Build Connectivity Graph *
;;;*****************************

(define-blackboard-command (build_graph :menu-accelerator
                                        "Build Connectivity Graph"
                                        :keyboard-accelerator #\b)
                  ()

;;;*****************************
;;; Clear connectivity lists *
;;;*****************************

  (setf gamma-list '()
      lambda-ks-list '()
      omega-list '()
      pie-list '()
      sigma-list '()
      pointstack '()
      markstack '()
      connected t)

  (setf adj-list (make-array ks_count
                    :initial-element '()))

  (setf mark (make-array ks_count
                    :initial-element nil))

;;;*****************************
;;; Build connectivity Graph *
;;;*****************************

  (build_connect beta beta)
  )

;;;*****************************************************************************
;;; define functions to build connectivity sets and adjacency list. *
;;;*****************************************************************************

(defun build_connect (ks-list beta-list)
  (cond
    ((NOT ks-list) t)
```

```
    (t (build_item (car ks-list) (remove (car ks-list) beta-list))
      (build_connect (cdr ks-list) beta-list)) ) )

(defun build_item (ksj ks-in)
  (cond
    ((NOT ks-in) t)
    (t (create_gamma ksj (nth 0 ks-in))
      (create_lambda-ks ksj (nth 0 ks-in))
      (create_successor ksj (nth 0 ks-in))
      (create_adjlist ksj (nth 0 ks-in))
      (create_predecessor ksj (nth 0 ks-in))
      (build_item ksj (cdr ks-in)))) )

;;;*************************************************
;;; function to compute the adjacency list of a KS *
;;;*************************************************

(defun create_adjlist (ksj ksk)
  (cond ((intersection (get-phi ksj) (get-psi ksk))
       (sort (pushnew (get-depth ksk) (aref adj-list (get-depth ksj))) '<) )
      (t t))
  )

;;;*********************************************
;;; function to compute the sucessors of a KS *
;;;*********************************************

(defun create_predecessor (ksj ksk)
  (cond ((intersection (get-psi ksj) (get-phi ksk))
       (pushnew (get-name ksk) (get-predecessor_list ksj)))
      (t t))
  )

;;;*********************************************
;;; function to compute the sucessors of a KS *
;;;*********************************************

(defun create_successor (ksj ksk)
  (cond ((intersection (get-phi ksj) (get-psi ksk))
       (pushnew (get-name ksk) (get-successor_list ksj)))
      (t t))
  )

;;;********************
;;; View Gamma Command *
;;;********************

(define-blackboard-command (display_gamma :menu-accelerator
                                          "View Gamma")
```

```
                ()
  (send *pane4* :clear-window)

  (loop for x in gamma-list
        do (present x 'gamma :stream *pane4*))
    )

;;;*************************
;;; View Lambda-ks Command *
;;;*************************

(define-blackboard-command (display_lambda :menu-accelerator
                                           "View Lambda")
                ()
  (send *pane4* :clear-window)

  (loop for x in lambda-ks-list
        do (present x 'lambda-ks :stream *pane4*))
    )

;;;**********************
;;; View Omega Command *
;;;**********************

(define-blackboard-command (display_omega :menu-accelerator "View Omega")
                ()
  (send *pane4* :clear-window)

  (loop for x in omega-list
        do (present x 'omega :stream *pane4*))
    )

;;;******************
;;; View Pi Command *
;;;******************

(define-blackboard-command (display_pi :menu-accelerator
                                       "View Pi")
                ()
  (send *pane4* :clear-window)

  (loop for x in pie-list
        do (present x 'pie :stream *pane4*))
    )

;;;**********************
;;; View Sigma Command *
;;;**********************
```

```
(define-blackboard-command (display_sigma :menu-accelerator
                                          "View Sigma")
                           ()
  (send *pane4* :clear-window)

  (loop for x in sigma-list
        do (present x 'sigma :stream *pane4*))
    )

;;;*****************************************
;;; Edit a Blackboard Data Object Command *
;;;*****************************************

(define-blackboard-command (edit_bb-do :menu-accelerator
                                       "Edit a Blackboard Data Object"
                                       :keyboard-accelerator #\2)
                           ()
  (setf data-in nil)
  (select-data-object-name)

  (loop for x in d/o-list
        when (equal data-in1 (get-name x))
        do (setf do-edit x))

  (setf in-name (get-name do-edit)
        in-type (get-type do-edit)
        in-value (get-value do-edit)
        in-in_list (get-input_list do-edit)
        in-out_list (get-output_list do-edit) )

  (tv:choose-variable-values
   '((in-name "Data Object Name" :expression)
     (in-type "Data Object Type"  :expression)
     (in-value "Data Object Value"  :expression)
     (in-in_list "Data Object Used as Input By"  :expression)
     (in-out_list "Data Object Updated as Output By"  :expression))
   ':label "Knowledge Source Editor")

  (setf (get-name do-edit) in-name
        (get-type do-edit) in-type
        (get-value do-edit) in-value
        (get-input_list do-edit) in-in_list
        (get-output_list do-edit) in-out_list)
  )

;;;*************************************
;;; Clear the Blackboard Specification *
;;;*************************************
```

```
(define-blackboard-command (clear_bb-do :menu-accelerator
                                        "Clear the Blackboard Specification"
                                        :keyboard-accelerator #\0)
                           ()
  (send *pane3* :clear-window)
  (send *pane4* :clear-window)
  (send *pane6* :clear-window)
  (init_system)
  )

;;;*********************
;;; initialize system *
;;;*********************

(defun my-top-level (program)
  (init_system)
  (pop-panes)
  (dw:default-command-top-level program))

(defun init_system ()
 (setf beta '()
        data-object-list '()
        d/o-list '()
        gamma-list '()
        lambda-ks-list '()
        omega-list '()
        ks-list '()
        pie-list '()
        sigma-list '()
        ks_count 0
        sensor_count 0
        connected nil)
 )

(defun pop-panes ()
 (setf *pane1* (dw:get-program-pane 'pane-1)
        *pane3* (dw:get-program-pane 'pane-3)
        *pane4* (dw:get-program-pane 'pane-4)
        *pane5* (dw:get-program-pane 'pane-5)
        *pane6* (dw:get-program-pane 'pane-6))
 )

;;;*******************************************
;;; define a function to pop a menu to allow *
;;; the user to select a knowledge source    *
;;;*******************************************

(defun select-ks-name ()
  (send *ks-menu* ':set-item-list ks-list)
```

```
  (send *ks-menu* ':expose-near '(:mouse))
  (setq ks-in (send *ks-menu* ':choose))
  (send *ks-menu* ':deactivate) t)

(defun select-data-object-name ()
  (send *do-menu* ':set-item-list data-object-list)
  (send *do-menu* ':expose-near '(:mouse))
  (setq data-in1 (send *do-menu* ':choose))
  (send *do-menu* ':deactivate) t)
```

Appendix D Concurrent Blackboard Simulation System Verification Results and COBS Simulation System Software.

This appendix contains the software and simulation runs used to verify that the COBS Simulation System performed according to specifications. The COBS Simulation System software is included at the end of this appendix.

D.1 Simulation Software and Results for B_1

This section contains the B1 software generated by the COBS simulation tool and simulation runs for B1

D.1.1 COBS Simulation Software for B_1

```
;;; -*- Syntax: Common-Lisp; Package: COMMON-LISP-USER; Base: 10; Mode:
LISP -*-

;;;************************
;;;* Load the Simulation *
;;;* Support Functions   *
;;;************************

(si:load "m:>john>sim-base")

;;;*********************
;;;* Define Variables *
;;;*********************

(defvar d/o-list '())

(defvar k/s-list '())

(defvar ks-list  '(KS4 KS3 KS2 KS1))

(defvar undefined '"undefined")
(defvar D8-LOCK '())
(defvar D8 '())
(defvar KS4-D8 '())
(defvar D7-LOCK '())
(defvar D7 '())
(defvar KS4-D7 '())
(defvar D6-LOCK '())
(defvar D6 '())
(defvar KS3-D6 '())
```

```
(defvar D9-LOCK '())
(defvar D9 '())
(defvar KS2-D9 '())
(defvar D1-LOCK '())
(defvar D1 '())
(defvar KS1-D1 '())
(defvar D2-LOCK '())
(defvar D2 '())
(defvar KS1-D2 '())
(defvar D3-LOCK '())
(defvar D3 '())
(defvar KS2-D3 '())
(defvar D4-LOCK '())
(defvar D4 '())
(defvar KS3-D4 '())
(defvar D5-LOCK '())
(defvar D5 '())
(defvar KS4-D5 '())

(defvar KS4D5-list '())
(defvar KS4D7-list '())
(defvar KS4D8-list '())
(defvar KS3D4-list '())
(defvar KS3D6-list '())
(defvar KS2D3-list '())
(defvar KS2D9-list '())

;;;*****************************************
;;;* Define Blackboard Data Object Locks *
;;;*****************************************

(Setf D5-LOCK (process:make-lock "D5-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D4-LOCK (process:make-lock "D4-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D3-LOCK (process:make-lock "D3-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D2-LOCK (process:make-lock "D2-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D1-LOCK (process:make-lock "D1-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D9-LOCK (process:make-lock "D9-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D6-LOCK (process:make-lock "D6-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D7-LOCK (process:make-lock "D7-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D8-LOCK (process:make-lock "D8-LOCK" :type :multiple-reader-single-writer
:recursive t))
```

```
;;;**********************************
;;;* Build Knowledge Source Objects *
;;;**********************************

(defun build-ks-list ()

 (setf k/s-list '())

 (setf KS1 (make-instance 'base_ks
                          :name "KS1"
                          :type "SENSOR"
                          :init-function '(KS1-verify)
                          :act-function '(KS1-activation)
                          :exec-function '(KS1-func)
                          :execution-delay 5
                          :update-interval 0.0
                          :completion-time 0
                          :activation-time 0 ))

 (push KS1 k/s-list)

 (setf KS2 (make-instance 'base_ks
                          :name "KS2"
                          :type "PROCESSOR"
                          :init-function '(KS2-verify)
                          :act-function '(KS2-activation)
                          :exec-function '(KS2-func)
                          :execution-delay 4
                          :update-interval 0.0
                          :completion-time 0
                          :activation-time 0 ))

 (push KS2 k/s-list)

 (setf KS3 (make-instance 'base_ks
                          :name "KS3"
                          :type "PROCESSOR"
                          :init-function '(KS3-verify)
                          :act-function '(KS3-activation)
                          :exec-function '(KS3-func)
                          :execution-delay 6
                          :update-interval 0.0
                          :completion-time 0
                          :activation-time 0 ))

 (push KS3 k/s-list)

 (setf KS4 (make-instance 'base_ks
```

```
                        :name "KS4"
                        :type "PROCESSOR"
                        :init-function '(KS4-verify)
                        :act-function '(KS4-activation)
                        :exec-function '(KS4-func)
                        :execution-delay 8
                        :update-interval 0.0
                        :completion-time 0
                        :activation-time 0 ))

 (push KS4 k/s-list)

 )

;;;*************************************
;;;* Define Knowledge Source Processess *
;;;*************************************

(defun KS4-activation ()
 (setf KS4D5-list (push (fetch D5) KS4D5-list))
 (setf KS4D7-list (push (fetch D7) KS4D7-list))
 (setf KS4D8-list (push (fetch D8) KS4D8-list))
 (setf KS4D5-temp (pop ks4D5-list))
 (setf KS4D7-temp (pop ks4D7-list))
 (setf KS4D8-temp (pop ks4D8-list))
 (cond
  ((NOT ks4d5-list) (setf KS4-D8 '())
   (setf KS4-D7 '())
   (setf KS4-D5 '()))
  (t t))
 )

(defun KS4-verify ()
 (and KS4-D8 KS4-D7 KS4-D5 (NOT (= (GET-VALUE D9) 13.6))
     (NOT (= (GET-VALUE D2) 0.0)) (NOT (= (GET-VALUE D8) 0.0))))

(defun KS4-func ()
 (print "call to knowledge source handler")
 (ks4-exec)
 (update D9 KS4D9-temp)
)

(defun ks4-exec ()
 (setf KS4D9-temp (max KS4D5-temp KS4D7-temp KS4D8-temp))
 )

(defun KS3-activation ()
 (setf KS3D4-list (push (fetch D4) KS3D4-list))
 (setf KS3D6-list (push (fetch D6) KS3D6-list))
```

```
 (setf KS3D4-temp (pop ks3d4-list))
 (setf KS3D6-temp (pop ks3d6-list))
 (cond
  ((NOT ks3d4-list) (setf KS3-D6 '())
   (setf KS3-D4 '()))
  (t t))
 )

(defun KS3-verify ()
 (and KS3-D6 KS3-D4 (NOT (= (GET-VALUE D8) PI)) (NOT (= (GET-VALUE
D9) 13.6))))

(defun KS3-func ()
 (print "call to knowledge source handler")
 (ks3-exec)
 (update D8 KS3D8-temp)
 )

(defun ks3-exec ()
 (setf KS3D3-temp (max KS3D4-temp KS3D6-temp))
 )

(defun KS2-activation ()
 (setf KS2D3-list (push (fetch D3) KS2D3-list))
 (setf KS2D9-list (push (fetch D9) KS2D9-list))
 (setf KS2D3-temp (pop ks2d3-list))
 (setf KS2D9-temp (pop ks2d9-list))
 (cond
  ((NOT ks2d3-list) (setf KS2-D9 '())
   (setf KS2-D3 '()))
  (t t))
 )

(defun KS2-verify ()
 (and KS2-D9 KS2-D3 (NOT (= (GET-VALUE D4) 7)) (NOT (= (GET-VALUE
D8) PI))))

(defun KS2-func ()
 (print "call to knowledge source handler")
 (ks2-exec)
 (update D6 KS2D6-temp)
 (update D7 KS2D7-temp)
 )

(defun ks2-exec ()
 (setf KS2D6-temp (max KS3D3-temp 4.10))
 (setf KS2D7-temp (* KS2D9-temp 0.41))
 )
```

```
(defun KS1-activation ()
  (setf KS1D1-temp (fetch D1))
  (setf KS1D2-temp (fetch D2))
  (setf KS1-D2 '())
  (setf KS1-D1 '())

  (KS1-update)
  )

(defun KS1-verify ()
  (and KS1-D2 KS1-D1))

(defun KS1-func ()
  (print "call to knowledge source handler")
  (ks1-exec)
  (update D3 KS1D3-temp)
  (update D4 KS1D4-temp)
  (update D5 KS1D5-temp)
  )

(defun ks1-exec ()
  (setf KS1D3-temp (max KS1D1-temp KS1D2-temp))
  (setf KS1D4-temp (* KS1D1-temp KS1D2-temp))
  (setf KS1D5-temp (/ KS1D1-temp KS1D2-temp))
  )

;;;**********************************
;;;* Build Blackboard Data Objects *
;;;**********************************

(defun build-do-list ()

(setf d/o-list '())

(setf D5 (make-instance 'blackboard_data_object
    :name "D5"
    :type "REAL"
    :value UNDEFINED
    :lock D5-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS4)
    :ic_list '(KS4-D5)
    :output_list '(KS1)))

(push D5 d/o-list)

(setf D4 (make-instance 'blackboard_data_object
    :name "D4"
```

```
    :type "REAL"
    :value UNDEFINED
    :lock D4-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D4)
    :output_list '(KS1)))

(push D4 d/o-list)

(setf D3 (make-instance 'blackboard_data_object
    :name "D3"
    :type "REAL"
    :value UNDEFINED
    :lock D3-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS2)
    :ic_list '(KS2-D3)
    :output_list '(KS1)))

(push D3 d/o-list)

(setf D2 (make-instance 'blackboard_data_object
    :name "D2"
    :type "REAL"
    :value 1.8
    :lock D2-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS1)
    :ic_list '(KS1-D2)
    :output_list 'NIL))

(push D2 d/o-list)

(setf D1 (make-instance 'blackboard_data_object
    :name "D1"
    :type "REAL"
    :value 2.2
    :lock D1-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS1)
    :ic_list '(KS1-D1)
    :output_list 'NIL))

(push D1 d/o-list)
```

```
(setf D9 (make-instance 'blackboard_data_object
    :name "D9"
    :type "REAL"
    :value UNDEFINED
    :lock D9-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS2)
    :ic_list '(KS2-D9)
    :output_list '(KS4)))

(push D9 d/o-list)

(setf D6 (make-instance 'blackboard_data_object
    :name "D6"
    :type "REAL"
    :value UNDEFINED
    :lock D6-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D6)
    :output_list '(KS2)))

(push D6 d/o-list)

(setf D7 (make-instance 'blackboard_data_object
    :name "D7"
    :type "REAL"
    :value UNDEFINED
    :lock D7-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS4)
    :ic_list '(KS4-D7)
    :output_list '(KS2)))

(push D7 d/o-list)

(setf D8 (make-instance 'blackboard_data_object
    :name "D8"
    :type "REAL"
    :value UNDEFINED
    :lock D8-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS4)
    :ic_list '(KS4-D8)
```

```
   :output_list '(KS3)))

(push D8 d/o-list)
   )
```

D.1.1 Simulation Results for B_1

List of Knowledge sources:

Knowledge Source: KS4
Type: PROCESSOR
Verify Function (KS4-VERIFY)
Activation Function (KS4-ACTIVATION)
Execution Function (KS4-FUNC)
Execution Delay: 8
Update Interval: 0.0

Knowledge Source: KS3
Type: PROCESSOR
Verify Function (KS3-VERIFY)
Activation Function (KS3-ACTIVATION)
Execution Function (KS3-FUNC)
Execution Delay: 6
Update Interval: 0.0

Knowledge Source: KS2
Type: PROCESSOR
Verify Function (KS2-VERIFY)
Activation Function (KS2-ACTIVATION)
Execution Function (KS2-FUNC)
Execution Delay: 4
Update Interval: 0.0

Knowledge Source: KS1
Type: SENSOR
Verify Function (KS1-VERIFY)
Activation Function (KS1-ACTIVATION)
Execution Function (KS1-FUNC)
Execution Delay: 5
Update Interval: 12

B_1 Execution:

KS KS1 executed at 5
Sensor KS1 activated at 12
KS KS1 executed at 17
Sensor KS1 activated at 24
KS KS1 executed at 29
Sensor KS1 activated at 36

KS KS1 executed at 41
Sensor KS1 activated at 48
KS KS1 executed at 53
Sensor KS1 activated at 60
KS KS1 executed at 65
Sensor KS1 activated at 72
KS KS1 executed at 77
Sensor KS1 activated at 84
KS KS1 executed at 89
Sensor KS1 activated at 96
KS KS1 executed at 101
Sensor KS1 activated at 108
KS KS1 executed at 113
Sensor KS1 activated at 120
KS KS1 executed at 125
Sensor KS1 activated at 132
KS KS1 executed at 137
Sensor KS1 activated at 144
KS KS1 executed at 149
Sensor KS1 activated at 156
KS KS1 executed at 161
Data Object: D8
Type: REAL
Value undefined
Times Read Locked 0
Time Write Locked 0

B_1 Results:

Data Object: D7
Type: REAL
Value undefined
Times Read Locked 0
Time Write Locked 0

Data Object: D6
Type: REAL
Value undefined
Times Read Locked 0
Time Write Locked 0

Data Object: D9
Type: REAL
Value undefined
Times Read Locked 0
Time Write Locked 0

Data Object: D1
Type: REAL
Value 13

Times Read Locked 14
Time Write Locked 13

Data Object: D2
Type: REAL
Value 13
Times Read Locked 14
Time Write Locked 13

Data Object: D3
Type: REAL
Value 13
Times Read Locked 0
Time Write Locked 14

Data Object: D4
Type: REAL
Value 169
Times Read Locked 0
Time Write Locked 14

Data Object: D5
Type: REAL
Value 1
Times Read Locked 0
Time Write Locked 14

The simulation results show that B_1 livelocked due to the two feedback loops. Sensor ks_1 executed and read all ot its input stream. When the input stream was empty B_1 stopped execution. The simulation results are the same as the results acheived by hand executing system B_1, and the COBS generated software matched the design specification.

D.2 Simulation Software and Results for B_2

This section contains the B_2 software generated by the COBS simulation tool and simulation runs for B_2

D.2.1 COBS Simulation Software for B_2

```
;;; -*- Syntax: Common-Lisp; Package: COMMON-LISP-USER; Base: 10; Mode:
LISP -*-

;;;**********************
;;;* Load the Simulation *
;;;* Support Functions   *
```

```
;;;**********************

(si:load "m:>john>sim-base")

;;;*********************
;;;* Define Variables *
;;;*********************

(defvar d/o-list '())

(defvar k/s-list '())

(defvar ks-list  '(KS4 KS3 KS2 KS1))

(defvar undefined '"undefined")
(defvar D8-LOCK '())
(defvar D8 '())
(defvar KS4-D8 '())
(defvar D7-LOCK '())
(defvar D7 '())
(defvar KS4-D7 '())
(defvar D6-LOCK '())
(defvar D6 '())
(defvar KS3-D6 '())
(defvar D9-LOCK '())
(defvar D9 '())
(defvar KS2-D9 '())
(defvar D1-LOCK '())
(defvar D1 '())
(defvar KS1-D1 '())
(defvar D2-LOCK '())
(defvar D2 '())
(defvar KS1-D2 '())
(defvar D3-LOCK '())
(defvar D3 '())
(defvar KS2-D3 '())
(defvar D4-LOCK '())
(defvar D4 '())
(defvar KS3-D4 '())
(defvar D5-LOCK '())
(defvar D5 '())
(defvar KS4-D5 '())

;;;*****************************************
;;;* Define Blackboard Data Object Locks *
;;;*****************************************

(Setf D5-LOCK (process:make-lock "D5-LOCK" :type :multiple-reader-single-writer
:recursive t))
```

```
(Setf D4-LOCK (process:make-lock "D4-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D3-LOCK (process:make-lock "D3-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D2-LOCK (process:make-lock "D2-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D1-LOCK (process:make-lock "D1-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D9-LOCK (process:make-lock "D9-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D6-LOCK (process:make-lock "D6-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D7-LOCK (process:make-lock "D7-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D8-LOCK (process:make-lock "D8-LOCK" :type :multiple-reader-single-writer
:recursive t))

;;;**********************************
;;;* Build Knowledge Source Objects *
;;;**********************************

(defun build-ks-list ()

(setf k/s-list '())

(setf KS1 (make-instance 'base_ks
    :name "KS1"
    :type "SENSOR"
    :init-function '(KS1-verify)
    :act-function '(KS1-activation)
    :exec-function '(KS1-func)
    :execution-delay 5
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS1 k/s-list)

(setf KS2 (make-instance 'base_ks
    :name "KS2"
    :type "PROCESSOR"
    :init-function '(KS2-verify)
    :act-function '(KS2-activation)
    :exec-function '(KS2-func)
    :execution-delay 4
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))
```

```
(push KS2 k/s-list)

(setf KS3 (make-instance 'base_ks
    :name "KS3"
    :type "PROCESSOR"
    :init-function '(KS3-verify)
    :act-function '(KS3-activation)
    :exec-function '(KS3-func)
    :execution-delay 6
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS3 k/s-list)

(setf KS4 (make-instance 'base_ks
    :name "KS4"
    :type "PROCESSOR"
    :init-function '(KS4-verify)
    :act-function '(KS4-activation)
    :exec-function '(KS4-func)
    :execution-delay 8
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS4 k/s-list)

   )

;;;*************************************
;;;* Define Knowledge Source Processess *
;;;*************************************

(defun KS4-activation ()
   (setf KS4D5-temp (fetch D5))
   (setf KS4D7-temp (fetch D7))
   (setf KS4D8-temp (fetch D8))
   (setf KS4-D8 '())
   (setf KS4-D7 '())
   (setf KS4-D5 '())

   )

(defun KS4-verify ()
    (and KS4-D8 KS4-D7 KS4-D5 (NOT (= (GET-VALUE D9) 13.6)) (NOT (=
(GET-VALUE D2) 0.0)) (NOT (= (GET-VALUE D8) 0.0))))

(defun KS4-func ()
```

```
    (print "call to knowledge source handler")
    (ks4-exec)
    (update D9 KS4D9-temp)
)

(defun KS3-activation ()
   (setf KS3D4-temp (fetch D4))
   (setf KS3D6-temp (fetch D6))
   (setf KS3-D6 '())
   (setf KS3-D4 '())

   )

(defun KS3-verify ()
    (and KS3-D6 KS3-D4 (NOT (= (GET-VALUE D8) PI)) (NOT (= (GET-VALUE
D9) 13.6))))

(defun KS3-func ()
    (print "call to knowledge source handler")
    (ks3-exec)
    (update D8 KS3D8-temp)
)

(defun KS2-activation ()
   (setf KS2D3-temp (fetch D3))
   (setf KS2-D9 '())
   (setf KS2-D3 '())

   )

(defun KS2-verify ()
    (and KS2-D3 (NOT (= (GET-VALUE D4) 7)) (NOT (= (GET-VALUE D8)
PI))))

(defun KS2-func ()
    (print "call to knowledge source handler")
    (ks2-exec)
    (update D6 KS2D6-temp)
    (update D7 KS2D7-temp)
)

(defun KS1-activation ()
   (setf KS1D1-temp (fetch D1))
   (setf KS1D2-temp (fetch D2))
   (setf KS1-D2 '())
   (setf KS1-D1 '())

   (KS1-update)
   )
```

```
(defun KS1-verify ()
    (and KS1-D2 KS1-D1))

(defun KS1-func ()
    (print "call to knowledge source handler")
    (ks1-exec)
    (update D3 KS1D3-temp)
    (update D4 KS1D4-temp)
    (update D5 KS1D5-temp)
)

;;;**********************************
;;;* Build Blackboard Data Objects *
;;;**********************************

(defun build-do-list ()

(setf d/o-list '())

(setf D5 (make-instance 'blackboard_data_object
    :name "D5"
    :type "REAL"
    :value UNDEFINED
    :lock D5-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS4)
    :ic_list '(KS4-D5)
    :output_list '(KS1)))

(push D5 d/o-list)

(setf D4 (make-instance 'blackboard_data_object
    :name "D4"
    :type "REAL"
    :value UNDEFINED
    :lock D4-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D4)
    :output_list '(KS1)))

(push D4 d/o-list)

(setf D3 (make-instance 'blackboard_data_object
    :name "D3"
    :type "REAL"
```

```
    :value UNDEFINED
    :lock D3-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS2)
    :ic_list '(KS2-D3)
    :output_list '(KS1)))

(push D3 d/o-list)

(setf D2 (make-instance 'blackboard_data_object
    :name "D2"
    :type "REAL"
    :value 1.8
    :lock D2-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS1)
    :ic_list '(KS1-D2)
    :output_list 'NIL))

(push D2 d/o-list)

(setf D1 (make-instance 'blackboard_data_object
    :name "D1"
    :type "REAL"
    :value 2.2
    :lock D1-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS1)
    :ic_list '(KS1-D1)
    :output_list 'NIL))

(push D1 d/o-list)

(setf D9 (make-instance 'blackboard_data_object
    :name "D9"
    :type "REAL"
    :value undefined
    :lock D9-LOCK
    :read-lock 0
    :write-lock 0
    :input_list 'NIL
    :ic_list '(KS2-D9)
    :output_list '(KS4)))

(push D9 d/o-list)
```

```
(setf D6 (make-instance 'blackboard_data_object
    :name "D6"
    :type "REAL"
    :value UNDEFINED
    :lock D6-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D6)
    :output_list '(KS2)))

(push D6 d/o-list)

(setf D7 (make-instance 'blackboard_data_object
    :name "D7"
    :type "REAL"
    :value UNDEFINED
    :lock D7-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS4)
    :ic_list '(KS4-D7)
    :output_list '(KS2)))

(push D7 d/o-list)

(setf D8 (make-instance 'blackboard_data_object
    :name "D8"
    :type "REAL"
    :value UNDEFINED
    :lock D8-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS4)
    :ic_list '(KS4-D8)
    :output_list '(KS3)))

(push D8 d/o-list)

   )

(defun ks4-exec ()
  (setf KS4D9-temp (max KS4D5-temp KS4D7-temp KS4D8-temp))
  )

(defun ks3-exec ()
  (setf KS3D8-temp (max KS3D4-temp KS3D6-temp))
  )
```

```
(defun ks2-exec ()
 (setf KS2D6-temp (max KS2D3-temp 4.10))
 (setf KS2D7-temp (* KS2D3-temp 0.41))
 )

(defun ks1-exec ()
 (setf KS1D3-temp (max KS1D1-temp KS1D2-temp))
 (setf KS1D4-temp (* KS1D1-temp KS1D2-temp))
 (setf KS1D5-temp (/ KS1D1-temp KS1D2-temp))
 )
```

D.2.2 COBS Simulation Software for B_2

List of Knowledge Sources:

Knowledge Source: KS4
Type: PROCESSOR
Verify Function (KS4-VERIFY)
Activation Function (KS4-ACTIVATION)
Execution Function (KS4-FUNC)
Execution Delay: 8
Update Interval: 0.0

Knowledge Source: KS3
Type: PROCESSOR
Verify Function (KS3-VERIFY)
Activation Function (KS3-ACTIVATION)
Execution Function (KS3-FUNC)
Execution Delay: 6
Update Interval: 0.0

Knowledge Source: KS2
Type: PROCESSOR
Verify Function (KS2-VERIFY)
Activation Function (KS2-ACTIVATION)
Execution Function (KS2-FUNC)
Execution Delay: 4
Update Interval: 0.0

Knowledge Source: KS1
Type: SENSOR
Verify Function (KS1-VERIFY)
Activation Function (KS1-ACTIVATION)
Execution Function (KS1-FUNC)
Execution Delay: 5
Update Interval: 12

B_2 Execution:

Sensor KS1 activated at 0
Event queue (KS1)
KS1 executed at 5
Event queue (KS2)
KS2 executed at 9
Sensor KS1 activated at 12
Event queue (KS1 KS3)
KS3 executed at 15
Event queue (KS1 KS4)
KS1 executed at 17
Event queue (KS4 KS2)
KS2 executed at 21
Event queue (KS4 KS3)
KS4 executed at 23
Sensor KS1 activated at 24
Event queue (KS1 KS3)
KS3 executed at 27
Event queue (KS1 KS4)
KS1 executed at 29
Event queue (KS4 KS2)
KS2 executed at 33
Event queue (KS4 KS3)
KS4 executed at 35
Sensor KS1 activated at 36
Event queue (KS1 KS3)
KS3 executed at 39
Event queue (KS1 KS4)
KS1 executed at 41
Event queue (KS4 KS2)
KS2 executed at 45
Event queue (KS4 KS3)
KS4 executed at 47
Sensor KS1 activated at 48
Event queue (KS1 KS3)
KS3 executed at 51
Event queue (KS1 KS4)
KS1 executed at 53
Event queue (KS4 KS2)
KS2 executed at 57
Event queue (KS4 KS3)
KS4 executed at 59
Sensor KS1 activated at 60
Event queue (KS1 KS3)
KS3 executed at 63
Event queue (KS1 KS4)
KS1 executed at 65
Event queue (KS4 KS2)

KS2 executed at 69
Event queue (KS4 KS3)
KS4 executed at 71
Sensor KS1 activated at 72
Event queue (KS1 KS3)
KS3 executed at 75
Event queue (KS1 KS4)
KS1 executed at 77
Event queue (KS4 KS2)
KS2 executed at 81
Event queue (KS4 KS3)
KS4 executed at 83
Sensor KS1 activated at 84
Event queue (KS1 KS3)
KS3 executed at 87
Event queue (KS1 KS4)
KS1 executed at 89
Event queue (KS4 KS2)
KS2 executed at 93
Event queue (KS4 KS3)
KS4 executed at 95
Sensor KS1 activated at 96
Event queue (KS1 KS3)
KS3 executed at 99
Event queue (KS1 KS4)
KS1 executed at 101
Event queue (KS4 KS2)
KS2 executed at 105
Event queue (KS4 KS3)
KS4 executed at 107
Sensor KS1 activated at 108
Event queue (KS1 KS3)
KS3 executed at 111
Event queue (KS1 KS4)
KS1 executed at 113
Event queue (KS4 KS2)
KS2 executed at 117
Event queue (KS4 KS3)
KS4 executed at 119
Sensor KS1 activated at 120
Event queue (KS1 KS3)
KS3 executed at 123
Event queue (KS1 KS4)
KS1 executed at 125
Event queue (KS4 KS2)
KS2 executed at 129
Event queue (KS4 KS3)
KS4 executed at 131
Sensor KS1 activated at 132

Event queue (KS1 KS3)
KS3 executed at 135
Event queue (KS1 KS4)
KS1 executed at 137
Event queue (KS4 KS2)
KS2 executed at 141
Event queue (KS4 KS3)
KS4 executed at 143
Sensor KS1 activated at 144
Event queue (KS1 KS3)
KS3 executed at 147
Event queue (KS1 KS4)
KS1 executed at 149
Event queue (KS4 KS2)
KS2 executed at 153
Event queue (KS4 KS3)
KS4 executed at 155
Sensor KS1 activated at 156
Event queue (KS1 KS3)
KS3 executed at 159
Event queue (KS1 KS4)
KS1 executed at 161
Event queue (KS4 KS2)
KS2 executed at 165
Event queue (KS4 KS3)
KS4 executed at 167
Event queue (KS3)
KS3 executed at 171
Event queue (KS4)
KS4 executed at 179

B_2 Results:

Data Object: D8
Type: REAL
Value 169
Times Read Locked 14
Time Write Locked 14

Data Object: D7
Type: REAL
Value 5.33
Times Read Locked 14
Time Write Locked 14

Data Object: D6
Type: REAL
Value 13
Times Read Locked 14
Time Write Locked 14

Data Object: D9
Type: REAL
Value 169
Times Read Locked 0
Time Write Locked 14

Data Object: D1
Type: REAL
Value 13
Times Read Locked 14
Time Write Locked 13

Data Object: D2
Type: REAL
Value 13
Times Read Locked 14
Time Write Locked 13

Data Object: D3
Type: REAL
Value 13
Times Read Locked 14
Time Write Locked 14

Data Object: D4
Type: REAL
Value 169
Times Read Locked 14
Time Write Locked 14

Data Object: D5
Type: REAL
Value 1
Times Read Locked 14
Time Write Locked 14

The simulation results show that B_2 executed to completion. Sensor ks_1 executed and read all ot its input stream. When the input stream was empty B_2 stopped execution. The simulation results are the same as the results acheived by hand executing system B_2, and the COBS generated software matched the design specification.

D.3 Simulation Software and Results for B_3

This section contains the B_3 software generated by the COBS simulation tool and simulation runs for B_3

D.3.1 COBS Simulation Software for B_3

```
;;; -*- Syntax: Common-Lisp; Package: COMMON-LISP-USER; Base: 10; Mode:
LISP -*-

;;;************************
;;;* Load the Simulation *
;;;* Support Functions   *
;;;************************
;;;

(si:load "m:>john>sim-base")

;;;*********************
;;;* Define Variables *
;;;*********************
;;;

(defvar d/o-list '())

(defvar k/s-list '())

(defvar ks-list  '(KS1 KS2 KS3 KS4))

(defvar undefined '"undefined")
(defvar D1-LOCK '())
(defvar D1 '())
(defvar KS1-D1 '())
(defvar D2-LOCK '())
(defvar D2 '())
(defvar KS1-D2 '())
(defvar D3-LOCK '())
(defvar D3 '())
(defvar KS2-D3 '())
(defvar D4-LOCK '())
(defvar D4 '())
(defvar KS3-D4 '())
(defvar D5-LOCK '())
(defvar D5 '())
(defvar KS4-D5 '())
(defvar D6-LOCK '())
(defvar D6 '())
(defvar KS3-D6 '())
(defvar D7-LOCK '())
(defvar D7 '())
(defvar KS4-D7 '())
(defvar D8-LOCK '())
(defvar D8 '())
(defvar KS4-D8 '())
(defvar D9-LOCK '())
(defvar D9 '())
```

```
;;;*************************************
;;;* Define Blackboard Data Object Locks *
;;;*************************************

(Setf D9-LOCK (process:make-lock "D9-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D8-LOCK (process:make-lock "D8-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D7-LOCK (process:make-lock "D7-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D6-LOCK (process:make-lock "D6-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D5-LOCK (process:make-lock "D5-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D4-LOCK (process:make-lock "D4-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D3-LOCK (process:make-lock "D3-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D2-LOCK (process:make-lock "D2-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D1-LOCK (process:make-lock "D1-LOCK" :type :multiple-reader-single-writer
:recursive t))

;;;*********************************
;;;* Build Knowledge Source Objects *
;;;*********************************

(defun build-ks-list ()

(setf k/s-list '())

(setf KS4 (make-instance 'base_ks
    :name "KS4"
    :type "PROCESSOR"
    :init-function '(KS4-verify)
    :act-function '(KS4-activation)
    :exec-function '(KS4-func)
    :execution-delay 3
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS4 k/s-list)

(setf KS3 (make-instance 'base_ks
    :name "KS3"
    :type "PROCESSOR"
    :init-function '(KS3-verify)
```

```
    :act-function '(KS3-activation)
    :exec-function '(KS3-func)
    :execution-delay 5
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS3 k/s-list)

(setf KS2 (make-instance 'base_ks
    :name "KS2"
    :type "PROCESSOR"
    :init-function '(KS2-verify)
    :act-function '(KS2-activation)
    :exec-function '(KS2-func)
    :execution-delay 3
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS2 k/s-list)

(setf KS1 (make-instance 'base_ks
    :name "KS1"
    :type "SENSOR"
    :init-function '(KS1-verify)
    :act-function '(KS1-activation)
    :exec-function '(KS1-func)
    :execution-delay 4
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS1 k/s-list)

   )

;;;*************************************
;;;* Define Knowledge Source Processess *
;;;*************************************

(defun KS1-activation ()
 (setf KS1D1-temp (fetch D1))
 (setf KS1D2-temp (fetch D2))
 (setf KS1-D2 '())
 (setf KS1-D1 '())
 (KS1-update)
 )
```

```
(defun KS1-verify ()
  (and KS1-D2 KS1-D1))

(defun KS1-func ()
  (print "call to knowledge source handler")
  (ks1-exec)
  (update D3 KS1D3-temp)
  (update D4 KS1D4-temp)
  (update D5 KS1D5-temp)
  (update D6 KS1D6-temp)
)

(defun KS2-activation ()
  (setf KS2D3-temp (fetch D3))
  (setf KS2-D3 '())
  )

(defun KS2-verify ()
  (and KS2-D3 (NOT (= (GET-VALUE D4) 7))))

(defun KS2-func ()
  (print "call to knowledge source handler")
  (ks2-exec)
  (update D7 KS2D7-temp)
)

(defun KS3-activation ()
  (setf KS3D4-temp (fetch D4))
  (setf KS3D6-temp (fetch D6))
  (setf KS3-D6 '())
  (setf KS3-D4 '())
  )

(defun KS3-verify ()
  (and KS3-D6 KS3-D4 (NOT (= (GET-VALUE D8) PI))
      (NOT (= (GET-VALUE D9) 13.6))))

(defun KS3-func ()
  (print "call to knowledge source handler")
  (ks3-exec)
  (update D8 KS3D8-temp)
)

(defun KS4-activation ()
  (setf KS4D5-temp (fetch D5))
  (setf KS4D7-temp (fetch D7))
  (setf KS4D8-temp (fetch D8))
  (setf KS4-D8 '())
  (setf KS4-D7 '())
```

```
 (setf KS4-D5 '())
 )

(defun KS4-verify ()
 (and KS4-D8 KS4-D7 KS4-D5 (NOT (= (GET-VALUE D9) 13.6))
    (NOT (= (GET-VALUE D2) 0.0)) (NOT (= (GET-VALUE D8) 0.0))))

(defun KS4-func ()
 (print "call to knowledge source handler")
 (ks4-exec)
 (update D9 KS4D9-temp)
)

;;;**********************************
;;;* Build Blackboard Data Objects *
;;;**********************************

(defun build-do-list ()

(setf d/o-list '())

(setf D9 (make-instance 'blackboard_data_object
   :name "D9"
   :type "NIL"
   :value undefined
   :lock D9-LOCK
   :read-lock 0
   :write-lock 0
   :input_list 'NIL
   :ic_list 'NIL
   :output_list '(KS4)))

(push D9 d/o-list)

(setf D8 (make-instance 'blackboard_data_object
   :name "D8"
   :type "NIL"
   :value undefined
   :lock D8-LOCK
   :read-lock 0
   :write-lock 0
   :input_list '(KS4)
   :ic_list '(KS4-D8)
   :output_list '(KS3)))

(push D8 d/o-list)

(setf D7 (make-instance 'blackboard_data_object
   :name "D7"
```

```
    :type "NIL"
    :value undefined
    :lock D7-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS4)
    :ic_list '(KS4-D7)
    :output_list '(KS2)))

(push D7 d/o-list)

(setf D6 (make-instance 'blackboard_data_object
    :name "D6"
    :type "NIL"
    :value undefined
    :lock D6-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D6)
    :output_list '(KS1)))

(push D6 d/o-list)

(setf D5 (make-instance 'blackboard_data_object
    :name "D5"
    :type "NIL"
    :value undefined
    :lock D5-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS4)
    :ic_list '(KS4-D5)
    :output_list '(KS1)))

(push D5 d/o-list)

(setf D4 (make-instance 'blackboard_data_object
    :name "D4"
    :type "NIL"
    :value undefined
    :lock D4-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D4)
    :output_list '(KS1)))

(push D4 d/o-list)
```

```
(setf D3 (make-instance 'blackboard_data_object
    :name "D3"
    :type "NIL"
    :value undefined
    :lock D3-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS2)
    :ic_list '(KS2-D3)
    :output_list '(KS1)))

(push D3 d/o-list)

(setf D2 (make-instance 'blackboard_data_object
    :name "D2"
    :type "REAL"
    :value 14.0
    :lock D2-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS1)
    :ic_list '(KS1-D2)
    :output_list 'NIL))

(push D2 d/o-list)

(setf D1 (make-instance 'blackboard_data_object
    :name "D1"
    :type "REAL"
    :value 12
    :lock D1-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS1)
    :ic_list '(KS1-D1)
    :output_list 'NIL))

(push D1 d/o-list)

   )

;;;*******************************
;;;* Add User Defined Execution *
;;;* Functions                  *
;;;*******************************

(defun ks4-exec ()
  (setf KS4D9-temp (max KS4D5-temp KS4D7-temp KS4D8-temp))
```

```
 )

(defun ks3-exec ()
 (setf KS3D8-temp (max KS3D4-temp KS3D6-temp))
 )

(defun ks2-exec ()
 (setf KS2D7-temp (* KS2D3-temp 0.41))
 )

(defun ks1-exec ()
 (setf KS1D3-temp (max KS1D1-temp KS1D2-temp))
 (setf KS1D4-temp (* KS1D1-temp KS1D2-temp))
 (setf KS1D5-temp (/ KS1D1-temp KS1D2-temp))
 (setf KS1D6-temp (max KS1D3-temp 4.10))
 )
```

This section contains the B1 software generated by the COBS simulation tool and simulation runs for B1

D.3.2 COBS Simulation Software for B_3

List of Knowledge Sources:

Knowledge Source: KS1
Type: SENSOR
Verify Function (KS1-VERIFY)
Activation Function (KS1-ACTIVATION)
Execution Function (KS1-FUNC)
Execution Delay: 4
Update Interval: 12

Knowledge Source: KS2
Type: PROCESSOR
Verify Function (KS2-VERIFY)
Activation Function (KS2-ACTIVATION)
Execution Function (KS2-FUNC)
Execution Delay: 3
Update Interval: 0.0

Knowledge Source: KS3
Type: PROCESSOR
Verify Function (KS3-VERIFY)
Activation Function (KS3-ACTIVATION)
Execution Function (KS3-FUNC)
Execution Delay: 5
Update Interval: 0.0

Knowledge Source: KS4
Type: PROCESSOR
Verify Function (KS4-VERIFY)
Activation Function (KS4-ACTIVATION)
Execution Function (KS4-FUNC)
Execution Delay: 3
Update Interval: 0.0

B_3 Execution:

Event queue (KS1)
KS1 executed at 4
Event queue (KS2 KS3)
KS2 executed at 7
Event queue (KS3)
KS3 executed at 9
Sensor KS1 activated at 12
Event queue (KS1 KS4)
KS4 executed at 12
Event queue (KS1)
KS1 executed at 16
Event queue (KS2 KS3)
KS2 executed at 19
Event queue (KS3)
KS3 executed at 21
Sensor KS1 activated at 24
Event queue (KS1 KS4)
KS4 executed at 24
Event queue (KS1)
KS1 executed at 28
Event queue (KS2 KS3)
KS2 executed at 31
Event queue (KS3)
KS3 executed at 33
Sensor KS1 activated at 36
Event queue (KS1 KS4)
KS4 executed at 36
Event queue (KS1)
KS1 executed at 40
Event queue (KS2 KS3)
KS2 executed at 43
Event queue (KS3)
KS3 executed at 45
Sensor KS1 activated at 48
Event queue (KS1 KS4)
KS4 executed at 48
Event queue (KS1)
KS1 executed at 52
Event queue (KS2 KS3)

KS2 executed at 55
Event queue (KS3)
KS3 executed at 57
Sensor KS1 activated at 60
Event queue (KS1 KS4)
KS4 executed at 60
Event queue (KS1)
KS1 executed at 64
Event queue (KS2 KS3)
KS2 executed at 67
Event queue (KS3)
KS3 executed at 69
Sensor KS1 activated at 72
Event queue (KS1 KS4)
KS4 executed at 72
Event queue (KS1)
KS1 executed at 76
Event queue (KS2 KS3)
KS2 executed at 79
Event queue (KS3)
KS3 executed at 81
Sensor KS1 activated at 84
Event queue (KS1 KS4)
KS4 executed at 84
Event queue (KS1)
KS1 executed at 88
Event queue (KS2 KS3)
KS2 executed at 91
Event queue (KS3)
KS3 executed at 93
Sensor KS1 activated at 96
Event queue (KS1 KS4)
KS4 executed at 96
Event queue (KS1)
KS1 executed at 100
Event queue (KS2 KS3)
KS2 executed at 103
Event queue (KS3)
KS3 executed at 105
Sensor KS1 activated at 108
Event queue (KS1 KS4)
KS4 executed at 108
Event queue (KS1)
KS1 executed at 112
Event queue (KS2 KS3)
KS2 executed at 115
Event queue (KS3)
KS3 executed at 117
Sensor KS1 activated at 120

Event queue (KS1 KS4)
KS4 executed at 120
Event queue (KS1)
KS1 executed at 124
Event queue (KS2 KS3)
KS2 executed at 127
Event queue (KS3)
KS3 executed at 129
Sensor KS1 activated at 132
Event queue (KS1 KS4)
KS4 executed at 132
Event queue (KS1)
KS1 executed at 136
Event queue (KS2 KS3)
KS2 executed at 139
Event queue (KS3)
KS3 executed at 141
Sensor KS1 activated at 144
Event queue (KS1 KS4)
KS4 executed at 144
Event queue (KS1)
KS1 executed at 148
Event queue (KS2 KS3)
KS2 executed at 151
Event queue (KS3)
KS3 executed at 153
Sensor KS1 activated at 156
Event queue (KS1 KS4)
KS4 executed at 156
Event queue (KS1)
KS1 executed at 160
Event queue (KS2 KS3)
KS2 executed at 163
Event queue (KS3)
KS3 executed at 165
Event queue (KS4)
KS4 executed at 168

B_3 Results:

Data Object: D1
Type: REAL
Value 13
Times Read Locked 14
Time Write Locked 13

Data Object: D2
Type: REAL
Value 13

Times Read Locked 14
Time Write Locked 13

Data Object: D3
Type: NIL
Value 13
Times Read Locked 14
Time Write Locked 14

Data Object: D4
Type: NIL
Value 169
Times Read Locked 14
Time Write Locked 14

Data Object: D5
Type: NIL
Value 1
Times Read Locked 14
Time Write Locked 14

Data Object: D6
Type: NIL
Value 13
Times Read Locked 14
Time Write Locked 14

Data Object: D7
Type: NIL
Value 5.33
Times Read Locked 14
Time Write Locked 14

Data Object: D8
Type: NIL
Value 169
Times Read Locked 14
Time Write Locked 14

Data Object: D9
Type: NIL
Value 169
Times Read Locked 0
Time Write Locked 14

The simulation results show that B_3 executed to completion. Sensor ks_1 executed and read all ot its input stream. When the input stream was empty B_3 stopped execution. The simulation results are the same as the results acheived by hand

executing system B_3, and the COBS generated software matched the design specification.

D.4 Simulation Software and Results for B_4

This section contains the B_3 software generated by the COBS simulation tool and simulation runs for B_3

D.4.1 COBS Simulation Software for B_4

```
;;; -*- Syntax: Common-Lisp; Package: COMMON-LISP-USER; Base: 10; Mode:
LISP -*-

;;;************************
;;;* Load the Simulation *
;;;* Support Functions   *
;;;************************

(si:load "m:>john>sim-base")

;;;*********************
;;;* Define Variables *
;;;*********************

(defvar d/o-list '())

(defvar k/s-list '())

(defvar ks-list  '(KS1 KS2 KS3))

(defvar undefined '"undefined")
(defvar D1-LOCK '())
(defvar D1 '())
(defvar KS1-D1 '())
(defvar D2-LOCK '())
(defvar D2 '())
(defvar KS1-D2 '())
(defvar D3-LOCK '())
(defvar D3 '())
(defvar KS3-D3 '())
(defvar D4-LOCK '())
(defvar D4 '())
(defvar KS2-D4 '())
(defvar D5-LOCK '())
(defvar D5 '())
(defvar KS3-D5 '())
(defvar D6-LOCK '())
```

```
(defvar D6 '())
(defvar KS2-D6 '())
(defvar D8-LOCK '())
(defvar D8 '())
(defvar KS3-D8 '())
(defvar D9-LOCK '())
(defvar D9 '())

;;;*****************************************
;;;* Define Blackboard Data Object Locks *
;;;*****************************************

(Setf D9-LOCK (process:make-lock "D9-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D8-LOCK (process:make-lock "D8-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D6-LOCK (process:make-lock "D6-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D5-LOCK (process:make-lock "D5-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D4-LOCK (process:make-lock "D4-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D3-LOCK (process:make-lock "D3-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D2-LOCK (process:make-lock "D2-LOCK" :type :multiple-reader-single-writer
:recursive t))
(Setf D1-LOCK (process:make-lock "D1-LOCK" :type :multiple-reader-single-writer
:recursive t))

;;;***********************************
;;;* Build Knowledge Source Objects *
;;;***********************************

(defun build-ks-list ()

(setf k/s-list '())

(setf KS3 (make-instance 'base_ks
    :name "KS3"
    :type "PROCESSOR"
    :init-function '(KS3-verify)
    :act-function '(KS3-activation)
    :exec-function '(KS3-func)
    :execution-delay 6
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS3 k/s-list)
```

```
(setf KS2 (make-instance 'base_ks
    :name "KS2"
    :type "PROCESSOR"
    :init-function '(KS2-verify)
    :act-function '(KS2-activation)
    :exec-function '(KS2-func)
    :execution-delay 4
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS2 k/s-list)

(setf KS1 (make-instance 'base_ks
    :name "KS1"
    :type "SENSOR"
    :init-function '(KS1-verify)
    :act-function '(KS1-activation)
    :exec-function '(KS1-func)
    :execution-delay 4
    :update-interval 0.0
    :completion-time 0
    :activation-time 0 ))

(push KS1 k/s-list)

   )

;;;****************************************
;;;* Define Knowledge Source Processess *
;;;****************************************

(defun KS1-activation ()
   (setf KS1D1-temp (fetch D1))
   (setf KS1D2-temp (fetch D2))
   (setf KS1-D2 '())
   (setf KS1-D1 '())
   (KS1-update)
   )

(defun KS1-verify ()
  (and KS1-D2 KS1-D1))

(defun KS1-func ()
  (print "call to knowledge source handler")
  (ks1-exec)
  (update D3 KS1D3-temp)
  (update D4 KS1D4-temp)
```

```
 (update D5 KS1D5-temp)
 (update D6 KS1D6-temp)
)

;;;*******************************
;;;* Add User Defined Execution *
;;;* Functions                  *
;;;*******************************

(defun ks1-exec ()
 (setf KS1D3-temp (max KS1D1-temp KS1D2-temp))
 (setf KS1D3-temp (* KS1D3-temp 0.41))
 (setf KS1D4-temp (* KS1D1-temp KS1D2-temp))
 (setf KS1D5-temp (/ KS1D1-temp KS1D2-temp))
 (setf KS1D6-temp (max KS1D3-temp 4.10))
 )

;;;*******************************

(defun KS2-activation ()
   (setf KS2D4-temp (fetch D4))
   (setf KS2D6-temp (fetch D6))
   (setf KS2-D6 '())
   (setf KS2-D4 '())
   )

(defun KS2-verify ()
    (and KS2-D6 KS2-D4 (NOT (= (GET-VALUE D8) PI)) (NOT (= (GET-VALUE
D9) 13.6))))

(defun KS2-func ()
 (print "call to knowledge source handler")
 (ks2-exec)
 (update D8 KS2D8-temp)
)

;;;*******************************
;;;* Add User Defined Execution *
;;;* Functions                  *
;;;*******************************

(defun ks2-exec ()
 (setf KS2D8-temp (max KS2D4-temp KS2D6-temp))
 )

(defun KS3-activation ()
   (setf KS3D3-temp (fetch D3))
   (setf KS3D5-temp (fetch D5))
   (setf KS3D8-temp (fetch D8))
```

```
   (setf KS3-D8 '())
   (setf KS3-D5 '())
   (setf KS3-D3 '())
   )

(defun KS3-verify ()
    (and KS3-D8 KS3-D5 KS3-D3 (NOT (= (GET-VALUE D4) 7)) (NOT (= (GET-
VALUE D2) 0.0)) (NOT (= (GET-VALUE D8) 0.0)) (NOT (= (GET-VALUE D9)
13.6))))

(defun KS3-func ()
 (print "call to knowledge source handler")
 (ks3-exec)
 (update D9 KS3D9-temp)
)

;;;*******************************
;;;* Add User Defined Execution *
;;;* Functions                  *
;;;*******************************

(defun ks3-exec ()
 (setf KS3D9-temp (max KS3D5-temp KS3D3-temp KS3D8-temp))
 )

;;;*********************************
;;;* Build Blackboard Data Objects *
;;;*********************************

(defun build-do-list ()

(setf d/o-list '())

(setf D9 (make-instance 'blackboard_data_object
   :name "D9"
   :type "NIL"
   :value undefined
   :lock D9-LOCK
   :read-lock 0
   :write-lock 0
   :input_list 'NIL
   :ic_list 'NIL
   :output_list '(KS3)))

(push D9 d/o-list)

(setf D8 (make-instance 'blackboard_data_object
   :name "D8"
   :type "NIL"
```

```
    :value undefined
    :lock D8-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D8)
    :output_list '(KS2)))

(push D8 d/o-list)

(setf D6 (make-instance 'blackboard_data_object
    :name "D6"
    :type "NIL"
    :value undefined
    :lock D6-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS2)
    :ic_list '(KS2-D6)
    :output_list '(KS1)))

(push D6 d/o-list)

(setf D5 (make-instance 'blackboard_data_object
    :name "D5"
    :type "NIL"
    :value undefined
    :lock D5-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D5)
    :output_list '(KS1)))

(push D5 d/o-list)

(setf D4 (make-instance 'blackboard_data_object
    :name "D4"
    :type "NIL"
    :value undefined
    :lock D4-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS2)
    :ic_list '(KS2-D4)
    :output_list '(KS1)))

(push D4 d/o-list)
```

```
(setf D3 (make-instance 'blackboard_data_object
    :name "D3"
    :type "NIL"
    :value undefined
    :lock D3-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS3)
    :ic_list '(KS3-D3)
    :output_list '(KS1)))

(push D3 d/o-list)

(setf D2 (make-instance 'blackboard_data_object
    :name "D2"
    :type "REAL"
    :value 14
    :lock D2-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS1)
    :ic_list '(KS1-D2)
    :output_list 'NIL))

(push D2 d/o-list)

(setf D1 (make-instance 'blackboard_data_object
    :name "D1"
    :type "REAL"
    :value 12
    :lock D1-LOCK
    :read-lock 0
    :write-lock 0
    :input_list '(KS1)
    :ic_list '(KS1-D1)
    :output_list 'NIL))

(push D1 d/o-list)

    )
```

This section contains the B1 software generated by the COBS simulation tool and simulation runs for B1

D.1.1 COBS Simulation Software for B_1

List of Knowledge Sources:

Knowledge Source: KS1

Type: SENSOR
Verify Function (KS1-VERIFY)
Activation Function (KS1-ACTIVATION)
Execution Function (KS1-FUNC)
Execution Delay: 4
Update Interval: 12

Knowledge Source: KS2
Type: PROCESSOR
Verify Function (KS2-VERIFY)
Activation Function (KS2-ACTIVATION)
Execution Function (KS2-FUNC)
Execution Delay: 4
Update Interval: 0.0

Knowledge Source: KS3
Type: PROCESSOR
Verify Function (KS3-VERIFY)
Activation Function (KS3-ACTIVATION)
Execution Function (KS3-FUNC)
Execution Delay: 6
Update Interval: 0.0

B_4 Execution:

Event queue (KS1)
KS1 executed at 4
Event queue (KS2)
KS2 executed at 8
Sensor KS1 activated at 12
Event queue (KS1 KS3)
KS3 executed at 14
Event queue (KS1)
KS1 executed at 16
Event queue (KS2)
KS2 executed at 20
Sensor KS1 activated at 24
Event queue (KS1 KS3)
KS3 executed at 26
Event queue (KS1)
KS1 executed at 28
Event queue (KS2)
KS2 executed at 32
Sensor KS1 activated at 36
Event queue (KS1 KS3)
KS3 executed at 38
Event queue (KS1)
KS1 executed at 40
Event queue (KS2)
KS2 executed at 44

Sensor KS1 activated at 48
Event queue (KS1 KS3)
KS3 executed at 50
Event queue (KS1)
KS1 executed at 52
Event queue (KS2)
KS2 executed at 56
Sensor KS1 activated at 60
Event queue (KS1 KS3)
KS3 executed at 62
Event queue (KS1)
KS1 executed at 64
Event queue (KS2)
KS2 executed at 68
Sensor KS1 activated at 72
Event queue (KS1 KS3)
KS3 executed at 74
Event queue (KS1)
KS1 executed at 76
Event queue (KS2)
KS2 executed at 80
Sensor KS1 activated at 84
Event queue (KS1 KS3)
KS3 executed at 86
Event queue (KS1)
KS1 executed at 88
Event queue (KS2)
KS2 executed at 92
Sensor KS1 activated at 96
Event queue (KS1 KS3)
KS3 executed at 98
Event queue (KS1)
KS1 executed at 100
Event queue (KS2)
KS2 executed at 104
Sensor KS1 activated at 108
Event queue (KS1 KS3)
KS3 executed at 110
Event queue (KS1)
KS1 executed at 112
Event queue (KS2)
KS2 executed at 116
Sensor KS1 activated at 120
Event queue (KS1 KS3)
KS3 executed at 122
Event queue (KS1)
KS1 executed at 124
Event queue (KS2)
KS2 executed at 128

Sensor KS1 activated at 132
Event queue (KS1 KS3)
KS3 executed at 134
Event queue (KS1)
KS1 executed at 136
Event queue (KS2)
KS2 executed at 140
Sensor KS1 activated at 144
Event queue (KS1 KS3)
KS3 executed at 146
Event queue (KS1)
KS1 executed at 148
Event queue (KS2)
KS2 executed at 152
Sensor KS1 activated at 156
Event queue (KS1 KS3)
KS3 executed at 158
Event queue (KS1)
KS1 executed at 160
Event queue (KS2)
KS2 executed at 164
Event queue (KS3)
KS3 executed at 170

B_4 Results:

Data Object: D1
Type: REAL
Value 13
Times Read Locked 14
Time Write Locked 13

Data Object: D2
Type: REAL
Value 13
Times Read Locked 14
Time Write Locked 13

Data Object: D3
Type: NIL
Value 5.33
Times Read Locked 14
Time Write Locked 14

Data Object: D4
Type: NIL
Value 169
Times Read Locked 14
Time Write Locked 14

Data Object: D5
Type: NIL
Value 1
Times Read Locked 14
Time Write Locked 14

Data Object: D6
Type: NIL
Value 5.33
Times Read Locked 14
Time Write Locked 14

Data Object: D8
Type: NIL
Value 169
Times Read Locked 14
Time Write Locked 14

Data Object: D9
Type: NIL
Value 169
Times Read Locked 0
Time Write Locked 14

The simulation results show that B_4 executed to completion. Sensor ks_1 executed and read all ot its input stream. When the input stream was empty B_4 stopped execution. The simulation results are the same as the results acheived by hand executing system B_4, and the COBS generated software matched the design specification.

D.5 Simulation Support Software

This section contains the COBS Simulation System support software. This file contains all the support function and data structure definitions. This file is automatically loaded by all simulation systems generated by COBS.

```
;;; -*- Syntax: Common-Lisp; Package: COMMON-LISP-USER; Base: 10; Mode:
LISP -*-
;;;*************************************************************
;;;* File Sim-Base.lisp   Last Update 2/8/92          *
;;;*                                                  *
;;;* Sim-base.lisp contains the support functions for the *
;;;* Blackboard System Simulation System.             *
;;;*************************************************************
;;;
;;;* Define Variables *
;;;**********************
;;;
```

```
(defvar *execution-queue* '())
(defvar *activation-queue* '())
(defvar *sensor-list* '())
(defvar *kp-list* '())
(defvar *simulation-clock* 0)
(defvar *ready-list* '())
(defvar *min* 1000000)
(defvar *head-execution-queue* 1000000)
(defvar *data-out* '())

(define-presentation-type output-file-name ()
  :abbreviation-for `string)

(define-presentation-type update-interval ()
  :abbreviation-for `integer)

(define-presentation-type execution-delay ()
  :abbreviation-for `integer)

;;;*********************************
;;;* Define Knowledge Source Flavor *
;;;*********************************

(defflavor base_ks ((name nil)
    (type '" " 'string)
    (init-function nil)
    (act-function nil)
    (exec-function nil)
    (execution-delay 0)
    (update-interval 0)
    (completion-time 0)
    (activation-time 0))
          ()
 (:conc-name get-)
 :initable-instance-variables
 :writable-instance-variables)

;;;*******************************
;;;* Define a Methods to Display *
;;;* a Knowledge Source          *
;;;*******************************

(defmethod (show base_ks)
          ()
 (format *data-out* " ~% Knowledge Source: ~a ~
                ~& Type: ~a ~
                ~& Verify Function ~a ~
                ~& Activation Function ~a ~
```

```
                ~& Execution Function ~a ~
                ~& Execution Delay: ~d ~
                ~& Update Interval: ~d ~%"
              name type init-function act-function exec-function
              execution-delay update-interval)
)

;;;;*****************************************
;;;* Define Blackboard Data Object Flavor *
;;;*****************************************

(defflavor blackboard_data_object ((name nil)
                                   (type '" " 'string)
                                   (value nil)
                                   (lock '" " 'string)
                                   (read-lock 0)
                                   (write-lock 0)
                                   (input_list '() 'list)
                                   (ic_list '() 'list)
                                   (output_list '() 'list))
          ()
 (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

;;;*************************
;;;* Define a Methods to   *
;;;* Display a Data Object *
;;;*************************

(defmethod (show blackboard_data_object)
          ()
 (format *data-out* " ~% Data Object: ~a ~
                ~& Type: ~a ~
                ~& Value ~a ~
                ~& Times Read Locked ~a ~
                ~& Time Write Locked ~a ~%"
              name type value read-lock write-lock)
 )

;;;*********************************
;;;* Define a Method to Initialize *
;;;* the Knowledge Sources         *
;;;*********************************

(defmethod (init-sim base_ks)
          ()
 (cond
  ((equal type "SENSOR") (init-sensor self))
```

```
   (t (init-kp self)))
  )

;;;*************************
;;;* Define a Method to    *
;;;* Initialize the Sensors *
;;;*************************

(defmethod (init-sensor base_ks)
           ()
 (cond
  ((= 0 update-interval) (setf update-interval (accept 'update-interval)))
  (t t))
 (cond
  ((= 0 execution-delay) (setf execution-delay (accept 'execution-delay)))
  (t t))
 (push self *sensor-list*)
 (show self)
 (cond
  ((and (eval init-function) (= activation-time *simulation-clock*))
   (format *data-out* "~% Sensor ~a activated at ~a" name *simulation-clock*)
   (eval act-function)
   (setf completion-time (+ activation-time execution-delay))
   (setf activation-time (+ activation-time update-interval))
   (push self *execution-queue*))
  (t t))
 )

;;;*********************************
;;;* Define a Method to Initialize *
;;;* the Knowledge Processors      *
;;;*********************************

(defmethod (init-kp base_ks)
           ()
 (cond
  ((= 0 execution-delay) (setf execution-delay (accept 'execution-delay)))
  (t t))
 (push self *kp-list*)
 (show self)
 (cond
  ((eval init-function) (eval act-function)
   (setf completion-time (+ *simulation-clock* execution-delay))
   (push self *execution-queue*))
  (t t))
 )

;;;************************************************
;;;* Define Function to Initialized the Blackboard *
```

```
;;;*************************************************

(defun run-simulation ()
 (setf *execution-queue* '()
       *activation-queue* '()
       *sensor-list* '()
       *kp-list* '()
       *simulation-clock* 0
       *ready-list* '())
 (setf *data-out* (open (fs:merge-pathnames (accept 'output-file-name))
                        :direction :output
                        :if-exists :new-version))

 (build-ks-list)
 (build-do-list)
 (initialize-blackboard)
 (mapcar 'init-sim k/s-list)
 (execute-ready)
 (main-loop)
 (mapcar 'show d/o-list)
 (close *data-out*)
 )

;;;*************************************************
;;;* Define Function to Wakeup a Blackboard Handler *
;;;*************************************************

(defun wake-them-up (*in*)
 (cond
  ((equal (get-type (symbol-value *in*)) "SENSOR") nil)
  (t (cond
       ((eval (get-init-function (symbol-value *in*)))
        (eval (get-act-function (symbol-value *in*)))
        (setf (get-completion-time (symbol-value *in*))
              (+ *simulation-clock* (get-execution-delay (symbol-value *in*))))
        (push (symbol-value *in*) *execution-queue*))
       (t t))) )
 )

;;;*************************************************
;;;* Define Function to Initialized the Blackboard *
;;;*************************************************

(defun initialize-blackboard ()
 (start-them-up d/o-list)
 )

(defun start-them-up (*in*)
 (mapcar 'init-stuff *in*)
```

```
 )

;;;*******************************
;;;* Define Method to Initialize *
;;;* the Blackboard Data Objects *
;;;*******************************

(defmethod (init-stuff blackboard_data_object)
          ()
 (mapcar 'reset-input-conditionals ic_list)
 (cond
  ((or (NOT type) (equal value "undefined")))
  (t (mapcar 'set-input-conditionals ic_list))
  )
 )

;;;*****************************
;;;* Define Methods to Update *
;;;* Blackboard Data Objects  *
;;;*****************************

(defmethod (update blackboard_data_object)
          (*value-in*)
 (incf write-lock)
 (process:with-lock (lock :mode :write)
  (setf value *value-in*))
  (mapcar 'set-input-conditionals ic_list)
  (mapcar 'wake-them-up input_list)
 )

;;;****************************
;;;* Define Methods to Fetch *
;;;* Blackboard Data Objects *
;;;****************************

(defmethod (fetch blackboard_data_object)
          ()
 (incf read-lock)
 (process:with-lock (lock :mode :read)
  (mapcar 'reset-input-conditionals ic_list))
 value
  )

;;;*********************************************
;;;* Define Functions to Set and Reset      *
;;;* a Knowledge Source's Input Conditionals *
;;;*********************************************

(defun set-input-conditionals (*in*)
```

```
 (set *in* t)
 )

(defun reset-input-conditionals (*in*)
 (set *in* nil)
 )

;;;*****************************************************
;;;* Define Functions to Manage the Event queue, *
;;;* Find the first element in the queue, and       *
;;;* Execute the knowledge sources on the Ready List *
;;;*****************************************************

(defun main-loop ()
 (execute-ready)
 (eval-sensors)
 (cond
  ((NOT *execution-queue*) nil)
  (t (main-loop)) )
 )

(defun execute-ready ()
 (format *data-out* "~% Event queue ~a" (mapcar 'get-name *execution-queue*))
 (cond
  ((NOT *execution-queue*) t)
  ((= (length *execution-queue*) 1)
   (let ((temp (car *execution-queue*)))
    (process-ks (car *execution-queue*))
    (setf *execution-queue* (remove temp *execution-queue*))) )
  (t (process-queue)) )
 )

(defun process-queue ()
 (setf *ready-list* '())
 (setf *min* 1000000)
 (mapcar 'find-head-queue *execution-queue*)
 (mapcar 'get-list *execution-queue*)
 (mapcar 'process-ks *ready-list*)
 )

;;;*****************************************
;;;* Define a Method Execute the Knowledge *
;;;* Sources on the Ready-list             *
;;;*****************************************

 (defmethod (process-ks base_ks)
          ()
 (setf *simulation-clock* completion-time)
 (eval exec-function)
```

```
 (setf *execution-queue* (get-it-out self))
 (format *data-out* "~% ~a executed at ~a" name *simulation-clock*)
 )

;;;*****************************************
;;;* Define a Method Finds the Completion *
;;;* Time of the Knowledge Sources at     *
;;;* the Head of the Event queue      *
;;;*****************************************

(defmethod (find-head-queue base_ks)
          ()
;  (format *data-out* "~% name= ~a *min* = ~a completion-time = ~a"
;          name *min* completion-time)
 (cond
  ((> *min* completion-time) (setf *min* completion-time))
  (t t))
 )

;;;*****************************************
;;;* Define a Method Finds the Knowledge  *
;;;* Sources that should Execute Next and *
;;;* pushs them on the Ready List         *
;;;*****************************************

(defmethod (get-list base_ks)
          ()
 (cond
  ((= *min* completion-time)
   (setf *execution-queue* (get-it-out self))
   (push self *ready-list*))
  (t t))
 )

;;;*****************************************
;;;* Define a Method to Remove Knowledge *
;;;* Sources from the Event queue     *
;;;*****************************************

(defmethod (get-it-out base_ks)
          ()
 (setf *temp* '())
 (loop for x in *execution-queue*
       when (and (NOT (equal name (get-name x)))
                 (NOT (equal completion-time (get-completion-time x))))
        do (push x *temp*)
          )
 *temp*
)
```

```
;;;*************************************
;;;* Define Functions and Methods     *
;;;* to Evaluate and process Sensors *
;;;*************************************

(defun eval-sensors ()
; (print (mapcar 'get-name *execution-queue*))
 (cond
  ((NOT *sensor-list*) t)
  ((= (length *sensor-list*) 1) (process-sensor (car *sensor-list*)))
  (t (process-sensor-list)) )
 )

(defun process-sensor-list ()
 (setf *ready-list* '())
 (setf *min* 1000000)
 (setf *head-execution-queue* 1000000)
 (mapcar 'Find-head-queue *execution-queue*)
 (setf *head-execution-queue* *min*)
 (order-list *sensor-list*)
 (mapcar 'process-sensor *ready-list*)
 )

;;;*****************************************
;;;* Define a Method Finds the Knowledge  *
;;;* Sources that should Execute Next and *
;;;* pushs them on the Ready List         *
;;;*****************************************

(defmethod (order-list base_ks)
          (*in-list*)
 (cond
  ((NOT *in-list*) (setf *ready-list* (reverse *ready-list*)))
  ((= (length *in-list*) 1) (setf *ready-list* *in-list*))
  (t (setf *min* 1000000)
    (mapcar 'Find-head-queue *in-list*)
    (cond
        ((= *min* completion-time)
         (setf *in-list* (remove self *in-list*))
         (push self *ready-list*))
        (t t)) ))
 )

;;;*********************************
;;;* Define a Method to Process the *
;;;* Sensors on the Ready-list     *
;;;*********************************
```

```
(defmethod (process-sensor base_ks)
          ()
 (setf *min* 1000000)
 (setf *head-execution-queue* 1000000)
 (mapcar 'Find-head-queue *execution-queue*)
 (setf *head-execution-queue* *min*)

; (format *data-out* "~% ~a name *head-x-q* ~a Activation time ~a"
;          name *head-execution-queue* activation-time)

 (cond
  ((and (eval init-function) ( activation-time *head-execution-queue*))
   (eval act-function)
   (setf completion-time (+ activation-time execution-delay))
   (setf *simulation-clock* activation-time)
   (format *data-out* "~% Sensor ~a activated at ~a" name *simulation-clock*)
   (setf activation-time (+ activation-time update-interval))
   (push self *execution-queue*))
  (t t))
 )

;;;*****************************
;;;* Define a Function to View *
;;;* the Input Conditionals    *
;;;*****************************

(defun look ()
  (print (list "ks1-d2" ks1-d2))
  (print (list "ks1-d1" ks1-d1))
  (print (list "ks2-d3" ks2-d3))
  (print (list "ks3-d6" ks3-d6))
  (print (list "ks3-d4" ks3-d4))
  (print (list "ks4-d5" ks4-d5))
  (print (list "ks4-d7" ks4-d7))
  (print (list "ks4-d8" ks4-d8)) t )

;;;*********************************
;;;* Example Functions to Manage *
;;;* Sensor input queues         *
;;;*********************************

(defun ks1-update ()
 (cond
  ((> (length d1-list) 0) (update D1 (pop d1-list))
   (update D2 (pop d2-list)))
  (t (print "data all gone")) )
 )

(defun init-input ()
```

```
 (setf d1-list '(1 2 3 4 5 6 7 8 9 10 11 12 13))
 (setf d2-list '(1 2 3 4 5 6 7 8 9 10 11 12 13))
 )
```

D.6 COBS Simulation System Software

This section contains a listing of the LISP code for the COBS Simulation Tool.

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: simtools; Base: 10 -*-
;;;*****************************************************************
;;;
*
;;; Program: Concurrent Blackboard System
;;;        Simulation Code Generator
;;;
;;;*********************************
;;; define the program framework *
;;;*********************************
;;;

(DW:DEFINE-PROGRAM-FRAMEWORK simulation

;;;************************
;;; define the top level *
;;; program framework    *
;;;************************
;;;

 :top-level (my-top-level)

;;;************************************
;;; define the program selection key *
;;;************************************
;;;

  :SELECT-KEY
  #\s

;;;*********************************
;;; define the program framework *
;;;*********************************
;;;

  :COMMAND-DEFINER
  T

;;;************************************
;;; define the program command table *
;;;************************************
;;;

  :COMMAND-TABLE
  (:INHERIT-FROM
  '("accept-values-pane" "colon full command" "standard arguments"
```

```
   "input editor compatibility" "standard scrolling")
  :KBD-ACCELERATOR-P 'T)

;;;*************************************
;;; define the program state variables *
;;; and their initial values          *
;;;*************************************

  :STATE-VARIABLES
  ()

;;;*******************************
;;; define the program windows *
;;;*******************************

  :PANES
  ((PANE-1 :TITLE :REDISPLAY-STRING "Concurrent Object Oriented Blackboard
System"
         :HEIGHT-IN-LINES 1
         :REDISPLAY-AFTER-COMMANDS NIL)
   (PANE-4 :display
        :margin-components '((dw:margin-borders)
                    (dw:margin-label :string "Graphic Display Window"
                                :margin :top
                                :box :inside
                                :box-thickness 2
                                :style (:fix :bold :large)
                                :centered-p t)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-scroll-bar)) )
   (PANE-3 :listener
         :HEIGHT-IN-LINES 35
        :margin-components '((dw:margin-borders)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-label :string "Listener Window"
                                :margin :top
                                :box :inside
                                :box-thickness 2
                                :style (:fix :bold :large)
                                :centered-p t)
                    (dw:margin-scroll-bar)) )
   (PANE-6 :display
        :margin-components '((dw:margin-borders)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-label :string "List of Knowledge Sources"
                                :margin :top
                                :box :inside
                                :box-thickness 2
                                :centered-p t
```

```
                                :style (:fix :bold :large))
                        (dw:margin-scroll-bar)) )
   (PANE-5 :COMMAND-MENU
         :CENTER-P T
         :MENU-LEVEL
         :TOP-LEVEL
         :margin-components '((dw:margin-borders)
                        (dw:margin-white-borders :thickness 4)
                        (dw:margin-label :string "Commands"
                                :margin :top
                                :box :inside
                                :box-thickness 2
                                :style (:fix :bold :large)
                                :centered-p t)) )
   )


;;;****************************************
;;; define the window layout configuration *
;;;****************************************

  :CONFIGURATIONS
  '((DW::MAIN
     (:LAYOUT (DW::MAIN :COLUMN PANE-1 PANE-4 ROW-1)
      (ROW-1 :ROW PANE-3 PANE-6 PANE-5))
     (:SIZES (DW::MAIN (PANE-1 1 :LINES)
                  :THEN (PANE-4 :EVEN) (ROW-1 :EVEN))
      (ROW-1 (PANE-3 30 :LINES) (PANE-6 20 :LINES)
       (PANE-5 :ASK-WINDOW SELF
            :SIZE-FOR-PANE PANE-5) :THEN)))))

;;;*******************************
;;; define any needed variables *
;;;*******************************

  (defvar *data-out* nil)       ;;; Output Data File
  (defvar *data-in* nil)        ;;; Input Data File
  (defvar beta '())             ;;; List Of Knowledge Sources
  (defvar ks-list '())          ;;; List Of Knowledge Source Names
  (defvar data-object-list '()) ;;; List Of Blackboard Data Object Names
  (defvar d/o-list '())         ;;; List Of Blackboard Data Object(s)
  (defvar k/s-list '())         ;;;
  (defvar ks-in '())            ;;; Knowledge Source selected from menu
  (defvar do-in '())            ;;; Blackboard Data object selected from menu
  (defvar ks-name-in '())
  (defvar *defvar-list* '())    ;;; List of Input-conditionals to be DEFVAR[ed]
  (defvar *defvar-lock-list* '()) ;;; List of Locks to be DEFVAR[ed]
  (defvar *name* '())           ;;; Name of Data Object to be Updated
  (defvar ks-edit '())
```

```
  (defvar do-edit '())
  (defvar data-in1 '())
  (defvar ks_count 0)
  (defvar sensor_count 0)

  (defvar connected nil)
  (defvar data-in nil)
  (defvar *ks-spec* nil)
  (defvar *do-spec* nil)

  (defvar *pane1*)
  (defvar *pane3*)
  (defvar *pane4*)
  (defvar *pane5*)
  (defvar *pane6*)

;;;***********************************************
;;; define the variables required to build a  *
;;; pop up window to select knowledge sources *
;;;***********************************************

(defvar geometry-list (list 2))

(defvar *ks-menu* (tv:make-window 'tv:pop-up-menu
                        ':label '(:string "Select a Knowledge Source"
                                :character-style (:swiss :italic :normal))
;;                      ':geometry geometry-list
                        ':borders 4))

(defvar *do-menu* (tv:make-window 'tv:pop-up-menu
                        ':label '(:string "Select a Blackboard Data Object"
                                :character-style (:swiss :italic :normal))
                        ':borders 4))

;;;*****************************************
;;; define any needed presentation types. *
;;;*****************************************

(define-presentation-type ks-type ()
  :abbreviation-for `string)

(define-presentation-type output-file-name ()
  :abbreviation-for `string)

(define-presentation-type input-file-name ()
  :abbreviation-for `string)

(define-presentation-type input-conditional-list ()
  :abbreviation-for `list)
```

```
(define-presentation-type precondition-list ()
  :abbreviation-for `list)

(define-presentation-type postcondition-list ()
  :abbreviation-for `list)

(define-presentation-type input-variable-list ()
  :abbreviation-for `list)

(define-presentation-type output-variable-list ()
  :abbreviation-for `list)

(define-presentation-type rotation-about-y-axis-in-degrees ((limit))
  :abbreviation-for `((integer 0,limit)))

(define-presentation-type ks-name (())
  :no-deftype t
  :printer ((ks-name stream)
            (format stream " ~% knowledge source ~a" ks-name))
  :parser ((stream)
        (accept 'string :stream stream
                    :prompt "enter a Knowledge Source name")) )

;;;*************************************************
;;; define area for instances for specialized GC. *
;;;*************************************************

(defvar *instance-area*
      (make-area :name '*instance-area*
            :gc ':dynamic) )

;;;***************************************
;;;define Flavor for a Knowledge Source *
;;;***************************************

(defflavor base_ks ((name " " 'ks-name)
              (type " " 'ks-type)
              (input-conditionals '() 'input-conditional-list)
              (preconditions '() 'precondition-list)
              (postconditions '() 'postcondition-list)
              (psi '() 'input-variable-list)
              (phi '() 'output-variable-ist)
              (execution_time 0.0 'number)
              (depth 0 'number)
              (successor_list '() 'list)
              (predecessor_list '() 'list)
              (update_rate 0.0 'number)
              (psi_card 0 'integer)
```

```
            (phi_card 0 'integer))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type base_ks (())
  :no-deftype t
  :printer ((base_ks stream)
            (format stream " ~% knowledge source: ~a ~
                 ~& knowledge source type: ~a ~
                 ~& input variables: ~a ~
                 ~& output variables: ~a ~
                 ~& input conditionals: ~d ~
                 ~& preconditions: ~d ~
                 ~& postconditions: ~d ~
                 ~& depth: ~d ~
                 ~& predecessors: ~d ~
                 ~& successors: ~d ~
                 ~& knowledge source execution time: ~d ~
                 ~& cardinality of input variable set: ~d ~
                 ~& cardinality of output variable set: ~d ~%"
                 (get-name base_ks) (get-type base_ks)
                 (get-psi base_ks) (get-phi base_ks)
                 (get-input-conditionals base_ks)
                 (get-preconditions base_ks) (get-postconditions base_ks)
                 (get-depth base_ks) (get-predecessor_list base_ks)
                 (get-successor_list base_ks)
                 (get-execution_time base_ks)
                 (get-psi_card base_ks) (get-phi_card base_ks)) ) )

;;;*****************************************
;;;
;;;define methods for a knowledge source *
;;;*****************************************
;;;

(defmethod (show base_ks)
  ()
(format *data-out* " ~% knowledge source: ~a ~
                 ~& knowledge source type: ~a ~
                 ~& input variables: ~a ~
                 ~& output variables: ~a ~
                 ~& input conditionals: ~d ~
                 ~& preconditions: ~d ~
                 ~& postconditions: ~d ~
                 ~& depth: ~d ~
                 ~& predecessors: ~d ~
                 ~& successors: ~d ~
                 ~& knowledge source execution time: ~d ~
                 ~& cardinality of input variable set: ~d ~
```

```
            ~& cardinality of output variable set: ~d ~%"
        name type psi phi input-conditionals preconditions postconditions
        depth predecessor_list successor_list execution_time
        psi_card  phi_card)
)

(defmethod (display base_ks)
 ()
 (format *data-out* " ~% ~a ~
            ~& ~a ~
            ~& ~a ~
            ~& ~a ~
            ~& ~d ~
            ~& ~d ~
            ~& ~d ~
            ~& ~d ~
            ~& ~d ~
            ~& ~d ~
            ~& ~d ~
            ~& ~d ~
            ~& ~d ~%"
      name type psi phi input-conditionals preconditions postconditions
      depth predecessor_list successor_list execution_time
      psi_card  phi_card) )

(defmethod (build_process base_ks)
 ()
 (format *data-out* "~%(defun ~a ()" (String-append name "-activation"))
 (setq *name* name)
 (mapcar 'gen-fetch psi)
 (mapcar 'gen-resets input-conditionals)
 (cond
  ((string-equal type "SENSOR") (format *data-out* "~%~t (~a)"
                                  (String-append name "-update")))
  (t t) )
 (format *data-out* "~% ~T)~%")

 (format *data-out* "~%(defun ~a ()~
            ~& ~t (and"
            (String-append name "-verify"))
 (mapcar 'dump-inits (append input-conditionals preconditions))
 (format *data-out* ")) ~%" )

 (format *data-out* "~%(defun ~a ()" (string-append name "-func"))
 (format *data-out* "~& ~t (print ~\\quoted-string\\) ~%"
       "call to knowledge source handler")
 (mapcar 'gen-update phi)
 (format *data-out* ") ~%" )
```

```
  )

(defun dump-inits (*in*)
 (format *data-out* " ~a" *in*)
 )

(defun gen-fetch (*in*)
  (format *data-out* "~&~t (setf ~a~a-temp (fetch ~a))~%" *name* *in* *in*)
  )

(defun gen-resets (*in*)
  (format *data-out* "~&~t (setf ~a '())~%" *in*)
  )

(defun gen-update (*in*)
  (format *data-out* "~& ~t (update ~a ~a~a-temp)~%" *in* *name* *in*)
  )

;;;*****************************************
;;;Define flavor for a Blackboard Data Object *
;;;*****************************************

(defflavor blackboard_data_object ((name nil)
                                   (type '" " 'string)
                                   (value nil)
                                   (lock '" " 'string)
                                   (read-lock 0 'integer)
                                   (write-lock 0 'integer)
                                   (input_list '() 'list)
                                   (ic_list '() 'list)
                                   (output_list '() 'list))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type blackboard_data_object (())
  :no-deftype t
  :printer ((blackboard_data_object stream)
            (format stream "~& Blackboard Data Object Name: ~a ~
                        ~& Data Object Type: ~a ~
                        ~& Data Object Value: ~a ~
                        ~& Data Object Lock: ~a ~
                        ~& Data Object Read Lock Count: ~a ~
                        ~& Data Object Write Lock Count: ~a ~
                        ~& Input Conditionals: ~a ~
                        ~& Used as an Input by: ~a ~
                        ~& Output by: ~a ~%"
                    (get-name blackboard_data_object)
```

```
            (get-type blackboard_data_object)
            (get-value blackboard_data_object)
            (get-lock blackboard_data_object)
            (get-read-lock blackboard_data_object)
            (get-write-lock blackboard_data_object)
            (get-ic_list blackboard_data_object)
            (get-input_list blackboard_data_object)
            (get-output_list blackboard_data_object)) ) )

(defmethod (display blackboard_data_object)
 ()
  (format *data-out* " ~% ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~%"
      name type value lock read-lock write-lock input_list ic_list output_list) )

(defmethod (build_lock blackboard_data_object)
 ()
  (format *data-out* "~%(Setf ~a (process:make-lock ~\\quoted-string\\ :type ~
            :multiple-reader-single-writer :recursive t))"
      lock lock)
  )

(defmethod (build_data_instance blackboard_data_object)
 ()
  (format *data-out* "~%(setf ~a (make-instance 'blackboard_data_object ~
                        ~% ~t :name ~\\quoted-string\\ ~
                        ~% ~t :type ~\\quoted-string\\ ~
                        ~% ~t :value ~a ~
                        ~% ~t :lock ~a ~
                        ~% ~t :read-lock ~a ~
                        ~% ~t :write-lock ~a ~
                        ~% ~t :input_list '~a ~
                        ~% ~t :ic_list '~a ~
                        ~% ~t :output_list '~a)) ~%"
        name name type value lock read-lock write-lock input_list ic_list output_list)
  (format *data-out* "~%(push ~a d/o-list) ~%" name)
  )

(defmethod (build_ks_instance base_ks)
 ()
  (format *data-out* "~%(setf ~a (make-instance 'base_ks ~
                        ~% ~t :name ~\\quoted-string\\ ~
```

```
                                ~% ~t :type ~\\quoted-string\\ ~
                                ~% ~t :init-function '(~a) ~
                                ~% ~t :act-function '(~a) ~
                                ~% ~t :exec-function '(~a) ~
                                ~% ~t :execution-delay ~a ~
                                ~% ~t :update-interval ~a ~
                                ~% ~t :completion-time ~a ~
                                ~% ~t :activation-time ~a )) ~%"
          name name type (string-append name "-verify")(string-append name "-
activation")
          (string-append name "-func") execution_time update_rate 0 0)
  (format *data-out* "~%(push ~a k/s-list) ~%" name)
  )

(defun build-ks-flavor ()
  (format *data-out* "~%(defflavor base_ks ((name nil) ~
                ~& ~T (type '~\\quoted-string\\ 'string) ~
                ~& ~T (function nil) ~
                ~& ~T (execution-delay 0) ~
                ~& ~T (update-interval 0) ~
                ~& ~T (completion-time 0) ~
                ~& ~T (activation-time 0)) ~
  ~& ~T() ~
  ~& ~T(:conc-name get-) ~
  ~& ~T :initable-instance-variables ~
  ~& ~T :writable-instance-variables) " "" " "" ")
  )

;;;*************************************************************
;;;define functions to build component names from KS names. *
;;;*************************************************************

(defun display-ks-list ()
  (send *pane6* :clear-window)
  (present "Current List of Knowledge Sources" 'string :stream *pane6*)
  (loop for x in ks-list
       do (present x 'ks-name :stream *pane6*)) )

;;;*****************************
;;; define blackboard commands *
;;;*****************************
;;; View a KS Command *
;;;*********************

(define-simulation-command (display_ks :menu-accelerator
                                       "View a Knowledge Source"
                                       :keyboard-accelerator #\v)
                    ()
  (setf ks-in nil)
```

```
  (select-ks-name)

  (send *pane4* :clear-window)
  (loop for x in beta
       when (equal ks-in (get-name x))
        do (present x 'base_ks :stream *pane4*)
          )
  )

;;;*********************************
;;; View a BB data object Command *
;;;*********************************

(define-simulation-command (display_do :menu-accelerator
                                       "View a Blackboard Data Object"
                                       :keyboard-accelerator #\d)
                    ()
  (setf data-in nil)
  (select-data-object-name)

  (send *pane4* :clear-window)
  (loop for x in d/o-list
       when (equal data-in1 (get-name x))
        do (present x 'blackboard_data_object :stream *pane4*))
  )

;;;***************************
;;; Load a BB Specification *
;;;***************************

(define-simulation-command (load_spec :menu-accelerator
                                      "Load a Blackboard Specification"
                                      :keyboard-accelerator #\l)
                    ()
;;; open the specification file

  (setf *data-in* (open (fs:merge-pathnames (accept 'input-file-name))
                    :direction :input))

;;; Read in the Number of Knowledge Sources in the Specification file

  (init_system)
  (setf *ks-spec* (read *data-in*))

  (dotimes (i *ks-spec*)
    (pushnew (make-instance  'base_ks
                      :name (read *data-in*)
                      :type (read *data-in*)
                      :psi (read *data-in*)
```

```
                    :phi (read *data-in*)
                    :input-conditionals (read *data-in*)
                    :preconditions (read *data-in*)
                    :postconditions (read *data-in*)
                    :depth (read *data-in*)
                    :predecessor_list (read *data-in*)
                    :successor_list (read *data-in*)
                    :execution_time (read *data-in*)
                    :psi_card (read *data-in*)
                    :phi_card (read *data-in*))
      beta)

   (pushnew (get-name (car beta)) ks-list)
   )

  (setf *do-spec* (read *data-in*))

  (dotimes (i *do-spec*)
   (pushnew (make-instance  'blackboard_data_object
                            :area      *instance-area*
                            :name       (read *data-in*)
                            :type      (read *data-in*)
                            :value      (read *data-in*)
                            :lock      (read *data-in*)
                            :read-lock  0
                            :write-lock  0
                            :input_list  (read *data-in*)
                            :ic_list    (read *data-in*)
                            :output_list (read *data-in*))
          d/o-list)
   (pushnew (get-name (car d/o-list)) data-object-list)
)


;;; close the specification file

   (setf ks_count *ks-spec*)
   (close *data-in*)
   (send *pane6* :clear-window)
   (display-ks-list)
   )

;;;***************
;;; Quit Command *
;;;***************

(define-simulation-command (quit_bb :menu-accelerator
                                    "Quit"
                                    :keyboard-accelerator #\q)
```

```
                ()
  (send *pane1* :clear-window)
  (send *pane3* :clear-window)
  (send *pane4* :clear-window)
  (send *pane5* :clear-window)
  (send *pane6* :clear-window)
  (send dw:*program-frame* :clear-window)
  (send *pane1* :bury)
  (send *pane3* :bury)
  (send *pane4* :bury)
  (send *pane5* :bury)
  (send *pane6* :bury)
  (send dw:*program-frame* :bury)
  (process-abort *current-process* :all t))

;;;*************************************
;;; Clear the Blackboard Specification *
;;;*************************************

(define-simulation-command (clear_sim :menu-accelerator
                                      "Clear the Blackboard Specification"
                                      :keyboard-accelerator #\c)
                           ()
  (send *pane3* :clear-window)
  (send *pane4* :clear-window)
  (send *pane6* :clear-window)
  (init_system)
  )

;;;*****************************
;;; Generate Blackboard System *
;;;*****************************

(define-simulation-command (generate_bb_code :menu-accelerator
                                             "Generate BlackBoard System Simulation
Code"
                                             :keyboard-accelerator #\G)
  ()
 (setf *data-out* (open (fs:merge-pathnames (accept 'output-file-name))
                :direction :output
                :if-exists :new-version))

 (format *data-out*
        ";;; -*- Syntax: Common-Lisp; Package: COMMON-LISP-USER; Base: 10;
Mode: LISP -*- ~%")
 (format *data-out* "~%;;;************************")
 (format *data-out* "~%;;;* Load the Simulation *")
 (format *data-out* "~%;;;* Support Functions  *")
 (format *data-out* "~%;;;************************ ~%")
```

```
 (format *data-out* "~%(si:load ~\\quoted-string\\) ~%" "m:>john>sim-base")
 (format *data-out* "~%;;;*********************")
 (format *data-out* "~%;;;* Define Variables *")
 (format *data-out* "~%;;;********************* ~%")
 (format *data-out* "~%(defvar d/o-list '()) ~%")
 (format *data-out* "~%(defvar k/s-list '()) ~%")
 (format *data-out* "~%(defvar ks-list  '~a) ~%" ks-list)
 (format *data-out* "~%(defvar undefined '~\\quoted-string\\) ~%" "undefined")
 (build-variable-defvars)
 (format *data-out* "~%;;;****************************************")
 (format *data-out* "~%;;;* Define Blackboard Data Object Locks *")
 (format *data-out* "~%;;;**************************************** ~%")
 (mapcar 'build_lock (reverse d/o-list))
 (format *data-out* "~%~%;;;***********************************")
 (format *data-out* "~%;;;* Build Knowledge Source Objects *")
 (format *data-out* "~%;;;*********************************** ~%")
 (format *data-out* "~%(defun build-ks-list () ~%")
 (format *data-out* "~%(setf k/s-list '()) ~%")
 (mapcar 'build_ks_instance (reverse beta))
 (format *data-out* "~% ~t ) ~%")
 (format *data-out* "~%;;;****************************************")
 (format *data-out* "~%;;;* Define Knowledge Source Processess *")
 (format *data-out* "~%;;;**************************************** ~%")
 (mapcar 'build_process  beta)
 (format *data-out* "~%;;;***********************************")
 (format *data-out* "~%;;;* Build Blackboard Data Objects *")
 (format *data-out* "~%;;;*********************************** ~%")
 (format *data-out* "~%(defun build-do-list () ~%")
 (format *data-out* "~%(setf d/o-list '()) ~%")
 (mapcar 'build_data_instance (reverse d/o-list))
 (format *data-out* "~% ~t ) ~%")
 (close *data-out*)
 (format t "~%"))

;;;**********************************
;;; Functions Used to Generate Code *
;;;**********************************

(defun build-variable-defvars ()
 (setf *defvar-list* '())
 (get-defvars d/o-list)
 (mapcar 'defvar-form (reverse *defvar-list*))
 )

(defun get-defvars (*in-list*)
  (cond
    ((NOT *in-list*) t)
    (t (make-defvar-list (append (list (get-lock (car *in-list*)))
                                 (list (get-name (car *in-list*)))
```

```
                                    (get-ic_list (car *in-list*))) )
        (get-defvars (cdr *in-list*))) )
  )

(defun make-defvar-list (*in-ic*)
  (cond
   ((NOT *in-ic*) t)
   (t (pushnew (car *in-ic*) *defvar-list*)
        (make-defvar-list (cdr *in-ic*))) )
  )

(defun defvar-form (*ic-in*)
 (format *data-out* "(defvar ~a '()) ~%" *ic-in*)
 )

;;;*********************
;;; initialize system *
;;;*********************

(defun my-top-level (program)
  (init_system)
  (pop-panes)
  (dw:default-command-top-level program))

(defun init_system ()
 (setf beta '()
        data-object-list '()
        d/o-list '()
        ks-list '()
        ks_count 0
        sensor_count 0
        connected nil)
 )

(defun pop-panes ()
 (setf *pane1* (dw:get-program-pane 'pane-1)
        *pane3* (dw:get-program-pane 'pane-3)
        *pane4* (dw:get-program-pane 'pane-4)
        *pane5* (dw:get-program-pane 'pane-5)
        *pane6* (dw:get-program-pane 'pane-6))
 )

;;;*********************************************
;;; define a function to pop a menu to allow *
;;; the user to select a knowledge source   *
;;;*********************************************

(defun select-ks-name ()
  (send *ks-menu* ':set-item-list ks-list)
```

```
  (send *ks-menu* ':expose-near '(:mouse))
  (setq ks-in (send *ks-menu* ':choose))
  (send *ks-menu* ':deactivate) t)

(defun select-data-object-name ()
  (send *do-menu* ':set-item-list data-object-list)
  (send *do-menu* ':expose-near '(:mouse))
  (setq data-in1 (send *do-menu* ':choose))
  (send *do-menu* ':deactivate) t)
```

Appendix E The COBS Code Generator

This appendix contaion a listing of the COBS Code Generator and a listing of the code generated for example system B_2

E.1 Listing of the COBS Code Generator

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: buildtools; Base: 10 -*-
;;;*****************************************************************************
;;;
*
;;; Program: Concurrent Blackboard System
;;;        Code Generator
;;;
;;;*********************************
;;;
;;; define the program framework *
;;;*********************************
;;;

(DW:DEFINE-PROGRAM-FRAMEWORK build-code

;;;************************
;;;
;;; define the top level *
;;; program framework    *
;;;************************
;;;

 :top-level (my-top-level)

;;;************************************
;;;
;;; define the program selection key *
;;;************************************
;;;

 :SELECT-KEY
 #\b

;;;*********************************
;;;
;;; define the program framework *
;;;*********************************
;;;

 :COMMAND-DEFINER
 T

;;;************************************
;;;
;;; define the program command table *
;;;************************************
;;;

 :COMMAND-TABLE
 (:INHERIT-FROM
 '("accept-values-pane" "colon full command" "standard arguments"
```

```
   "input editor compatibility" "standard scrolling")
  :KBD-ACCELERATOR-P 'T)

;;;*************************************
;;; define the program state variables *
;;; and their initial values          *
;;;*************************************

 :STATE-VARIABLES
 ()

;;;*****************************
;;; define the program windows *
;;;*****************************

 :PANES
 ((PANE-1 :TITLE :REDISPLAY-STRING "Concurrent Object Oriented Blackboard
System"
        :HEIGHT-IN-LINES 1
        :REDISPLAY-AFTER-COMMANDS NIL)
  (PANE-4 :display
       :margin-components '((dw:margin-borders)
                    (dw:margin-label :string "Graphic Display Window"
                                :margin :top
                                :box :inside
                                :box-thickness 2
                                :style (:fix :bold :large)
                                :centered-p t)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-scroll-bar)) )
  (PANE-3 :listener
        :HEIGHT-IN-LINES 35
       :margin-components '((dw:margin-borders)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-label :string "Listener Window"
                                :margin :top
                                :box :inside
                                :box-thickness 2
                                :style (:fix :bold :large)
                                :centered-p t)
                    (dw:margin-scroll-bar)) )
  (PANE-6 :display
       :margin-components '((dw:margin-borders)
                    (dw:margin-white-borders :thickness 4)
                    (dw:margin-label :string "List of Knowledge Sources"
                                :margin :top
                                :box :inside
                                :box-thickness 2
                                :centered-p t
```

```
                                :style (:fix :bold :large))
                        (dw:margin-scroll-bar)) )
  (PANE-5 :COMMAND-MENU
        :CENTER-P T
        :MENU-LEVEL
        :TOP-LEVEL
        :margin-components '((dw:margin-borders)
                        (dw:margin-white-borders :thickness 4)
                        (dw:margin-label :string "Commands"
                                :margin :top
                                :box :inside
                                :box-thickness 2
                                :style (:fix :bold :large)
                                :centered-p t)) )
  )


;;;*****************************************
;;; define the window layout configuration *
;;;*****************************************

  :CONFIGURATIONS
  '((DW::MAIN
    (:LAYOUT (DW::MAIN :COLUMN PANE-1 PANE-4 ROW-1)
     (ROW-1 :ROW PANE-3 PANE-6 PANE-5))
    (:SIZES (DW::MAIN (PANE-1 1 :LINES)
                :THEN (PANE-4 :EVEN) (ROW-1 :EVEN))
     (ROW-1 (PANE-3 30 :LINES) (PANE-6 20 :LINES)
      (PANE-5 :ASK-WINDOW SELF
           :SIZE-FOR-PANE PANE-5) :THEN)))))

;;;*******************************
;;; define any needed variables *
;;;*******************************

  (defvar *data-out* nil)       ;;; Output Data File
  (defvar *data-in* nil)        ;;; Input Data File
  (defvar beta '())             ;;; List Of Knowledge Sources
  (defvar ks-list '())          ;;; List Of Knowledge Source Names
  (defvar data-object-list '()) ;;; List Of Blackboard Data Object Names
  (defvar d/o-list '())         ;;; List Of Blackboard Data Object(s)
  (defvar ks-in '())            ;;; Knowledge Source selected from menu
  (defvar do-in '())            ;;; Blackboard Data object selected from menu
  (defvar ks-name-in '())
  (defvar *defvar-list* '())    ;;; List of Input-conditionals to be DEFVAR[ed]
  (defvar *defvar-lock-list* '()) ;;; List of Locks to be DEFVAR[ed]
  (defvar *name* '())           ;;; Name of Data Object to be Updated
  (defvar ks-edit '())
  (defvar do-edit '())
```

```
  (defvar data-in1 '())
  (defvar ks_count 0)
  (defvar sensor_count 0)

  (defvar connected nil)
  (defvar data-in nil)
  (defvar *ks-spec* nil)
  (defvar *do-spec* nil)

  (defvar *pane1*)
  (defvar *pane3*)
  (defvar *pane4*)
  (defvar *pane5*)
  (defvar *pane6*)

;;;*********************************************
;;; define the variables required to build a  *
;;; pop up window to select knowledge sources *
;;;*********************************************

(defvar geometry-list (list 2))

(defvar *ks-menu* (tv:make-window 'tv:pop-up-menu
                        ':label '(:string "Select a Knowledge Source"
                                :character-style (:swiss :italic :normal))
;;                      ':geometry geometry-list
                        ':borders 4))

(defvar *do-menu* (tv:make-window 'tv:pop-up-menu
                        ':label '(:string "Select a Blackboard Data Object"
                                :character-style (:swiss :italic :normal))
                        ':borders 4))

;;;*****************************************
;;; define any needed presentation types. *
;;;*****************************************

(define-presentation-type ks-type ()
  :abbreviation-for `string)

(define-presentation-type output-file-name ()
  :abbreviation-for `string)

(define-presentation-type input-file-name ()
  :abbreviation-for `string)

(define-presentation-type input-conditional-list ()
  :abbreviation-for `list)
```

```
(define-presentation-type precondition-list ()
  :abbreviation-for `list)

(define-presentation-type postcondition-list ()
  :abbreviation-for `list)

(define-presentation-type input-variable-list ()
  :abbreviation-for `list)

(define-presentation-type output-variable-list ()
  :abbreviation-for `list)

(define-presentation-type rotation-about-y-axis-in-degrees ((limit))
  :abbreviation-for `((integer 0,limit)))

(define-presentation-type ks-name (())
  :no-deftype t
  :printer ((ks-name stream)
            (format stream " ~% knowledge source ~a" ks-name))
  :parser ((stream)
        (accept 'string :stream stream
                    :prompt "enter a Knowledge Source name")) )

;;;*************************************************
;;; define area for instances for specialized GC. *
;;;*************************************************

(defvar *instance-area*
      (make-area :name '*instance-area*
              :gc ':dynamic) )

;;;***************************************
;;;define Flavor for a Knowledge Source *
;;;***************************************

(defflavor base_ks ((name " " 'ks-name)
                (type " " 'ks-type)
                (input-conditionals '() 'input-conditional-list)
                (preconditions '() 'precondition-list)
                (postconditions '() 'postcondition-list)
                (psi '() 'input-variable-list)
                (phi '() 'output-variable-ist)
                (execution_time 0.0 'number)
                (depth 0 'number)
                (successor_list '() 'list)
                (predecessor_list '() 'list)
                (update_rate 0.0 'number)
                (psi_card 0 'integer)
                (phi_card 0 'integer))
```

```
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type base_ks (())
  :no-deftype t
  :printer ((base_ks stream)
            (format stream " ~% knowledge source: ~a ~
                  ~& knowledge source type: ~a ~
                  ~& input variables: ~a ~
                  ~& output variables: ~a ~
                  ~& input conditionals: ~d ~
                  ~& preconditions: ~d ~
                  ~& postconditions: ~d ~
                  ~& depth: ~d ~
                  ~& predecessors: ~d ~
                  ~& successors: ~d ~
                  ~& knowledge source execution time: ~d ~
                  ~& cardinality of input variable set: ~d ~
                  ~& cardinality of output variable set: ~d ~%"
                  (get-name base_ks) (get-type base_ks)
                  (get-psi base_ks) (get-phi base_ks)
                  (get-input-conditionals base_ks)
                  (get-preconditions base_ks) (get-postconditions base_ks)
                  (get-depth base_ks) (get-predecessor_list base_ks)
                  (get-successor_list base_ks)
                  (get-execution_time base_ks)
                  (get-psi_card base_ks) (get-phi_card base_ks)) ) )

;;;*************************************
;;;define methods for a knowledge source *
;;;*************************************

(defmethod (show base_ks)
  ()
(format *data-out* " ~% knowledge source: ~a ~
                  ~& knowledge source type: ~a ~
                  ~& input variables: ~a ~
                  ~& output variables: ~a ~
                  ~& input conditionals: ~d ~
                  ~& preconditions: ~d ~
                  ~& postconditions: ~d ~
                  ~& depth: ~d ~
                  ~& predecessors: ~d ~
                  ~& successors: ~d ~
                  ~& knowledge source execution time: ~d ~
                  ~& cardinality of input variable set: ~d ~
                  ~& cardinality of output variable set: ~d ~%"
```

```
        name type psi phi input-conditionals preconditions postconditions
        depth predecessor_list successor_list execution_time
        psi_card  phi_card)
)

(defmethod (display base_ks)
 ()
 (format *data-out* " ~% ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~
                ~& ~d ~%"
       name type psi phi input-conditionals preconditions postconditions
       depth predecessor_list successor_list execution_time
       psi_card  phi_card) )

(defmethod (compute_cardinality base_ks)
 ()
  (setf psi_card (length psi)
      phi_card (length phi)) )

(defmethod (build_process base_ks)
 ()
  (format *data-out* "~%(defun ~a ()"
          (String-append name "-init"))

  (setq *name* name)
  (mapcar 'gen-fetch psi)
  (mapcar 'gen-resets input-conditionals)

  (format *data-out* "~& ~t (print ~\\quoted-string\\) ~%"
          "replace this with call to knowledge source handler")

  (mapcar 'gen-update phi)
  (format *data-out* ")~%")

  (format *data-out* "~%(defun ~a ()~
                ~& ~t (and"
                (String-append name "-verify"))
  (mapcar 'dump-inits (append input-conditionals preconditions))
  (format *data-out* ")) ~%" )
```

```
  (format *data-out* "~%(Setf ~a (process:make-process ~\\quoted-string\\ ~
                ~& ~T :initial-function '~a ~
                ~& ~T :verify-function '~a ~
                ~& ~T :top-level-whostate ~\\quoted-string\\ ~
                ~& ~T :simple-p t)) ~%"
        name name (string-append name "-init") (string-append name "-verify") name)
  )

(defun dump-inits (*in*)
  (format *data-out* " ~a" *in*)
  )

(defun gen-fetch (*in*)
  (format *data-out* "~&~t (setf ~a~a-temp (fetch ~a))~%" *name* *in* *in*)
  )

(defun gen-resets (*in*)
  (format *data-out* "~&~t (setf ~a '())~%" *in*)
  )

(defun gen-update (*in*)
  (format *data-out* "~& ~t (update ~a ~a~a-temp)~%" *in* *name* *in*)
  )

;;;*****************************************
;;;Define flavor for a Blackboard Data Object *
;;;*****************************************

(defflavor blackboard_data_object ((name nil)
                                   (type '" " 'string)
                                   (value nil)
                                   (lock '" " 'string)
                                   (input_list '() 'list)
                                   (ic_list '() 'list)
                                   (output_list '() 'list))
  ()
  (:conc-name get-)
  :initable-instance-variables
  :writable-instance-variables)

(define-presentation-type blackboard_data_object (())
  :no-deftype t
  :printer ((blackboard_data_object stream)
            (format stream "~& Blackboard Data Object Name: ~a ~
                        ~& Data Object Type: ~a ~
                        ~& Data Object Value: ~a ~
                        ~& Data Object Lock: ~a ~
                        ~& Input Conditionals: ~a ~
```

```
                    ~& Used as an Input by: ~a ~
                    ~& Output by: ~a ~%"
                (get-name blackboard_data_object)
                (get-type blackboard_data_object)
                (get-value blackboard_data_object)
                (get-lock blackboard_data_object)
                (get-ic_list blackboard_data_object)
                (get-input_list blackboard_data_object)
                (get-output_list blackboard_data_object)) ) )

(defmethod (display blackboard_data_object)
 ()
 (format *data-out* " ~% ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~
                ~& ~a ~%"
       name type value lock input_list ic_list output_list) )

(defmethod (build_lock blackboard_data_object)
 ()
 (format *data-out* "~%(Setf ~a (process:make-lock ~\\quoted-string\\ :type ~
              :multiple-reader-single-writer :recursive t))"
       lock lock)
 )

(defmethod (build_data_instance blackboard_data_object)
 ()
 (format *data-out* "~%(setf ~a (make-instance 'blackboard_data_object ~
                        ~% ~t :name ~\\quoted-string\\ ~
                        ~% ~t :type ~\\quoted-string\\ ~
                        ~% ~t :value ~a ~
                        ~% ~t :lock ~a ~
                        ~% ~t :input_list '~a ~
                        ~% ~t :ic_list '~a ~
                        ~% ~t :output_list '~a)) ~%"
         name name type value lock input_list ic_list output_list)
 (format *data-out* "~%(push ~a d/o-list) ~%" name)
 )

(defun dump-methods-update ()
 (format *data-out*
         "~%(defmethod (update blackboard_data_object) ~
~% ~t(*value-in*)~
~& ~t(process:with-lock (lock :mode :write)~
~& ~t  (setf value *value-in*)~
~& ~t  (mapcar 'set-input-conditionals ic_list)~
```

```
  ~& ~t  (mapcar 'wake-them-up input_list))~
  ~& ~t  )~%" )
   )

(defun dump-methods-fetch ()
   (format *data-out* "~%")
   (format *data-out*
           "~%(defmethod (fetch blackboard_data_object) ~
   ~% ~t ()~
   ~& ~t  (process:with-lock (lock :mode :read)~
   ~& ~t    (mapcar 'reset-input-conditionals ic_list))~
   ~& ~t value ~
   ~& ~t )~%" )
   )

(defun dump-set-input-cond ()
   (format *data-out*
           "~%(defun set-input-conditionals (*in*)~
   ~& ~t  (set *in* t)~
   ~& ~t  )~%" )

   (format *data-out*
           "~%(defun reset-input-conditionals (*in*)~
   ~& ~t  (set *in* nil)~
   ~& ~t  )~%" )
   )

(defun dump-wake-em-up ()
   (format *data-out*
           "~%(defun wake-them-up (*in*)~
   ~& ~t (process:wakeup (symbol-value *in*))~
   ~& ~t)~%" )
   )

(defun dump-start-em-up ()
   (format *data-out*
           "~%(defun start-them-up (*in*)~
   ~& ~t (mapcar 'init-them *in*))~
   ~& ~t)~%" )
   )

;;;*************************************************************
;;;define functions to build component names from KS names. *
;;;*************************************************************

(defun display-ks-list ()
   (send *pane6* :clear-window)
   (present "Current List of Knowledge Sources" 'string :stream *pane6*)
   (loop for x in ks-list
```

```
      do (present x 'ks-name :stream *pane6*)) )

(defun build-data-flavor ()
 (format *data-out* "~%(defflavor blackboard_data_object ((name nil) ~
              ~& ~34T (type '~\\quoted-string\\ 'string) ~
              ~& ~34T (value nil) ~
              ~& ~34T (lock '~\\quoted-string\\ 'string) ~
              ~& ~34T (input_list '() 'list) ~
              ~& ~34T (ic_list '() 'list) ~
              ~& ~34T (output_list '() 'list)) ~
  ~& ~T() ~
  ~& ~T(:conc-name get-) ~
  ~& ~T :initable-instance-variables ~
  ~& ~T :writable-instance-variables) " '" " '" ")
 )

(defun build-init-flavor ()
 (format *data-out* "~%(defmethod (init-them blackboard_data_object) ~
 ~& ~T () ~
 ~& ~T (cond ~
 ~& ~T   ((NOT type) t)~
 ~& ~T   (t (update self value))) ~
 ~& ~T )" )
 )

;;;*****************************
;;; define blackboard commands *
;;;*****************************
;;; View a KS Command *
;;;*********************

(define-build-code-command (display_ks :menu-accelerator
                                       "View a Knowledge Source"
                                       :keyboard-accelerator #\v)
                    ()
  (setf ks-in nil)
  (select-ks-name)

  (send *pane4* :clear-window)
  (loop for x in beta
      when (equal ks-in (get-name x))
       do (present x 'base_ks :stream *pane4*)
         )
  )

;;;*********************************
;;; View a BB data object Command *
;;;*********************************
```

```
(define-build-code-command (display_do :menu-accelerator
                                       "View a Blackboard Data Object"
                                       :keyboard-accelerator #\5)
                           ()
  (setf data-in nil)
  (select-data-object-name)

  (send *pane4* :clear-window)
  (loop for x in d/o-list
      when (equal data-in1 (get-name x))
       do (present x 'blackboard_data_object :stream *pane4*))
  )

;;;*****************************
;;; Load a BB Specification *
;;;*****************************

(define-build-code-command (load_spec :menu-accelerator
                                       "Load a Blackboard Specification"
                                       :keyboard-accelerator #\l)
                           ()
;;; open the specification file

  (setf *data-in* (open (fs:merge-pathnames (accept 'input-file-name))
                        :direction :input))

;;; Read in the Number of Knowledge Sources in the Specification file

  (init_system)
  (setf *ks-spec* (read *data-in*))

  (dotimes (i *ks-spec*)
    (pushnew (make-instance  'base_ks
                        :name (read *data-in*)
                        :type (read *data-in*)
                        :psi (read *data-in*)
                        :phi (read *data-in*)
                        :input-conditionals (read *data-in*)
                        :preconditions (read *data-in*)
                        :postconditions (read *data-in*)
                        :depth (read *data-in*)
                        :predecessor_list (read *data-in*)
                        :successor_list (read *data-in*)
                        :execution_time (read *data-in*)
                        :psi_card (read *data-in*)
                        :phi_card (read *data-in*))
      beta)

    (pushnew (get-name (car beta)) ks-list)
```

```
  )

 (setf *do-spec* (read *data-in*))

 (dotimes (i *do-spec*)
  (pushnew (make-instance  'blackboard_data_object
                           :area       *instance-area*
                           :name        (read *data-in*)
                           :type       (read *data-in*)
                           :value       (read *data-in*)
                           :lock       (read *data-in*)
                           :input_list  (read *data-in*)
                           :ic_list    (read *data-in*)
                           :output_list (read *data-in*))
           d/o-list)
  (pushnew (get-name (car d/o-list)) data-object-list)
)


;;; close the specification file

  (setf ks_count *ks-spec*)
  (close *data-in*)
  (send *pane6* :clear-window)
  (display-ks-list)
  )

;;;***************
;;; Quit Command *
;;;***************

(define-build-code-command (quit_bb :menu-accelerator
                                     "Quit"
                                     :keyboard-accelerator #\q)
                  ()
  (send *pane1* :clear-window)
  (send *pane3* :clear-window)
  (send *pane4* :clear-window)
  (send *pane5* :clear-window)
  (send *pane6* :clear-window)
  (send dw:*program-frame* :clear-window)
  (send *pane1* :bury)
  (send *pane3* :bury)
  (send *pane4* :bury)
  (send *pane5* :bury)
  (send *pane6* :bury)
  (send dw:*program-frame* :bury)
  (process-abort *current-process* :all t))
```

```
;;;*****************************
;;; Generate Blackboard System *
;;;*****************************

(define-build-code-command (generate_bb_code :menu-accelerator
                                              "Generate BlackBoard System Code"
                                              :keyboard-accelerator #\G)
  ()
 (setf *data-out* (open (fs:merge-pathnames (accept 'output-file-name))
                        :direction :output
                        :if-exists :new-version))

 (format *data-out*
         ";;; -*- Syntax: Common-Lisp; Package: COMMON-LISP-USER; Base: 10;
Mode: LISP -*- ~%")
 (format *data-out* "~%;;;*********************")
 (format *data-out* "~%;;;* Define Variables *")
 (format *data-out* "~%;;;********************* ~%")
 (format *data-out* "~%(defvar d/o-list '()) ~%")
 (build-variable-defvars)
 (format *data-out* "~%;;;****************************************")
 (format *data-out* "~%;;;* Define Blackboard Data Object Flavor *")
 (format *data-out* "~%;;;**************************************** ~%")
 (build-data-flavor)
 (format *data-out* "~%;;;****************************************")
 (format *data-out* "~%;;;* Define Blackboard Data Object Methods *")
 (format *data-out* "~%;;;**************************************** ~%")
 (build-init-flavor)
 (format *data-out* "~%")
 (format *data-out*
"~%;;;************************************************")
 (format *data-out* "~%;;;* Define Function to Wakeup a Blackboard Handler *")
 (format *data-out*
"~%;;;************************************************ ~%")
 (dump-wake-em-up)
 (format *data-out*
"~%;;;************************************************")
 (format *data-out* "~%;;;* Define Function to Initialized the Blackboard  *")
 (format *data-out*
"~%;;;************************************************ ~%")
 (dump-start-em-up)
 (format *data-out* "~%;;;*****************************")
 (format *data-out* "~%;;;* Define Methods to Update *")
 (format *data-out* "~%;;;* Blackboard Data Objects  *")
 (format *data-out* "~%;;;***************************** ~%")
 (dump-methods-update)
 (format *data-out* "~%;;;*****************************")
 (format *data-out* "~%;;;* Define Methods to Fetch *")
 (format *data-out* "~%;;;* Blackboard Data Objects *")
```

```
 (format *data-out* "~%;;;************************** ~%")
 (dump-methods-fetch)
 (format *data-out* "~%;;;*******************************************")
 (format *data-out* "~%;;;* Define Functions to Set and Reset      *")
 (format *data-out* "~%;;;* a Knowledge Source's Input Conditionals *")
 (format *data-out* "~%;;;*******************************************
~%")
 (dump-set-input-cond)
 (format *data-out* "~%;;;*****************************************")
 (format *data-out* "~%;;;* Define Blackboard Data Object Locks *")
 (format *data-out* "~%;;;***************************************** ~%")
 (mapcar 'build_lock (reverse d/o-list))
 (format *data-out* "~% ~%;;;*****************************************")
 (format *data-out* "~%;;;* Define Knowledge Source Processess *")
 (format *data-out* "~%;;;***************************************** ~%")
 (mapcar 'build_process  beta)
 (format *data-out* "~%;;;*********************************")
 (format *data-out* "~%;;;* Build Blackboard Data Objects *")
 (format *data-out* "~%;;;********************************* ~%")
 (mapcar 'build_data_instance (reverse d/o-list))
 (close *data-out*)
 (format t "~%"))

;;;*********************************
;;; Functions Used to Generate Code *
;;;*********************************

(defun build-variable-defvars ()
 (setf *defvar-list* '())
 (get-defvars d/o-list)
 (mapcar 'defvar-form (reverse *defvar-list*))
 )

(defun get-defvars (*in-list*)
  (cond
   ((NOT *in-list*) t)
   (t (make-defvar-list (append (list (get-lock (car *in-list*)))
                                (list (get-name (car *in-list*)))
                                (get-ic_list (car *in-list*))) )
        (get-defvars (cdr *in-list*))) )
  )

(defun make-defvar-list (*in-ic*)
  (cond
   ((NOT *in-ic*) t)
   (t (pushnew (car *in-ic*) *defvar-list*)
        (make-defvar-list (cdr *in-ic*))) )
  )
```

```
(defun defvar-form (*ic-in*)
 (format *data-out* "(defvar ~a '()) ~%" *ic-in*)
 )

;;;*********************
;;; initialize system *
;;;*********************

(defun my-top-level (program)
  (init_system)
  (pop-panes)
  (dw:default-command-top-level program))

(defun init_system ()
 (setf beta '()
       data-object-list '()
       d/o-list '()
       ks-list '()
       ks_count 0
       sensor_count 0
       connected nil)
 )

(defun pop-panes ()
 (setf *pane1* (dw:get-program-pane 'pane-1)
       *pane3* (dw:get-program-pane 'pane-3)
       *pane4* (dw:get-program-pane 'pane-4)
       *pane5* (dw:get-program-pane 'pane-5)
       *pane6* (dw:get-program-pane 'pane-6))
 )

;;;*******************************************
;;; define a function to pop a menu to allow *
;;; the user to select a knowledge source    *
;;;*******************************************

(defun select-ks-name ()
  (send *ks-menu* ':set-item-list ks-list)
  (send *ks-menu* ':expose-near '(:mouse))
  (setq ks-in (send *ks-menu* ':choose))
  (send *ks-menu* ':deactivate) t)

(defun select-data-object-name ()
  (send *do-menu* ':set-item-list data-object-list)
  (send *do-menu* ':expose-near '(:mouse))
  (setq data-in1 (send *do-menu* ':choose))
  (send *do-menu* ':deactivate) t)
```

E.2 COBS Generated B_2 Code

```
;;; -*- Syntax: Common-Lisp; Package: COMMON-LISP-USER; Base: 10; Mode:
LISP -*-

;;;**********************
;;;* Define Variables *
;;;**********************

(defvar d/o-list '())
(defvar D8-LOCK '())
(defvar D8 '())
(defvar KS4-D8 '())
(defvar D7-LOCK '())
(defvar D7 '())
(defvar KS4-D7 '())
(defvar D6-LOCK '())
(defvar D6 '())
(defvar KS3-D6 '())
(defvar D9-LOCK '())
(defvar D9 '())
(defvar KS2-D9 '())
(defvar D1-LOCK '())
(defvar D1 '())
(defvar KS1-D1 '())
(defvar D2-LOCK '())
(defvar D2 '())
(defvar KS1-D2 '())
(defvar D3-LOCK '())
(defvar D3 '())
(defvar KS2-D3 '())
(defvar D4-LOCK '())
(defvar D4 '())
(defvar KS3-D4 '())
(defvar D5-LOCK '())
(defvar D5 '())
(defvar KS4-D5 '())

;;;****************************************
;;;* Define Blackboard Data Object Flavor *
;;;****************************************

(defflavor blackboard_data_object ((name nil)
    (type '" " 'string)
    (value nil)
    (lock '" " 'string)
    (input_list '() 'list)
    (ic_list '() 'list)
    (output_list '() 'list))
   ()
   (:conc-name get-)
```

```
    :initable-instance-variables
    :writable-instance-variables)
;;;*****************************************
;;;* Define Blackboard Data Object Methods *
;;;*****************************************

(defmethod (init-them blackboard_data_object)
    ()
    (cond
     ((NOT type) t)
     (t (update self value)))
    )

;;;*****************************************************
;;;* Define Function to Wakeup a Blackboard Handler *
;;;*****************************************************

(defun wake-them-up (*in*)
    (process:wakeup (symbol-value *in*))
   )

;;;*****************************************************
;;;* Define Function to Initialized the Blackboard  *
;;;*****************************************************

(defun start-them-up (*in*)
    (mapcar 'init-them *in*)
   )

;;;*****************************
;;;* Define Methods to Update *
;;;* Blackboard Data Objects  *
;;;*****************************

(defmethod (update blackboard_data_object)
   (*value-in*)
   (process:with-lock (lock :mode :write)
    (setf value *value-in*)
    (mapcar 'set-input-conditionals ic_list)
    (mapcar 'wake-them-up input_list))
    )

;;;*****************************
;;;* Define Methods to Fetch *
;;;* Blackboard Data Objects *
;;;*****************************


(defmethod (fetch blackboard_data_object)
```

```
    ()
    (process:with-lock (lock :mode :read)
    value)
    )

;;;*****************************************
;;;* Define Functions to Set and Reset     *
;;;* a Knowledge Source's Input Conditionals *
;;;*****************************************

(defun set-input-conditionals (*in*)
    (set *in* t)
    )

(defun reset-input-conditionals (*in*)
    (set *in* nil)
    )

;;;***************************************
;;;* Define Blackboard Data Object Locks *
;;;***************************************

(Setf D5-LOCK (process:make-lock "D5-LOCK"
        :type :multiple-reader-single-writer :recursive t))
(Setf D4-LOCK (process:make-lock "D4-LOCK"
        :type :multiple-reader-single-writer :recursive t))
(Setf D3-LOCK (process:make-lock "D3-LOCK"
        :type :multiple-reader-single-writer :recursive t))
(Setf D2-LOCK (process:make-lock "D2-LOCK"
        :type :multiple-reader-single-writer :recursive t))
(Setf D1-LOCK (process:make-lock "D1-LOCK"
        :type :multiple-reader-single-writer :recursive t))
(Setf D9-LOCK (process:make-lock "D9-LOCK"
        :type :multiple-reader-single-writer :recursive t))
(Setf D6-LOCK (process:make-lock "D6-LOCK"
        :type :multiple-reader-single-writer :recursive t))
(Setf D7-LOCK (process:make-lock "D7-LOCK"
        :type :multiple-reader-single-writer :recursive t))
(Setf D8-LOCK (process:make-lock "D8-LOCK"
        :type :multiple-reader-single-writer :recursive t))

;;;*************************************
;;;* Define Knowledge Source Processess *
;;;*************************************

(defun KS4-init ()
   (setf KS4D5-temp (fetch D5))
   (setf KS4D7-temp (fetch D7))
   (setf KS4D8-temp (fetch D8))
```

```
   (setf KS4-D8 '())
   (setf KS4-D7 '())
   (setf KS4-D5 '())
   (print "replace this with call to knowledge source handler")
   (update D9 KS4D9-temp)
)

(defun KS4-verify ()
   (and KS4-D8 KS4-D7 KS4-D5 (NOT (= (GET-VALUE D9) 13.6))   (NOT (=
(GET-VALUE D2) 0.0))
        (NOT (= (GET-VALUE D8) 0.0))))

(Setf KS4 (process:make-process "KS4"
   :initial-function 'KS4-init
   :verify-function 'KS4-verify
   :top-level-whostate "KS4"
   :simple-p t))

(defun KS3-init ()
   (setf KS3D4-temp (fetch D4))
   (setf KS3D6-temp (fetch D6))
   (setf KS3-D6 '())
   (setf KS3-D4 '())
   (print "replace this with call to knowledge source handler")
   (update D8 KS3D8-temp)
)

(defun KS3-verify ()
   (and KS3-D6 KS3-D4 (NOT (= (GET-VALUE D8) PI))
        (NOT (= (GET-VALUE D9) 13.6))))

(Setf KS3 (process:make-process "KS3"
   :initial-function 'KS3-init
   :verify-function 'KS3-verify
   :top-level-whostate "KS3"
   :simple-p t))

(defun KS2-init ()
   (setf KS2D3-temp (fetch D3))
   (setf KS2-D9 '())
   (setf KS2-D3 '())
   (print "replace this with call to knowledge source handler")
   (update D6 KS2D6-temp)
   (update D7 KS2D7-temp)
)

(defun KS2-verify ()
   (and KS2-D9 KS2-D3 (NOT (= (GET-VALUE D4) 7))
        (NOT (= (GET-VALUE D8) PI))))
```

```
(Setf KS2 (process:make-process "KS2"
    :initial-function 'KS2-init
    :verify-function 'KS2-verify
    :top-level-whostate "KS2"
    :simple-p t))

(defun KS1-init ()
   (setf KS1D1-temp (fetch D1))
   (setf KS1D2-temp (fetch D2))
   (setf KS1-D2 '())
   (setf KS1-D1 '())
    (print "replace this with call to knowledge source handler")
    (update D3 KS1D3-temp)
    (update D4 KS1D4-temp)
    (update D5 KS1D5-temp)
)

(defun KS1-verify ()
    (and KS1-D2 KS1-D1))

(Setf KS1 (process:make-process "KS1"
    :initial-function 'KS1-init
    :verify-function 'KS1-verify
    :top-level-whostate "KS1"
    :simple-p t))

;;;**********************************
;;;* Build Blackboard Data Objects *
;;;**********************************

(setf D5 (make-instance 'blackboard_data_object
    :name "D5"
    :type "REAL"
    :value UNDEFINED
    :lock D5-LOCK
    :input_list '(KS4)
    :ic_list '(KS4-D5)
    :output_list '(KS1)))

(push D5 d/o-list)

(setf D4 (make-instance 'blackboard_data_object
    :name "D4"
    :type "REAL"
    :value UNDEFINED
    :lock D4-LOCK
    :input_list '(KS3)
    :ic_list '(KS3-D4)
```

```
    :output_list '(KS1)))

(push D4 d/o-list)

(setf D3 (make-instance 'blackboard_data_object
    :name "D3"
    :type "REAL"
    :value UNDEFINED
    :lock D3-LOCK
    :input_list '(KS2)
    :ic_list '(KS2-D3)
    :output_list '(KS1)))

(push D3 d/o-list)

(setf D2 (make-instance 'blackboard_data_object
    :name "D2"
    :type "REAL"
    :value 1.8
    :lock D2-LOCK
    :input_list '(KS1)
    :ic_list '(KS1-D2)
    :output_list 'NIL))

(push D2 d/o-list)

(setf D1 (make-instance 'blackboard_data_object
    :name "D1"
    :type "REAL"
    :value 2.2
    :lock D1-LOCK
    :input_list '(KS1)
    :ic_list '(KS1-D1)
    :output_list 'NIL))

(push D1 d/o-list)

(setf D9 (make-instance 'blackboard_data_object
    :name "D9"
    :type "REAL"
    :value 12.22
    :lock D9-LOCK
    :input_list 'NIL
    :ic_list '(KS2-D9)
    :output_list '(KS4)))

(push D9 d/o-list)

(setf D6 (make-instance 'blackboard_data_object
```

```
    :name "D6"
    :type "REAL"
    :value UNDEFINED
    :lock D6-LOCK
    :input_list '(KS3)
    :ic_list '(KS3-D6)
    :output_list '(KS2)))

(push D6 d/o-list)

(setf D7 (make-instance 'blackboard_data_object
    :name "D7"
    :type "REAL"
    :value UNDEFINED
    :lock D7-LOCK
    :input_list '(KS4)
    :ic_list '(KS4-D7)
    :output_list '(KS2)))

(push D7 d/o-list)

(setf D8 (make-instance 'blackboard_data_object
    :name "D8"
    :type "REAL"
    :value UNDEFINED
    :lock D8-LOCK
    :input_list '(KS4)
    :ic_list '(KS4-D8)
    :output_list '(KS3)))

(push D8 d/o-list)
```

Appendix F COBS Verification and Validation Results

This appendix contains the COBS blackboard system design analysis results for each of the four version of Cube_CLAWS and the two Paladin designs. The results of the COBS simulation runs for each system are included at the end of the appendix.

F.1 Cube Design Analysis Results:

Specialization Values:

Interdependence Values:

Π<ks5ks4> 0.75
Π<ks5ks1> 1.0
Π<ks5ks2> 0.75
Π<ks5ks9> 0.75
Π<ks4ks5> 1.0
Π<ks4ks1> 0.75
Π<ks4ks2> 0.75
Π<ks4ks9> 0.75
Π<ks1ks7> 1.0
Π<ks2ks4> 0.7692308
Π<ks2ks1> 0.7692308
Π<ks2ks3> 0.23076923
Π<ks2ks8> 0.23076923
Π<ks2ks9> 0.23076923
Π<ks3ks4> 1.0
Π<ks3ks1> 0.8333333
Π<ks6ks2> 1.0
Π<ks7ks4> 0.23076923
Π<ks7ks3> 0.23076923
Π<ks7ks6> 0.7692308
Π<ks7ks8> 0.23076923
Π<ks7ks9> 0.7692308
Π<ks8ks6> 0.8333333
Π<ks8ks9> 1.0
Π<ks9ks4> 0.75
Π<ks9ks6> 0.75
Π<ks9ks7> 0.75
Π<ks9ks10> 1.0
Π<ks10ks4> 0.75
Π<ks10ks6> 1.0
Π<ks10ks7> 0.75
Π<ks10ks9> 0.75

Serialization Values:

Σ<ks5ks4> 0.09677419
Σ<ks5ks1> 0.18181819
Σ<ks5ks2> 0.33333334
Σ<ks5ks9> 0.09677419
Σ<ks4ks5> 0.5
Σ<ks4ks1> 0.13636364
Σ<ks4ks2> 0.33333334
Σ<ks4ks9> 0.09677419
Σ<ks1ks7> 0.33333334
Σ<ks2ks4> 0.32258064
Σ<ks2ks1> 0.45454547
Σ<ks2ks3> 0.5
Σ<ks2ks8> 0.5
Σ<ks2ks9> 0.09677419
Σ<ks3ks4> 0.19354838
Σ<ks3ks1> 0.22727273
Σ<ks6ks2> 0.33333334
Σ<ks7ks4> 0.09677419
Σ<ks7ks3> 0.5
Σ<ks7ks6> 0.45454547
Σ<ks7ks8> 0.5
Σ<ks7ks9> 0.32258064
Σ<ks8ks6> 0.22727273
Σ<ks8ks9> 0.19354838
Σ<ks9ks4> 0.09677419
Σ<ks9ks6> 0.13636364
Σ<ks9ks7> 0.33333334
Σ<ks9ks10> 0.5
Σ<ks10ks4> 0.09677419
Σ<ks10ks6> 0.18181819
Σ<ks10ks7> 0.33333334
Σ<ks10ks9> 0.09677419

F.2 Cube2 Design Analysis Results:

Specialization Values:

Interdependence Values:

Π<ks5ks4> 0.85714287
Π<ks5ks1> 1.0
Π<ks5ks2> 0.85714287
Π<ks5ks9> 0.85714287
Π<ks5ks11> 0.85714287
Π<ks5ks12> 0.85714287
Π<ks4ks5> 1.0
Π<ks1ks7> 1.0
Π<ks2ks4> 0.7692308
Π<ks2ks1> 0.7692308
Π<ks2ks3> 0.23076923
Π<ks2ks8> 0.23076923
Π<ks2ks9> 0.23076923
Π<ks2ks11> 0.23076923
Π<ks2ks12> 0.7692308
Π<ks3ks4> 1.0
Π<ks3ks1> 0.8333333
Π<ks3ks12> 1.0
Π<ks6ks2> 1.0
Π<ks7ks4> 0.23076923
Π<ks7ks3> 0.23076923
Π<ks7ks6> 0.7692308
Π<ks7ks8> 0.23076923
Π<ks7ks9> 0.7692308
Π<ks7ks11> 0.7692308
Π<ks7ks12> 0.23076923
Π<ks8ks6> 0.8333333
Π<ks8ks9> 1.0
Π<ks8ks11> 1.0
Π<ks9ks10> 1.0
Π<ks10ks4> 0.85714287
Π<ks10ks6> 1.0
Π<ks10ks7> 0.85714287
Π<ks10ks9> 0.85714287
Π<ks10ks11> 0.85714287
Π<ks10ks12> 0.85714287
Π<ks11ks10> 1.0
Π<ks12ks5> 1.0

Serialization Values:

Σ<ks5ks4> 0.19354838
Σ<ks5ks1> 0.3181818
Σ<ks5ks2> 0.6666667
Σ<ks5ks9> 0.19354838
Σ<ks5ks11> 0.19354838
Σ<ks5ks12> 0.19354838
Σ<ks4ks5> 0.33333334
Σ<ks1ks7> 0.33333334
Σ<ks2ks4> 0.32258064
Σ<ks2ks1> 0.45454547
Σ<ks2ks3> 0.5
Σ<ks2ks8> 0.5
Σ<ks2ks9> 0.09677419
Σ<ks2ks11> 0.09677419
Σ<ks2ks12> 0.32258064
Σ<ks3ks4> 0.19354838
Σ<ks3ks1> 0.22727273
Σ<ks3ks12> 0.19354838
Σ<ks6ks2> 0.33333334
Σ<ks7ks4> 0.09677419
Σ<ks7ks3> 0.5
Σ<ks7ks6> 0.45454547
Σ<ks7ks8> 0.5
Σ<ks7ks9> 0.32258064
Σ<ks7ks11> 0.32258064
Σ<ks7ks12> 0.09677419
Σ<ks8ks6> 0.22727273
Σ<ks8ks9> 0.19354838
Σ<ks8ks11> 0.19354838
Σ<ks9ks10> 0.33333334
Σ<ks10ks4> 0.19354838
Σ<ks10ks6> 0.3181818
Σ<ks10ks7> 0.6666667
Σ<ks10ks9> 0.19354838
Σ<ks10ks11> 0.19354838
Σ<ks10ks12> 0.19354838
Σ<ks11ks10> 0.33333334
Σ<ks12ks5> 0.33333334

F.3 Cube4 Design Analysis Results:

Specialization Values:

Interdependence Values:

Π<ks16ks5> 1.0
Π<ks15ks5> 1.0
Π<ks14ks10> 0.25
Π<ks13ks10> 0.25
Π<ks12ks5> 1.0
Π<ks11ks10> 0.25
Π<ks10ks16> 0.85714287
Π<ks10ks15> 0.85714287
Π<ks10ks14> 0.85714287

Π<ks10ks13> 0.85714287
Π<ks10ks12> 0.85714287
Π<ks10ks11> 0.85714287
Π<ks10ks9> 0.85714287
Π<ks10ks7> 0.85714287
Π<ks10ks6> 1.0
Π<ks10ks4> 0.85714287
Π<ks9ks10> 0.25
Π<ks8ks14> 1.0
Π<ks8ks13> 1.0
Π<ks8ks11> 1.0
Π<ks8ks9> 1.0
Π<ks8ks6> 0.8333333
Π<ks7ks16> 0.23076923
Π<ks7ks15> 0.23076923
Π<ks7ks14> 0.7692308
Π<ks7ks13> 0.7692308
Π<ks7ks12> 0.23076923
Π<ks7ks11> 0.7692308
Π<ks7ks9> 0.7692308
Π<ks7ks8> 0.23076923
Π<ks7ks6> 0.7692308
Π<ks7ks3> 0.23076923
Π<ks7ks4> 0.23076923
Π<ks6ks2> 1.0
Π<ks3ks16> 1.0
Π<ks3ks15> 1.0
Π<ks3ks12> 1.0
Π<ks3ks1> 0.8333333
Π<ks3ks4> 1.0
Π<ks2ks16> 0.7692308
Π<ks2ks15> 0.7692308
Π<ks2ks14> 0.23076923
Π<ks2ks13> 0.23076923
Π<ks2ks12> 0.7692308
Π<ks2ks11> 0.23076923
Π<ks2ks9> 0.23076923
Π<ks2ks8> 0.23076923
Π<ks2ks3> 0.23076923
Π<ks2ks1> 0.7692308
Π<ks2ks4> 0.7692308
Π<ks1ks7> 1.0
Π<ks4ks5> 1.0
Π<ks5ks16> 0.85714287
Π<ks5ks15> 0.85714287
Π<ks5ks14> 0.85714287
Π<ks5ks13> 0.85714287
Π<ks5ks12> 0.85714287
Π<ks5ks11> 0.85714287
Π<ks5ks9> 0.85714287
Π<ks5ks2> 0.71428573
Π<ks5ks1> 1.0
Π<ks5ks4> 0.85714287

Serialization Values:

Σ<ks16ks5> 0.21052632
Σ<ks15ks5> 0.21052632
Σ<ks14ks10> 0.09090909
Σ<ks13ks10> 0.09090909
Σ<ks12ks5> 0.21052632
Σ<ks11ks10> 0.09090909
Σ<ks10ks16> 0.19354838
Σ<ks10ks15> 0.19354838
Σ<ks10ks14> 0.19354838
Σ<ks10ks13> 0.19354838
Σ<ks10ks12> 0.19354838
Σ<ks10ks11> 0.19354838
Σ<ks10ks9> 0.19354838
Σ<ks10ks7> 0.6666667
Σ<ks10ks6> 0.3181818
Σ<ks10ks4> 0.19354838
Σ<ks9ks10> 0.09090909
Σ<ks8ks14> 0.19354838
Σ<ks8ks13> 0.19354838
Σ<ks8ks11> 0.19354838
Σ<ks8ks9> 0.19354838
Σ<ks8ks6> 0.22727273
Σ<ks7ks16> 0.09677419
Σ<ks7ks15> 0.09677419
Σ<ks7ks14> 0.32258064
Σ<ks7ks13> 0.32258064
Σ<ks7ks12> 0.09677419
Σ<ks7ks11> 0.32258064
Σ<ks7ks9> 0.32258064
Σ<ks7ks8> 0.5
Σ<ks7ks6> 0.45454547
Σ<ks7ks3> 0.5
Σ<ks7ks4> 0.09677419
Σ<ks6ks2> 0.33333334
Σ<ks3ks16> 0.19354838
Σ<ks3ks15> 0.19354838
Σ<ks3ks12> 0.19354838
Σ<ks3ks1> 0.22727273
Σ<ks3ks4> 0.19354838
Σ<ks2ks16> 0.32258064
Σ<ks2ks15> 0.32258064
Σ<ks2ks14> 0.09677419
Σ<ks2ks13> 0.09677419

Σ<ks2ks12> 0.32258064
Σ<ks2ks11> 0.09677419
Σ<ks2ks9> 0.09677419
Σ<ks2ks8> 0.5
Σ<ks2ks3> 0.5
Σ<ks2ks1> 0.45454547
Σ<ks2ks4> 0.32258064
Σ<ks1ks7> 0.33333334
Σ<ks4ks5> 0.21052632
Σ<ks5ks16> 0.19354838
Σ<ks5ks15> 0.19354838
Σ<ks5ks14> 0.19354838
Σ<ks5ks13> 0.19354838
Σ<ks5ks12> 0.19354838
Σ<ks5ks11> 0.19354838
Σ<ks5ks9> 0.19354838
Σ<ks5ks2> 0.6666667
Σ<ks5ks1> 0.3181818
Σ<ks5ks4> 0.19354838

F.4 Cube8 Design Analysis Results:

Specialization Values:

Interdependence Values:

Π<ks5ks4> 0.85714287
Π<ks5ks1> 1.0
Π<ks5ks2> 0.85714287
Π<ks5ks9> 0.85714287
Π<ks5ks11> 0.85714287
Π<ks5ks12> 0.85714287
Π<ks5ks13> 0.85714287
Π<ks5ks14> 0.85714287
Π<ks5ks15> 0.85714287
Π<ks5ks16> 0.85714287
Π<ks5ks17> 0.85714287
Π<ks5ks18> 0.85714287
Π<ks5ks19> 0.85714287
Π<ks5ks20> 0.85714287
Π<ks5ks21> 0.85714287
Π<ks5ks22> 0.85714287
Π<ks5ks23> 0.85714287
Π<ks5ks24> 0.85714287
Π<ks4ks5> 1.0
Π<ks1ks7> 1.0
Π<ks2ks4> 0.7692308
Π<ks2ks1> 0.7692308
Π<ks2ks3> 0.23076923
Π<ks2ks8> 0.23076923
Π<ks2ks9> 0.23076923
Π<ks2ks11> 0.23076923
Π<ks2ks12> 0.7692308
Π<ks2ks13> 0.23076923
Π<ks2ks14> 0.23076923
Π<ks2ks15> 0.7692308
Π<ks2ks16> 0.7692308
Π<ks2ks17> 0.23076923
Π<ks2ks18> 0.23076923
Π<ks2ks19> 0.23076923
Π<ks2ks20> 0.23076923
Π<ks2ks21> 0.7692308
Π<ks2ks22> 0.7692308
Π<ks2ks23> 0.7692308
Π<ks2ks24> 0.7692308
Π<ks3ks4> 1.0
Π<ks3ks1> 0.8333333
Π<ks3ks12> 1.0
Π<ks3ks15> 1.0
Π<ks3ks16> 1.0
Π<ks3ks21> 1.0
Π<ks3ks22> 1.0
Π<ks3ks23> 1.0
Π<ks3ks24> 1.0
Π<ks6ks2> 1.0
Π<ks7ks4> 0.23076923
Π<ks7ks3> 0.23076923
Π<ks7ks6> 0.7692308
Π<ks7ks8> 0.23076923
Π<ks7ks9> 0.7692308
Π<ks7ks11> 0.7692308
Π<ks7ks12> 0.23076923
Π<ks7ks13> 0.7692308
Π<ks7ks14> 0.7692308
Π<ks7ks15> 0.23076923
Π<ks7ks16> 0.23076923
Π<ks7ks17> 0.7692308
Π<ks7ks18> 0.7692308
Π<ks7ks19> 0.7692308
Π<ks7ks20> 0.7692308
Π<ks7ks21> 0.23076923
Π<ks7ks22> 0.23076923
Π<ks7ks23> 0.23076923
Π<ks7ks24> 0.23076923
Π<ks8ks6> 0.8333333
Π<ks8ks9> 1.0
Π<ks8ks11> 1.0

Π<ks8ks13> 1.0
Π<ks8ks14> 1.0
Π<ks8ks17> 1.0
Π<ks8ks18> 1.0
Π<ks8ks19> 1.0
Π<ks8ks20> 1.0
Π<ks9ks10> 1.0
Π<ks10ks4> 0.85714287
Π<ks10ks6> 1.0
Π<ks10ks7> 0.85714287
Π<ks10ks9> 0.85714287
Π<ks10ks11> 0.85714287
Π<ks10ks12> 0.85714287
Π<ks10ks13> 0.85714287
Π<ks10ks14> 0.85714287
Π<ks10ks15> 0.85714287
Π<ks10ks16> 0.85714287
Π<ks10ks17> 0.85714287
Π<ks10ks18> 0.85714287
Π<ks10ks19> 0.85714287
Π<ks10ks20> 0.85714287
Π<ks10ks21> 0.85714287
Π<ks10ks22> 0.85714287
Π<ks10ks23> 0.85714287
Π<ks10ks24> 0.85714287
Π<ks11ks10> 1.0
Π<ks12ks5> 1.0
Π<ks13ks10> 1.0
Π<ks14ks10> 1.0
Π<ks15ks5> 1.0
Π<ks16ks5> 1.0
Π<ks17ks10> 1.0
Π<ks18ks10> 1.0
Π<ks19ks10> 1.0
Π<ks20ks10> 1.0
Π<ks21ks5> 1.0
Π<ks22ks5> 1.0
Π<ks23ks5> 1.0
Π<ks24ks5> 1.0

Serialization Values:

Σ<ks5ks4> 0.19354838
Σ<ks5ks1> 0.3181818
Σ<ks5ks2> 0.6666667
Σ<ks5ks9> 0.19354838
Σ<ks5ks11> 0.19354838
Σ<ks5ks12> 0.19354838
Σ<ks5ks13> 0.19354838
Σ<ks5ks14> 0.19354838
Σ<ks5ks15> 0.19354838
Σ<ks5ks16> 0.19354838
Σ<ks5ks17> 0.19354838
Σ<ks5ks18> 0.19354838
Σ<ks5ks19> 0.19354838
Σ<ks5ks20> 0.19354838
Σ<ks5ks21> 0.19354838
Σ<ks5ks22> 0.19354838
Σ<ks5ks23> 0.19354838
Σ<ks5ks24> 0.19354838
Σ<ks4ks5> 0.11111111
Σ<ks1ks7> 0.33333334
Σ<ks2ks4> 0.32258064
Σ<ks2ks1> 0.45454547
Σ<ks2ks3> 0.5
Σ<ks2ks8> 0.5
Σ<ks2ks9> 0.09677419
Σ<ks2ks11> 0.09677419
Σ<ks2ks12> 0.32258064
Σ<ks2ks13> 0.09677419
Σ<ks2ks14> 0.09677419
Σ<ks2ks15> 0.32258064
Σ<ks2ks16> 0.32258064
Σ<ks2ks17> 0.09677419
Σ<ks2ks18> 0.09677419
Σ<ks2ks19> 0.09677419
Σ<ks2ks20> 0.09677419
Σ<ks2ks21> 0.32258064
Σ<ks2ks22> 0.32258064
Σ<ks2ks23> 0.32258064
Σ<ks2ks24> 0.32258064
Σ<ks3ks4> 0.19354838
Σ<ks3ks1> 0.22727273
Σ<ks3ks12> 0.19354838
Σ<ks3ks15> 0.19354838
Σ<ks3ks16> 0.19354838
Σ<ks3ks21> 0.19354838
Σ<ks3ks22> 0.19354838
Σ<ks3ks23> 0.19354838
Σ<ks3ks24> 0.19354838
Σ<ks6ks2> 0.33333334
Σ<ks7ks4> 0.09677419
Σ<ks7ks3> 0.5
Σ<ks7ks6> 0.45454547
Σ<ks7ks8> 0.5
Σ<ks7ks9> 0.32258064
Σ<ks7ks11> 0.32258064
Σ<ks7ks12> 0.09677419
Σ<ks7ks13> 0.32258064

Σ<ks7ks14> 0.32258064
Σ<ks7ks15> 0.09677419
Σ<ks7ks16> 0.09677419
Σ<ks7ks17> 0.32258064
Σ<ks7ks18> 0.32258064
Σ<ks7ks19> 0.32258064
Σ<ks7ks20> 0.32258064
Σ<ks7ks21> 0.09677419
Σ<ks7ks22> 0.09677419
Σ<ks7ks23> 0.09677419
Σ<ks7ks24> 0.09677419
Σ<ks8ks6> 0.22727273
Σ<ks8ks9> 0.19354838
Σ<ks8ks11> 0.19354838
Σ<ks8ks13> 0.19354838
Σ<ks8ks14> 0.19354838
Σ<ks8ks17> 0.19354838
Σ<ks8ks18> 0.19354838
Σ<ks8ks19> 0.19354838
Σ<ks8ks20> 0.19354838
Σ<ks9ks10> 0.11111111
Σ<ks10ks4> 0.19354838
Σ<ks10ks6> 0.3181818
Σ<ks10ks7> 0.6666667
Σ<ks10ks9> 0.19354838
Σ<ks10ks11> 0.19354838
Σ<ks10ks12> 0.19354838
Σ<ks10ks13> 0.19354838
Σ<ks10ks14> 0.19354838
Σ<ks10ks15> 0.19354838
Σ<ks10ks16> 0.19354838
Σ<ks10ks17> 0.19354838
Σ<ks10ks18> 0.19354838
Σ<ks10ks19> 0.19354838
Σ<ks10ks20> 0.19354838
Σ<ks10ks21> 0.19354838
Σ<ks10ks22> 0.19354838
Σ<ks10ks23> 0.19354838
Σ<ks10ks24> 0.19354838
Σ<ks11ks10> 0.11111111
Σ<ks12ks5> 0.11111111
Σ<ks13ks10> 0.11111111
Σ<ks14ks10> 0.11111111
Σ<ks15ks5> 0.11111111
Σ<ks16ks5> 0.11111111
Σ<ks17ks10> 0.11111111
Σ<ks18ks10> 0.11111111
Σ<ks19ks10> 0.11111111
Σ<ks20ks10> 0.11111111
Σ<ks21ks5> 0.11111111
Σ<ks22ks5> 0.11111111
Σ<ks23ks5> 0.11111111
Σ<ks24ks5> 0.11111111

F.5 Cube Simulation Results

Knowledge Sources:

Knowledge Source: t_calc
Type: PROCESSOR
Execution Delay: 47

Knowledge Source: t_eval-move
Type: PROCESSOR
Execution Delay: 254

Knowledge Source: t_sit-check
Type: PROCESSOR
Execution Delay: 3

Knowledge Source: t_relgn
Type: PROCESSOR
Execution Delay: 10

Knowledge Source: t_display
Type: PROCESSOR
Execution Delay: 1

Knowledge Source: a_sit-check
Type: PROCESSOR
Execution Delay: 3

Knowledge Source: a_relgn
Type: PROCESSOR
Execution Delay: 10

Knowledge Source: a_display
Type: PROCESSOR
Execution Delay: 1

Knowledge Source: a_eval-move
Type: PROCESSOR
Execution Delay: 254

Knowledge Source: a_calc
Type: PROCESSOR
Execution Delay: 47

Results of Simulation run:

Event queue (a_display t_display)
t_display executed at 1
a_display executed at 1
Event queue (a_relgn t_relgn)
t_relgn executed at 11
a_relgn executed at 11
Event queue (a_sit-check t_sit-check)
t_sit-check executed at 14
a_sit-check executed at 14
Event queue (t_eval-move a_eval-move)
a_eval-move executed at 268
t_eval-move executed at 268
Event queue (a_calc t_calc)
t_calc executed at 315
a_calc executed at 315
Event queue (t_display a_display)
a_display executed at 316
t_display executed at 316
Event queue (t_relgn a_relgn)
a_relgn executed at 326
t_relgn executed at 326
Event queue (t_sit-check a_sit-check)
a_sit-check executed at 329
t_sit-check executed at 329
Event queue (a_eval-move t_eval-move)
t_eval-move executed at 583
a_eval-move executed at 583
Event queue (t_calc a_calc)
a_calc executed at 630
t_calc executed at 630
Event queue (a_display t_display)
t_display executed at 631
a_display executed at 631
Event queue (a_relgn t_relgn)
t_relgn executed at 641
a_relgn executed at 641
Event queue (a_sit-check t_sit-check)
t_sit-check executed at 644
a_sit-check executed at 644
Event queue (t_eval-move a_eval-move)
a_eval-move executed at 898
t_eval-move executed at 898
Event queue (a_calc t_calc)
t_calc executed at 945
a_calc executed at 945
Event queue (t_display a_display)
a_display executed at 946
t_display executed at 946
Event queue (t_relgn a_relgn)
a_relgn executed at 956
t_relgn executed at 956
Event queue (t_sit-check a_sit-check)

a_sit-check executed at 959
t_sit-check executed at 959
Event queue (a_eval-move t_eval-move)
t_eval-move executed at 1213
a_eval-move executed at 1213
Event queue (t_calc a_calc)
a_calc executed at 1260
t_calc executed at 1260
Event queue (a_display t_display)
t_display executed at 1261
a_display executed at 1261
Event queue (a_relgn t_relgn)
t_relgn executed at 1271
a_relgn executed at 1271
Event queue (a_sit-check t_sit-check)
t_sit-check executed at 1274
a_sit-check executed at 1274
Event queue (t_eval-move a_eval-move)
a_eval-move executed at 1528
t_eval-move executed at 1528
Event queue (a_calc t_calc)
t_calc executed at 1575
a_calc executed at 1575
Event queue (t_display a_display)
a_display executed at 1576
t_display executed at 1576
Event queue (t_relgn a_relgn)
a_relgn executed at 1586
t_relgn executed at 1586
Event queue (t_sit-check a_sit-check)
a_sit-check executed at 1589
t_sit-check executed at 1589
Event queue (a_eval-move t_eval-move)
t_eval-move executed at 1843
a_eval-move executed at 1843
Event queue (t_calc a_calc)
a_calc executed at 1890
t_calc executed at 1890
Event queue (a_display t_display)
t_display executed at 1891
a_display executed at 1891
Event queue (a_relgn t_relgn)
t_relgn executed at 1901
a_relgn executed at 1901
Event queue (a_sit-check t_sit-check)
t_sit-check executed at 1904
a_sit-check executed at 1904
Event queue (t_eval-move a_eval-move)
a_eval-move executed at 2158
t_eval-move executed at 2158
Event queue (a_calc t_calc)
t_calc executed at 2205
a_calc executed at 2205
Event queue (t_display a_display)
a_display executed at 2206
t_display executed at 2206
Event queue (t_relgn a_relgn)
a_relgn executed at 2216
t_relgn executed at 2216
Event queue (t_sit-check a_sit-check)
a_sit-check executed at 2219
t_sit-check executed at 2219
Event queue (a_eval-move t_eval-move)
t_eval-move executed at 2473
a_eval-move executed at 2473
Event queue (t_calc a_calc)
a_calc executed at 2520
t_calc executed at 2520
Event queue (a_display t_display)
t_display executed at 2521
a_display executed at 2521
Event queue (a_relgn t_relgn)
t_relgn executed at 2531
a_relgn executed at 2531
Event queue (a_sit-check t_sit-check)
t_sit-check executed at 2534
a_sit-check executed at 2534
Event queue (t_eval-move a_eval-move)
a_eval-move executed at 2788
t_eval-move executed at 2788
Event queue (a_calc t_calc)
t_calc executed at 2835
a_calc executed at 2835
Event queue (t_display a_display)
a_display executed at 2836
t_display executed at 2836
Event queue (t_relgn a_relgn)
a_relgn executed at 2846
t_relgn executed at 2846
Event queue (t_sit-check a_sit-check)
a_sit-check executed at 2849
t_sit-check executed at 2849
Event queue (a_eval-move t_eval-move)
t_eval-move executed at 3103
a_eval-move executed at 3103
Event queue (t_calc a_calc)
a_calc executed at 3150
t_calc executed at 3150

F.6 Cube2 Simulation Results

Knowledge Source: a_eval-2
Type: PROCESSOR
Execution Delay: 130

Knowledge Source: t_eval-2
Type: PROCESSOR
Execution Delay: 130

Knowledge Source: t_calc
Type: PROCESSOR
Execution Delay: 47

Knowledge Source: t_eval-1
Type: PROCESSOR
Execution Delay: 130

Knowledge Source: t_sit
Type: PROCESSOR
Execution Delay: 3

Knowledge Source: t_relgn
Type: PROCESSOR
Execution Delay: 10

Knowledge Source: t_sync
Type: PROCESSOR
Execution Delay: 1

Knowledge Source: a_sit
Type: PROCESSOR
Execution Delay: 3

Knowledge Source: a_relgn
Type: PROCESSOR
Execution Delay: 10

Knowledge Source: a_sync
Type: PROCESSOR
Execution Delay: 1

Knowledge Source: a_eval-1
Type: PROCESSOR
Execution Delay: 130

Knowledge Source: a_calc
Type: PROCESSOR
Execution Delay: 47

Results of Simulation run:

Event queue (a_sync t_sync)
t_sync executed at 1
a_sync executed at 1
Event queue (a_relgn t_relgn)
t_relgn executed at 11
a_relgn executed at 11
Event queue (a_sit t_sit)
t_sit executed at 14
a_sit executed at 14
Event queue (t_eval-2 t_eval-1 a_eval-1 a_eval-2)
a_eval-2 executed at 144
a_eval-1 executed at 144
t_eval-1 executed at 144
t_eval-2 executed at 144
Event queue (a_calc t_calc)
t_calc executed at 191
a_calc executed at 191
Event queue (t_sync a_sync)
a_sync executed at 192
t_sync executed at 192
Event queue (t_relgn a_relgn)
a_relgn executed at 202
t_relgn executed at 202
Event queue (t_sit a_sit)
a_sit executed at 205
t_sit executed at 205
Event queue (a_eval-2 a_eval-1 t_eval-1 t_eval-2)
t_eval-2 executed at 335
t_eval-1 executed at 335
a_eval-1 executed at 335
a_eval-2 executed at 335
Event queue (t_calc a_calc)
a_calc executed at 382
t_calc executed at 382
Event queue (a_sync t_sync)
t_sync executed at 383
a_sync executed at 383
Event queue (a_relgn t_relgn)
t_relgn executed at 393
a_relgn executed at 393
Event queue (a_sit t_sit)
t_sit executed at 396
a_sit executed at 396

Event queue (t_eval-2 t_eval-1 a_eval-1 a_eval-2)
a_eval-2 executed at 526
a_eval-1 executed at 526
t_eval-1 executed at 526
t_eval-2 executed at 526
Event queue (a_calc t_calc)
t_calc executed at 573
a_calc executed at 573
Event queue (t_sync a_sync)
a_sync executed at 574
t_sync executed at 574
Event queue (t_relgn a_relgn)
a_relgn executed at 584
t_relgn executed at 584
Event queue (t_sit a_sit)
a_sit executed at 587
t_sit executed at 587
Event queue (a_eval-2 a_eval-1 t_eval-1 t_eval-2)
t_eval-2 executed at 717
t_eval-1 executed at 717
a_eval-1 executed at 717
a_eval-2 executed at 717
Event queue (t_calc a_calc)
a_calc executed at 764
t_calc executed at 764
Event queue (a_sync t_sync)
t_sync executed at 765
a_sync executed at 765
Event queue (a_relgn t_relgn)
t_relgn executed at 775
a_relgn executed at 775
Event queue (a_sit t_sit)
t_sit executed at 778
a_sit executed at 778
Event queue (t_eval-2 t_eval-1 a_eval-1 a_eval-2)
a_eval-2 executed at 908
a_eval-1 executed at 908
t_eval-1 executed at 908
t_eval-2 executed at 908
Event queue (a_calc t_calc)
t_calc executed at 955
a_calc executed at 955
Event queue (t_sync a_sync)
a_sync executed at 956
t_sync executed at 956
Event queue (t_relgn a_relgn)
a_relgn executed at 966
t_relgn executed at 966
Event queue (t_sit a_sit)
a_sit executed at 969
t_sit executed at 969
Event queue (a_eval-2 a_eval-1 t_eval-1 t_eval-2)
t_eval-2 executed at 1099
t_eval-1 executed at 1099
a_eval-1 executed at 1099
a_eval-2 executed at 1099
Event queue (t_calc a_calc)
a_calc executed at 1146
t_calc executed at 1146
Event queue (a_sync t_sync)
t_sync executed at 1147
a_sync executed at 1147
Event queue (a_relgn t_relgn)
t_relgn executed at 1157
a_relgn executed at 1157
Event queue (a_sit t_sit)
t_sit executed at 1160
a_sit executed at 1160
Event queue (t_eval-2 t_eval-1 a_eval-1 a_eval-2)
a_eval-2 executed at 1290
a_eval-1 executed at 1290
t_eval-1 executed at 1290
t_eval-2 executed at 1290
Event queue (a_calc t_calc)
t_calc executed at 1337
a_calc executed at 1337
Event queue (t_sync a_sync)
a_sync executed at 1338
t_sync executed at 1338
Event queue (t_relgn a_relgn)
a_relgn executed at 1348
t_relgn executed at 1348
Event queue (t_sit a_sit)
a_sit executed at 1351
t_sit executed at 1351
Event queue (a_eval-2 a_eval-1 t_eval-1 t_eval-2)
t_eval-2 executed at 1481
t_eval-1 executed at 1481
a_eval-1 executed at 1481
a_eval-2 executed at 1481
Event queue (t_calc a_calc)
a_calc executed at 1528

t_calc executed at 1528
Event queue (a_sync t_sync)
t_sync executed at 1529
a_sync executed at 1529
Event queue (a_relgn t_relgn)
t_relgn executed at 1539
a_relgn executed at 1539
Event queue (a_sit t_sit)
t_sit executed at 1542
a_sit executed at 1542
Event queue (t_eval-2 t_eval-1 a_eval-1 a_eval-2)
a_eval-2 executed at 1672
a_eval-1 executed at 1672
t_eval-1 executed at 1672
t_eval-2 executed at 1672
Event queue (a_calc t_calc)
t_calc executed at 1719
a_calc executed at 1719
Event queue (t_sync a_sync)
a_sync executed at 1720
t_sync executed at 1720
Event queue (t_relgn a_relgn)
a_relgn executed at 1730
t_relgn executed at 1730
Event queue (t_sit a_sit)
a_sit executed at 1733
t_sit executed at 1733
Event queue (a_eval-2 a_eval-1 t_eval-1 t_eval-2)
t_eval-2 executed at 1863
t_eval-1 executed at 1863
a_eval-1 executed at 1863
a_eval-2 executed at 1863
Event queue (t_calc a_calc)
a_calc executed at 1910
t_calc executed at 1910

F.3 Cube4 Simulation Results

Knowledge Source: a_eval-4
Type: PROCESSOR
Execution Delay: 73

Knowledge Source: a_eval-3
Type: PROCESSOR
Execution Delay: 73

Knowledge Source: t_eval-4
Type: PROCESSOR
Execution Delay: 73

Knowledge Source: t_eval-3
Type: PROCESSOR
Execution Delay: 73

Knowledge Source: a_eval-2
Type: PROCESSOR
Execution Delay: 73

Knowledge Source: t_eval-2
Type: PROCESSOR
Execution Delay: 73

Knowledge Source: t_calc
Type: PROCESSOR
Execution Delay: 47

Knowledge Source: t_eval-1
Type: PROCESSOR
Execution Delay: 73

Knowledge Source: t_sit
Type: PROCESSOR
Execution Delay: 3

Knowledge Source: t_relgn
Type: PROCESSOR
Execution Delay: 10

Knowledge Source: t_sync
Type: PROCESSOR
Execution Delay: 1

Knowledge Source: a_sit
Type: PROCESSOR
Execution Delay: 3

Knowledge Source: a_relgn
Type: PROCESSOR
Execution Delay: 10

Knowledge Source: a_sync
Type: PROCESSOR
Execution Delay: 1

Knowledge Source: a_eval-1

Type: PROCESSOR
Execution Delay: 73

Knowledge Source: a_calc
Type: PROCESSOR
Execution Delay: 47

Simulation Results:

Event queue (a_sync t_sync)
t_sync executed at 1
a_sync executed at 1
Event queue (a_relgn t_relgn)
t_relgn executed at 11
a_relgn executed at 11
Event queue (a_sit t_sit)
t_sit executed at 14
a_sit executed at 14
Event queue (t_eval-4 t_eval-2 t_eval-1 t_eval-3 a_eval-3 a_eval-1 a_eval-2 a_eval-4)
a_eval-4 executed at 87
a_eval-2 executed at 87
a_eval-1 executed at 87
a_eval-3 executed at 87
t_eval-3 executed at 87
t_eval-1 executed at 87
t_eval-2 executed at 87
t_eval-4 executed at 87
Event queue (a_calc t_calc)
t_calc executed at 134
a_calc executed at 134
Event queue (t_sync a_sync)
a_sync executed at 135
t_sync executed at 135
Event queue (t_relgn a_relgn)
a_relgn executed at 145
t_relgn executed at 145
Event queue (t_sit a_sit)
a_sit executed at 148
t_sit executed at 148
Event queue (a_eval-4 a_eval-2 a_eval-1 a_eval-3 t_eval-3 t_eval-1 t_eval-2 t_eval-4)
t_eval-4 executed at 221
t_eval-2 executed at 221
t_eval-1 executed at 221
t_eval-3 executed at 221
a_eval-3 executed at 221
a_eval-1 executed at 221
a_eval-2 executed at 221
a_eval-4 executed at 221
Event queue (t_calc a_calc)
a_calc executed at 268
t_calc executed at 268
Event queue (a_sync t_sync)
t_sync executed at 269
a_sync executed at 269
Event queue (a_relgn t_relgn)
t_relgn executed at 279
a_relgn executed at 279
Event queue (a_sit t_sit)
t_sit executed at 282
a_sit executed at 282
Event queue (t_eval-4 t_eval-2 t_eval-1 t_eval-3 a_eval-3 a_eval-1 a_eval-2 a_eval-4)
a_eval-4 executed at 355
a_eval-2 executed at 355
a_eval-1 executed at 355
a_eval-3 executed at 355
t_eval-3 executed at 355
t_eval-1 executed at 355
t_eval-2 executed at 355
t_eval-4 executed at 355
Event queue (a_calc t_calc)
t_calc executed at 402
a_calc executed at 402
Event queue (t_sync a_sync)
a_sync executed at 403
t_sync executed at 403
Event queue (t_relgn a_relgn)
a_relgn executed at 413
t_relgn executed at 413
Event queue (t_sit a_sit)
a_sit executed at 416
t_sit executed at 416
Event queue (a_eval-4 a_eval-2 a_eval-1 a_eval-3 t_eval-3 t_eval-1 t_eval-2 t_eval-4)
t_eval-4 executed at 489
t_eval-2 executed at 489
t_eval-1 executed at 489
t_eval-3 executed at 489
a_eval-3 executed at 489
a_eval-1 executed at 489
a_eval-2 executed at 489
a_eval-4 executed at 489
Event queue (t_calc a_calc)

a_calc executed at 536
t_calc executed at 536
Event queue (a_sync t_sync)
t_sync executed at 537
a_sync executed at 537
Event queue (a_relgn t_relgn)
t_relgn executed at 547
a_relgn executed at 547
Event queue (a_sit t_sit)
t_sit executed at 550
a_sit executed at 550
Event queue (t_eval-4 t_eval-2 t_eval-1 t_eval-3 a_eval-3 a_eval-1 a_eval-2 a_eval-4)
a_eval-4 executed at 623
a_eval-2 executed at 623
a_eval-1 executed at 623
a_eval-3 executed at 623
t_eval-3 executed at 623
t_eval-1 executed at 623
t_eval-2 executed at 623
t_eval-4 executed at 623
Event queue (a_calc t_calc)
t_calc executed at 670
a_calc executed at 670
Event queue (t_sync a_sync)
a_sync executed at 671
t_sync executed at 671
Event queue (t_relgn a_relgn)
a_relgn executed at 681
t_relgn executed at 681
Event queue (t_sit a_sit)
a_sit executed at 684
t_sit executed at 684
Event queue (a_eval-4 a_eval-2 a_eval-1 a_eval-3 t_eval-3 t_eval-1 t_eval-2 t_eval-4)
t_eval-4 executed at 757
t_eval-2 executed at 757
t_eval-1 executed at 757
t_eval-3 executed at 757
a_eval-3 executed at 757
a_eval-1 executed at 757
a_eval-2 executed at 757
a_eval-4 executed at 757
Event queue (t_calc a_calc)
a_calc executed at 804
t_calc executed at 804
Event queue (a_sync t_sync)
t_sync executed at 805
a_sync executed at 805
Event queue (a_relgn t_relgn)
t_relgn executed at 815
a_relgn executed at 815
Event queue (a_sit t_sit)
t_sit executed at 818
a_sit executed at 818
Event queue (t_eval-4 t_eval-2 t_eval-1 t_eval-3 a_eval-3 a_eval-1 a_eval-2 a_eval-4)
a_eval-4 executed at 891
a_eval-2 executed at 891
a_eval-1 executed at 891
a_eval-3 executed at 891
t_eval-3 executed at 891
t_eval-1 executed at 891
t_eval-2 executed at 891
t_eval-4 executed at 891
Event queue (a_calc t_calc)
t_calc executed at 938
a_calc executed at 938
Event queue (t_sync a_sync)
a_sync executed at 939
t_sync executed at 939
Event queue (t_relgn a_relgn)
a_relgn executed at 949
t_relgn executed at 949
Event queue (t_sit a_sit)
a_sit executed at 952
t_sit executed at 952
Event queue (a_eval-4 a_eval-2 a_eval-1 a_eval-3 t_eval-3 t_eval-1 t_eval-2 t_eval-4)
t_eval-4 executed at 1025
t_eval-2 executed at 1025
t_eval-1 executed at 1025
t_eval-3 executed at 1025
a_eval-3 executed at 1025
a_eval-1 executed at 1025
a_eval-2 executed at 1025
a_eval-4 executed at 1025
Event queue (t_calc a_calc)
a_calc executed at 1072
t_calc executed at 1072
Event queue (a_sync t_sync)
t_sync executed at 1073
a_sync executed at 1073
Event queue (a_relgn t_relgn)

t_relgn executed at 1083
a_relgn executed at 1083
Event queue (a_sit t_sit)
t_sit executed at 1086
a_sit executed at 1086
Event queue (t_eval-4 t_eval-2 t_eval-1
t_eval-3 a_eval-3 a_eval-1 a_eval-2
a_eval-4)
a_eval-4 executed at 1159
a_eval-2 executed at 1159
a_eval-1 executed at 1159
a_eval-3 executed at 1159
t_eval-3 executed at 1159
t_eval-1 executed at 1159
t_eval-2 executed at 1159
t_eval-4 executed at 1159
Event queue (a_calc t_calc)
t_calc executed at 1206
a_calc executed at 1206
Event queue (t_sync a_sync)
a_sync executed at 1207
t_sync executed at 1207
Event queue (t_relgn a_relgn)
a_relgn executed at 1217
t_relgn executed at 1217
Event queue (t_sit a_sit)
a_sit executed at 1220
t_sit executed at 1220
Event queue (a_eval-4 a_eval-2 a_eval-1
a_eval-3 t_eval-3 t_eval-1 t_eval-2
t_eval-4)
t_eval-4 executed at 1293
t_eval-2 executed at 1293
t_eval-1 executed at 1293
t_eval-3 executed at 1293
a_eval-3 executed at 1293
a_eval-1 executed at 1293
a_eval-2 executed at 1293
a_eval-4 executed at 1293
Event queue (t_calc a_calc)
a_calc executed at 1340
t_calc executed at 1340

F.8 Cube8 Simulation Results

Knowledge Source: a_eval-8
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: a_eval-7
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: a_eval-6
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: a_eval-5
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_eval-8
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_eval-7
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_eval-6
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_eval-5
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: a_eval-4
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: a_eval-3
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_eval-4
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_eval-3
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: a_eval-2
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_eval-2
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_calc
Type: PROCESSOR
Execution Delay: 47

Knowledge Source: t_eval-1
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: t_sit
Type: PROCESSOR
Execution Delay: 3

Knowledge Source: t_relgn
Type: PROCESSOR
Execution Delay: 10

Knowledge Source: t_sync
Type: PROCESSOR
Execution Delay: 1

Knowledge Source: a_sit
Type: PROCESSOR
Execution Delay: 3

Knowledge Source: a_relgn
Type: PROCESSOR
Execution Delay: 10

Knowledge Source: a_sync
Type: PROCESSOR
Execution Delay: 1

Knowledge Source: a_eval-1
Type: PROCESSOR
Execution Delay: 52

Knowledge Source: a_calc
Type: PROCESSOR
Execution Delay: 47

Results of Simulation run:

Event queue (a_sync t_sync)
t_sync executed at 1
a_sync executed at 1
Event queue (a_relgn t_relgn)
t_relgn executed at 11
a_relgn executed at 11
Event queue (a_sit t_sit)
t_sit executed at 14
a_sit executed at 14
Event queue (t_eval-8 t_eval-6 t_eval-4
t_eval-1 t_eval-2 t_eval-3 t_eval-5
t_eval-7 a_eval-7 a_eval-5 a_eval-3
a_eval-1 a_eval-2 a_eval-4 a_eval-6
a_eval-8)
a_eval-8 executed at 66
a_eval-6 executed at 66
a_eval-4 executed at 66
a_eval-2 executed at 66
a_eval-1 executed at 66
a_eval-3 executed at 66
a_eval-5 executed at 66
a_eval-7 executed at 66
t_eval-7 executed at 66
t_eval-5 executed at 66
t_eval-3 executed at 66
t_eval-2 executed at 66
t_eval-1 executed at 66
t_eval-4 executed at 66
t_eval-6 executed at 66
t_eval-8 executed at 66
Event queue (a_calc t_calc)
t_calc executed at 113
a_calc executed at 113
Event queue (t_sync a_sync)
a_sync executed at 114
t_sync executed at 114
Event queue (t_relgn a_relgn)
a_relgn executed at 124
t_relgn executed at 124
Event queue (t_sit a_sit)
a_sit executed at 127
t_sit executed at 127
Event queue (a_eval-8 a_eval-6 a_eval-4
a_eval-2 a_eval-1 a_eval-3 a_eval-5
a_eval-7 t_eval-7 t_eval-5 t_eval-3
t_eval-2 t_eval-1 t_eval-4 t_eval-6
t_eval-8)
t_eval-8 executed at 179
t_eval-6 executed at 179
t_eval-4 executed at 179
t_eval-1 executed at 179
t_eval-2 executed at 179
t_eval-3 executed at 179

t_eval-5 executed at 179
t_eval-7 executed at 179
a_eval-7 executed at 179
a_eval-5 executed at 179
a_eval-3 executed at 179
a_eval-1 executed at 179
a_eval-2 executed at 179
a_eval-4 executed at 179
a_eval-6 executed at 179
a_eval-8 executed at 179
Event queue (t_calc a_calc)
a_calc executed at 226
t_calc executed at 226
Event queue (a_sync t_sync)
t_sync executed at 227
a_sync executed at 227
Event queue (a_relgn t_relgn)
t_relgn executed at 237
a_relgn executed at 237
Event queue (a_sit t_sit)
t_sit executed at 240
a_sit executed at 240
Event queue (t_eval-8 t_eval-6 t_eval-4 t_eval-1 t_eval-2 t_eval-3 t_eval-5 t_eval-7 a_eval-7 a_eval-5 a_eval-3 a_eval-1 a_eval-2 a_eval-4 a_eval-6 a_eval-8)
a_eval-8 executed at 292
a_eval-6 executed at 292
a_eval-4 executed at 292
a_eval-2 executed at 292
a_eval-1 executed at 292
a_eval-3 executed at 292
a_eval-5 executed at 292
a_eval-7 executed at 292
t_eval-7 executed at 292
t_eval-5 executed at 292
t_eval-3 executed at 292
t_eval-2 executed at 292
t_eval-1 executed at 292
t_eval-4 executed at 292
t_eval-6 executed at 292
t_eval-8 executed at 292
Event queue (a_calc t_calc)
t_calc executed at 339
a_calc executed at 339
Event queue (t_sync a_sync)
a_sync executed at 340
t_sync executed at 340
Event queue (t_relgn a_relgn)
a_relgn executed at 350
t_relgn executed at 350
Event queue (t_sit a_sit)
a_sit executed at 353
t_sit executed at 353
Event queue (a_eval-8 a_eval-6 a_eval-4 a_eval-2 a_eval-1 a_eval-3 a_eval-5 a_eval-7 t_eval-7 t_eval-5 t_eval-3 t_eval-2 t_eval-1 t_eval-4 t_eval-6 t_eval-8)
t_eval-8 executed at 405
t_eval-6 executed at 405
t_eval-4 executed at 405
t_eval-1 executed at 405
t_eval-2 executed at 405
t_eval-3 executed at 405
t_eval-5 executed at 405
t_eval-7 executed at 405
a_eval-7 executed at 405
a_eval-5 executed at 405
a_eval-3 executed at 405
a_eval-1 executed at 405
a_eval-2 executed at 405
a_eval-4 executed at 405
a_eval-6 executed at 405
a_eval-8 executed at 405
Event queue (t_calc a_calc)
a_calc executed at 452
t_calc executed at 452
Event queue (a_sync t_sync)
t_sync executed at 453
a_sync executed at 453
Event queue (a_relgn t_relgn)
t_relgn executed at 463
a_relgn executed at 463
Event queue (a_sit t_sit)
t_sit executed at 466
a_sit executed at 466
Event queue (t_eval-8 t_eval-6 t_eval-4 t_eval-1 t_eval-2 t_eval-3 t_eval-5 t_eval-7 a_eval-7 a_eval-5 a_eval-3 a_eval-1 a_eval-2 a_eval-4 a_eval-6 a_eval-8)
a_eval-8 executed at 518
a_eval-6 executed at 518
a_eval-4 executed at 518
a_eval-2 executed at 518
a_eval-1 executed at 518

a_eval-3 executed at 518
a_eval-5 executed at 518
a_eval-7 executed at 518
t_eval-7 executed at 518
t_eval-5 executed at 518
t_eval-3 executed at 518
t_eval-2 executed at 518
t_eval-1 executed at 518
t_eval-4 executed at 518
t_eval-6 executed at 518
t_eval-8 executed at 518
Event queue (a_calc t_calc)
t_calc executed at 565
a_calc executed at 565
Event queue (t_sync a_sync)
a_sync executed at 566
t_sync executed at 566
Event queue (t_relgn a_relgn)
a_relgn executed at 576
t_relgn executed at 576
Event queue (t_sit a_sit)
a_sit executed at 579
t_sit executed at 579
Event queue (a_eval-8 a_eval-6 a_eval-4 a_eval-2 a_eval-1 a_eval-3 a_eval-5 a_eval-7 t_eval-7 t_eval-5 t_eval-3 t_eval-2 t_eval-1 t_eval-4 t_eval-6 t_eval-8)
t_eval-8 executed at 631
t_eval-6 executed at 631
t_eval-4 executed at 631
t_eval-1 executed at 631
t_eval-2 executed at 631
t_eval-3 executed at 631
t_eval-5 executed at 631
t_eval-7 executed at 631
a_eval-7 executed at 631
a_eval-5 executed at 631
a_eval-3 executed at 631
a_eval-1 executed at 631
a_eval-2 executed at 631
a_eval-4 executed at 631
a_eval-6 executed at 631
a_eval-8 executed at 631
Event queue (t_calc a_calc)
a_calc executed at 678
t_calc executed at 678
Event queue (a_sync t_sync)
t_sync executed at 679
a_sync executed at 679
Event queue (a_relgn t_relgn)
t_relgn executed at 689
a_relgn executed at 689
Event queue (a_sit t_sit)
t_sit executed at 692
a_sit executed at 692
Event queue (t_eval-8 t_eval-6 t_eval-4 t_eval-1 t_eval-2 t_eval-3 t_eval-5 t_eval-7 a_eval-7 a_eval-5 a_eval-3 a_eval-1 a_eval-2 a_eval-4 a_eval-6 a_eval-8)
a_eval-8 executed at 744
a_eval-6 executed at 744
a_eval-4 executed at 744
a_eval-2 executed at 744
a_eval-1 executed at 744
a_eval-3 executed at 744
a_eval-5 executed at 744
a_eval-7 executed at 744
t_eval-7 executed at 744
t_eval-5 executed at 744
t_eval-3 executed at 744
t_eval-2 executed at 744
t_eval-1 executed at 744
t_eval-4 executed at 744
t_eval-6 executed at 744
t_eval-8 executed at 744
Event queue (a_calc t_calc)
t_calc executed at 791
a_calc executed at 791
Event queue (t_sync a_sync)
a_sync executed at 792
t_sync executed at 792
Event queue (t_relgn a_relgn)
a_relgn executed at 802
t_relgn executed at 802
Event queue (t_sit a_sit)
a_sit executed at 805
t_sit executed at 805
Event queue (a_eval-8 a_eval-6 a_eval-4 a_eval-2 a_eval-1 a_eval-3 a_eval-5 a_eval-7 t_eval-7 t_eval-5 t_eval-3 t_eval-2 t_eval-1 t_eval-4 t_eval-6 t_eval-8)
t_eval-8 executed at 857
t_eval-6 executed at 857
t_eval-4 executed at 857
t_eval-1 executed at 857

t_eval-2 executed at 857
t_eval-3 executed at 857
t_eval-5 executed at 857
t_eval-7 executed at 857
a_eval-7 executed at 857
a_eval-5 executed at 857
a_eval-3 executed at 857
a_eval-1 executed at 857
a_eval-2 executed at 857
a_eval-4 executed at 857
a_eval-6 executed at 857
a_eval-8 executed at 857
Event queue (t_calc a_calc)
a_calc executed at 904
t_calc executed at 904
Event queue (a_sync t_sync)
t_sync executed at 905
a_sync executed at 905
Event queue (a_relgn t_relgn)
t_relgn executed at 915
a_relgn executed at 915
Event queue (a_sit t_sit)
t_sit executed at 918
a_sit executed at 918
Event queue (t_eval-8 t_eval-6 t_eval-4
t_eval-1 t_eval-2 t_eval-3 t_eval-5
t_eval-7 a_eval-7 a_eval-5 a_eval-3
a_eval-1 a_eval-2 a_eval-4 a_eval-6
a_eval-8)
a_eval-8 executed at 970
a_eval-6 executed at 970
a_eval-4 executed at 970
a_eval-2 executed at 970
a_eval-1 executed at 970
a_eval-3 executed at 970
a_eval-5 executed at 970
a_eval-7 executed at 970
t_eval-7 executed at 970
t_eval-5 executed at 970
t_eval-3 executed at 970
t_eval-2 executed at 970
t_eval-1 executed at 970
t_eval-4 executed at 970
t_eval-6 executed at 970
t_eval-8 executed at 970
Event queue (a_calc t_calc)
t_calc executed at 1017
a_calc executed at 1017
Event queue (t_sync a_sync)
a_sync executed at 1018
t_sync executed at 1018
Event queue (t_relgn a_relgn)
a_relgn executed at 1028
t_relgn executed at 1028
Event queue (t_sit a_sit)
a_sit executed at 1031
t_sit executed at 1031
Event queue (a_eval-8 a_eval-6 a_eval-4
a_eval-2 a_eval-1 a_eval-3 a_eval-5
a_eval-7 t_eval-7 t_eval-5 t_eval-3
t_eval-2 t_eval-1 t_eval-4 t_eval-6
t_eval-8)
t_eval-8 executed at 1083
t_eval-6 executed at 1083
t_eval-4 executed at 1083
t_eval-1 executed at 1083
t_eval-2 executed at 1083
t_eval-3 executed at 1083
t_eval-5 executed at 1083
t_eval-7 executed at 1083
a_eval-7 executed at 1083
a_eval-5 executed at 1083
a_eval-3 executed at 1083
a_eval-1 executed at 1083
a_eval-2 executed at 1083
a_eval-4 executed at 1083
a_eval-6 executed at 1083
a_eval-8 executed at 1083
Event queue (t_calc a_calc)
a_calc executed at 1130
t_calc executed at 1130

F.9 Paladin Design Analysis Results:

Specialization Values:

Ω<KS8KS7> 0.25
Ω<KS7KS8> 0.25

Interdependence Values:

Π<KS10KS5> 1.0
Π<KS9KS6> 1.0
Π<KS8KS4> 0.25
Π<KS8KS3> 0.75
Π<KS8KS2> 0.125
Π<KS8KS1> 0.875
Π<KS7KS4> 0.75
Π<KS7KS3> 0.25
Π<KS7KS2> 0.875
Π<KS7KS1> 0.125
Π<KS6KS7> 1.0
Π<KS5KS8> 1.0
Π<KS4KS9> 1.0
Π<KS3KS10> 1.0
Π<KS2KS9> 1.0
Π<KS1KS10> 1.0

Serialization Values:

Σ<KS10KS5> 1.0
Σ<KS9KS6> 1.0
Σ<KS8KS4> 0.33333334
Σ<KS8KS3> 1.0
Σ<KS8KS2> 0.14285715
Σ<KS8KS1> 1.0
Σ<KS7KS4> 1.0
Σ<KS7KS3> 0.33333334
Σ<KS7KS2> 1.0
Σ<KS7KS1> 0.14285715
Σ<KS6KS7> 1.0
Σ<KS5KS8> 1.0
Σ<KS4KS9> 0.33333334
Σ<KS3KS10> 0.33333334
Σ<KS2KS9> 0.6666667
Σ<KS1KS10> 0.6666667

F.10 Paladin Simulation Results

Knowledge Sources:

Knowledge Source: Maneuver_A
Type: PROCESSOR
Execution Delay: 22

Knowledge Source: Maneuver_T
Type: PROCESSOR
Execution Delay: 22

Knowledge Source: Main_A
Type: PROCESSOR
Execution Delay: 35

Knowledge Source: Main_T
Type: PROCESSOR
Execution Delay: 35

Knowledge Source: Dynamics_T
Type: PROCESSOR
Execution Delay: 180

Knowledge Source: Dynamics_A
Type: PROCESSOR
Execution Delay: 180

Knowledge Source: Throttle_T
Type: PROCESSOR
Execution Delay: 16

Knowledge Source: Throttle_A
Type: PROCESSOR
Execution Delay: 16

Knowledge Source: Sit_T
Type: PROCESSOR
Execution Delay: 14

Knowledge Source: Sit_A
Type: PROCESSOR
Execution Delay: 14

Results of Simulation run:

Event queue (Dynamics_A Dynamics_T)
Dynamics_T executed at 180
Dynamics_A executed at 180
Event queue (Main_T Main_A)

```
Main_A executed at 215
Main_T executed at 215
Event queue (Throttle_A Sit_A Sit_T
Throttle_T)
Sit_T executed at 229
Sit_A executed at 229
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 231
Throttle_A executed at 231
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 253
Maneuver_T executed at 253
Event queue (Dynamics_A
Dynamics_T)
Dynamics_T executed at 433
Dynamics_A executed at 433
Event queue (Main_T Main_A)
Main_A executed at 468
Main_T executed at 468
Event queue (Throttle_A Sit_A Sit_T
Throttle_T)
Sit_T executed at 482
Sit_A executed at 482
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 484
Throttle_A executed at 484
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 506
Maneuver_T executed at 506
Event queue (Dynamics_A
Dynamics_T)
Dynamics_T executed at 686
Dynamics_A executed at 686
Event queue (Main_T Main_A)
Main_A executed at 721
Main_T executed at 721
Event queue (Throttle_A Sit_A Sit_T
Throttle_T)
Sit_T executed at 735
Sit_A executed at 735
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 737
Throttle_A executed at 737
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 759
Maneuver_T executed at 759
Event queue (Dynamics_A
Dynamics_T)
Dynamics_T executed at 939
Dynamics_A executed at 939
Event queue (Main_T Main_A)
Main_A executed at 974
Main_T executed at 974
Event queue (Throttle_A Sit_A Sit_T
Throttle_T)
Sit_T executed at 988
Sit_A executed at 988
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 990
Throttle_A executed at 990
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 1012
Maneuver_T executed at 1012
Event queue (Dynamics_A
Dynamics_T)
Dynamics_T executed at 1192
Dynamics_A executed at 1192
Event queue (Main_T Main_A)
Main_A executed at 1227
Main_T executed at 1227
Event queue (Throttle_A Sit_A Sit_T
Throttle_T)
Sit_T executed at 1241
Sit_A executed at 1241
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 1243
Throttle_A executed at 1243
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 1265
Maneuver_T executed at 1265
Event queue (Dynamics_A
Dynamics_T)
Dynamics_T executed at 1445
Dynamics_A executed at 1445
Event queue (Main_T Main_A)
Main_A executed at 1480
Main_T executed at 1480
Event queue (Throttle_A Sit_A Sit_T
Throttle_T)
Sit_T executed at 1494
Sit_A executed at 1494
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 1496
Throttle_A executed at 1496
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 1518
Maneuver_T executed at 1518
```

Event queue (Dynamics_A Dynamics_T)
Dynamics_T executed at 1698
Dynamics_A executed at 1698
Event queue (Main_T Main_A)
Main_A executed at 1733
Main_T executed at 1733
Event queue (Throttle_A Sit_A Sit_T Throttle_T)
Sit_T executed at 1747
Sit_A executed at 1747
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 1749
Throttle_A executed at 1749
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 1771
Maneuver_T executed at 1771
Event queue (Dynamics_A Dynamics_T)
Dynamics_T executed at 1951
Dynamics_A executed at 1951
Event queue (Main_T Main_A)
Main_A executed at 1986
Main_T executed at 1986
Event queue (Throttle_A Sit_A Sit_T Throttle_T)
Sit_T executed at 2000
Sit_A executed at 2000
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 2002
Throttle_A executed at 2002
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 2024
Maneuver_T executed at 2024
Event queue (Dynamics_A Dynamics_T)
Dynamics_T executed at 2204
Dynamics_A executed at 2204
Event queue (Main_T Main_A)
Main_A executed at 2239
Main_T executed at 2239
Event queue (Throttle_A Sit_A Sit_T Throttle_T)
Sit_T executed at 2253
Sit_A executed at 2253
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 2255
Throttle_A executed at 2255
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 2277
Maneuver_T executed at 2277
Event queue (Dynamics_A Dynamics_T)
Dynamics_T executed at 2457
Dynamics_A executed at 2457
Event queue (Main_T Main_A)
Main_A executed at 2492
Main_T executed at 2492
Event queue (Throttle_A Sit_A Sit_T Throttle_T)
Sit_T executed at 2506
Sit_A executed at 2506
Event queue (Throttle_A Throttle_T)
Throttle_T executed at 2508
Throttle_A executed at 2508
Event queue (Maneuver_T Maneuver_A)
Maneuver_A executed at 2530
Maneuver_T executed at 2530

Index

Agora 23
Bisiani 23
blackboard controller 28
blackboard data objects 6
blackboard handlers 81, 133
blackboard problem-solving model 14, 131
Blackboard systems 2
centralized blackboard controller 7
centralized controller 20
Concurrency 17
Concurrency Control 16
concurrent blackboard systems 16, 19, 131
Concurrent-CLAWS 17
Conflict resolution 14
Corkill 17, 21, 27
daemons 7, 80, 132
data–driven control structure 14
distributed blackboard systems 19
execution time
    knowledge source 90
Functionally Accurate, Cooperative 28
heterogeneous computer network 81, 133
heterogeneous network 16
inference engine 14
Jagannathan 30
knowledge source
    connectivity graph 5
    interdependence 5
Knowledge source connectivity 52, 90, 131
Knowledge Source Connectivity Analysis 5, 28, 53, 69
knowledge source output/input connectivity 5
Knowledge sources 14, 27
Lesser 28
message-passing 17, 30
Mode Control Panel 85
n-readers/one-writer protocol 81, 133
Newell 13
Nii 13
object–oriented 81, 133
Operating Systems 16
opportunistic problem solving 13
Paladin 8, 111
parallel blackboard systems 19
Penny Nii 15
Selfridge's Pandemonium 13
Serial blackboard systems 15
shared memory parallel processor 81, 133

simulation model
    blackboard 6
two-phase locking protocol 81, 133
Virtual Blackboard 30

Bibliography

Alan, Robert. "Design Criteria for Expert Systems" University of Houston - University Park, Houston Texas. 1987.

Almasi, George S. and Allan Gottlieb. Highly Parallel Computing. The Benjamin/Cummings Publishing Company, Inc. 1989.

Andriole, Stephen J., ed. Applications in Artificial Intelligence Princeton N.J., Petrocelli Books, 1985.

Astrom, K.J. and J.J. Anton. "Expert Control." In Proceedings IFAC 9th World Congress, 1984. pp. 2597 - 2585.

Barr, Avron and Edward A. Feigenbaum, eds. The Handbook of Artificial Intelligence. 2. vols. Los Altos, William Kaufmann, Inc. 1981

Baumert, Johnn Anna Critchfield, and Karen Leavitt. "The Need For A Comprehensive Expert System Development Methodology." In Proceedings 1988 Goddard Conference on Space Applications of Artificial Intelligence. May 24, 1988. pp. 449 - 460.

Belkin, Brenda L., and Robert F. Stengel, "Cooperative Rule-Based Systems for Aircraft Control." Presented at the 26th IEEE Conference on Decision and Control, Los Angeles, December 1987.

Bisiani, R. "A Software and Hardware Environment for Developing AI Applications on Parallel Processors.", In Proceedings AAAI-86, pp.742-747.

Bisiani, Roberto and A Forin. "Parallelization of Blackboard Architectures and the Agora System.", in Blackboard Architectures and Applications. ed. V. Jagannathan et al. , Academic Press, Inc. 1989.

Borchardt, Gart C. "STAR: A Computer Language for Hybrid AI Applications". In Coupling Symbolic and Numeric Computing in Expert Systems. ed. Janusz S. Kowalik. Amsterdam: New Holland, 1986.

Broadwell, M.M. and D. M. Smith. "Interfacing Symbolic Processes to a Flight Simulator. In Proceedings of the 1986 Summer Computer Simulation Conference. pp. 751 - 755.

Brownston, Lee, et al. Programming Expert Systems in OPS5 Addison-Wesley Publishing Company, Inc. 1985.

Buchanan, Bruce G. and Edward H.Shortliffe eds Rule-Based Expert Systems Addision-Wesley Publishing Company, 1985.

Burgin,G. H. et al.: An Adaptive Maneuvering Logic Computer Program for the Simulation of One-on-One Air-to-Air Combat. Vol I and Vol II. NASA CR-2582, CR-2583, 1975.

Chandy B. "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design." IEEE Expert, Fall 1986, pp. 23 - 30.

Chandrasekaran, K. M. and R. Sherman. "The Conditional-Event Approach to Distributed Simulation." Information Sciences Institute Researce Report, ISI/RR-88-262, June 1989

Chandrasekaran, B. and Karsten Schwan "Parallel Real-Time Expert Systems." Progress Report on AFOSR Grant 87-0076. February 20,1988.

Charniak, Eugene, and Drew McDermott, Introduction to Artificial Intelligence. Addison-Wesley Publishing Company, 1985

Cohen, Paul R. "A Survey of the Eigth Nationbal Conference on Artificial Intelligence: Pulling Together or Pulling Apart?", AI Magazine, Spring 1991.

Corkill, Daniel D. "Advanced Architectures : Concurrency and Parallelism.", In Blackboard Architectures and Applications. ed. V. Jagannathan et al. , Academic Press, Inc. 1989.

Corkill. Daniel D. "Design Alternatives for Parallel and Distributed Blackboard Systems." Presented at AAAI-88 Workshop on Blackboard Systems, St . Paul Minnesota.

Corkill. Daniel D., Kevin Q. Gallagher, and Philip M. Johnson. "Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures." In Proceedings AAAI-87. pp. 18-23. Morgan Kaufmann Publishers.

Corkill. Daniel D., Gallagher, K. Q., and Murrary, K. E. "GBB: a Generic Blackboard Development System." In Proceedings AAAI-86, pp. 1008 - 1014. Morgan Kaufmann Publishers.

Culbert, Chris, Gary Riley, and Robert T. Savely. *"Approaches To The Verification Of Rule-Based Expert Systems."* Presented At: SOAR '87, First Annual Workshop on Space Operations Automation and Robotics, August 5 - 7, 1987.

Culbert, Chris, Gary Riley, and Robert T. Savely. "An Expert System Development Methodology Which Supports Verification and Validation." NASA/Johnson Space Center, Artificial Intelligence Section -FM72.

Dally, William J. "Concurrent Computer Architecture" In The Proceedings of The Winter Meeting of the ASME, December 13 - 18, 1983. pp. 3 - 29.

Decker, Keith. "Blackboard Systems." IEEE Expert, October 1991, pp.71-72.

Denning, Peter J. "Towards A Science of Expert Systems." IEEE Expert. Summer 1986, pp.80 - 83.

Dodhiawala, Rajendra, et al. "The First Workshop on Blackboard Systems." AI Magazine, Spring 1989. pp 77-80.

Duke, Eugene, L. et al. "An Engineering Approach to the use of Expert Systems Technology in Avionics Applications." NASA Technical Memorandum 88263, 1986.

Engelmore, R.S., and A.J. Morgan, ed. BlackBoard Systems. Addison Wesley, 1988.

Ensor. J. Robert, and John D. Gabbe, "Transactional Blackboards" In Proceedings of the 1985 International Conference on Artificial Intelligence. pp. 340-344.

Erickson, William K. "The Blackboard Model: A Framework for Integrating Multiple Cooperating Expert Systems." Code RI: Information Sciences, NASA/Ames Research Center, 1985.

Erman, Lee D., Hayes-Roth, Frederick, Lessor, Victor, Reddy D. Raj. "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty" ACM Computing Surveys. Vol 12, No. 2, June 1980. pp. 213 - 253.

Fennell, Richard d., and Victor R. Lesser. "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay II." IEEE Transactions on Computers. C-26 (2):98-111,1977.

Fikes, Richard, and Tom Kehler. "The Role of Frame-Based Representation in Reasoning.", Communications of The ACM, September 1985, vol 28, no. 9, pp.904 - 920.

Fujimoto, Richard M. "Parallel Discrete Event Simulation.", Communications of The ACM, October 1990, vol 33, no. 10, pp.30-53.

Garvey, Thomas D. "A Survey of AI Approaches to the Integration of Information." SPIE Vol 782 Infrared Sensors and Sensor Fusion (1987). pp. 68 - 82.

Glickstein, Ira, and Peter Stiles. "Application Of AI Technology to Time-Critical Functions." In Proceedings AIAA/IEEE Digital Systems Avionics Conference, October 17-20, 1988.

Goguen, Joseph. "Fuzzy Sets and the Social Nature of Truth." In Advances in Fuzzy Set Theory and Applications. ed Madan M. Gupta, Amsterdan; New York : North Holland, 1979

Goodrich, Kenneth H. and John W. McManus. "Development of A Tactical Guidance Research and Evaluation System (TGRES)." In Proceedings AIAA Flight Simulation Technologies Conference, August 14-16,1989. AIAA Paper # 89-3312, pp. 350 - 356.

Goodrich, Kenneth H. and John W. McManus. "An Integrated Environment For Tactical Guidance Research and Evaluation" In Proceedings AIAA 5th Bi-Annual Flight Test Conference, May 1990. AIAA Paper # 90-1287

Gustafson, John L. "Reevaluating Amhdal's Law." Communications of the ACM, May 1988 Volume 31 Number 15. pp. 532 - 533

Handelman, David A. and Robert F. Stengel. "Combining Expert Systems and Analytical Redundancy Concepts for Fault-Tolerant Flight Control." AIAA Journal of Guidance, Vol. 12, NO. 1, Jan .- Feb. 1989. pp. 39 - 45.

Handelman, David A. and Robert F. Stengel. "Perspectives on the Use of Rule-Based Control." Presented at the IFAC Workshop on Artificial Intelligence in Real-Time Control, Swansea, UK, September 1988.

Hayes-Roth, Barbara. "A Blackboard Architecture for Control." Artificial Intelligence, **26** (1985). pp. 251 - 321.

Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S., and Cammarata, S. "Modeling Planning as an Incremental, Opportunistic Process." In Proceedings IJCAI-79, pp. 375 - 383. Morgan Kaufmann Publishers.

Herlihy, Maurice. "A Methodology for Implementing Highly Concurrent Data Structures." ACM 089791-350-7/90/0003/0197. pp 197 - 206.

Hu, David. C/C++For Expert Systems. Management Information Source, INC Portland, 1989.

Hueschen Richard M. and John W. McManus. "Application of AI Methods to Aircraft Guidance and Control." In Proceedings 1988 American Control Conference, June 15-17, 1988. pp. 195 - 201.

Jagannathan, V., Rajendra Dodhiawala, and Lawrence S. Baum, ed. Blackboard Architectures and Applications. Academic Press, Inc. 1989.

Jagannathan, Vasudeuau. Realizing the Concurrent Blackboard Model. Technical Report BCS-G2010-61, Boeing Advanced Technology Center, P.O. Box 24346, M.S. 7L-64, Seattle Washington.

Jones, J., Millington, M., and Ross P. "A Blackboard Shell in PROLOG." In Proceedings ECAI-86, pp. 428 - 436.

Lake, Ron T. "An Object Oriented Approach to Real Time Systems Development." In Proceedings of the 33rd International Instrumentation Symposium, May 3 - 8, 1987. pp. 453 - 468.

Lakin, W. L., Miles, J. A., and Byrne, C. D. "Intelligent Data Fusion for Naval Command and Control". In BlackBoard Systems. ed. R.S.Engelmore and A.J. Morgan. Addison Wesley, 1988.

Lesser, V. R., and Corkill, D. D. "Functionally Accurate, Cooperative Distributed Systems". IEEE Transactions on Systems, Man, and Cybernetics, Vol SMC-11, No. 1, January 1981. pp 81-95

Lesser, V. R., and Corkill, D. D. The Distributed Vehicle Monitoring Testbed: a Tool for Investigating Distributed Problem Solving Networks. AI Magazine, 7 (3), pp. 82 - 106.

Lesser, Victor R., Jasmina Pavlin, and Edmund H. Durfee. "Approximate Processing in Real-Time Problem Solving." AI Magazine, **9**, no. 1 (Spring, 1988).

Li, Kai and Paul Hudak. "Memory Coherence in Shared Virtual Memory Systems. In Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing. 1986 pp. 229-239

Liebowitz, Burt H., and John H. Carson. Multiple Processor Systems For Real-Time Applications. Englewood Cliffs, New Jersey : Prentice Hall, 1985.

Lind, Henrik and Richard Stenerson. "An Avionics Expert System For Ground Threat Assessment." In NAECON 87, Proceedings of the IEEE National Aerospace and Electronics Conference, May 18-22, 1987. Volume 2. pp. 326 - 331.

Lindsay, Ken J. "Frame-Based Knowledge Representation for Process Planning." In Proceedings 1st Annual Aerospace Applications of Artificial Intelligence Conference. 1985. pp. 127 - 135.

Lucas, P. J. F. Knowledge Representation and Inference in Rule-Based Systems, Report CS-R8613 Centre for Mathematics and Computer Science, Amsterdam, The Netherlands.

McManus, John W. and Kenneth H. Goodrich. "Application of Artificial Intelligence (AI) Programming Techniques to Tactical Guidance For Fighter Aircraft." In Proceedings AIAA Guidance, Navigation, and Control Conference, 1989. AIAA Paper # 89-3525, pp. 851-858.

McManus, John W. and Kenneth H. Goodrich. " Artificial Intelligence (AI) Based Tactical Guidance For Fighter Aircraft." In Proceedings AIAA Guidance, Navigation, and Control Conference, August 20 - 22, 1990. AIAA Paper # 90-3435.

McManus, John W. "A Parallel Distributed System for Aircraft Tactical Decision Generation." In Proceedings 9th AIAA/IEEE Digital Avionics Systems Conference, October 15 - 18, 1990.

McManus, John W. "Design and Analysis Tools for Concurrent Blackboard Systems." In Proceedings 10th AIAA/IEEE Digital Avionics Systems Conference, October 14 - 17, 1991.

McManus, John W, Douglas Arbuckle, and Alan Chappell. "Situation Assessment in the Paladin Tactical Decision Generation System" In Proceedings NATO-AGARD Guidance and Control Panel 53rd Symposium, October 22–25, 1991.

Meyer, Bertrand. Object-oriented Software Construction. Prentice Hall International (UK) Ltd., 1988.

Nagao, M., Matsuyama, T., and Mori, H. "Structural Analysis of Complex Aerial Photographs." In Proceedings IJCAI-79, pp.610 - 616. Morgan Kaufmann Publishers.

Newell, Allen., "Some Problems of Basic Organization in Problem-Solving Programs". In Conference on Self-Organizing Systems, ed. M.C. Youvits, G. T. Jacobi, and G. D. Goldstein. pp. 393-423. Spartan Books, Washington D.C., 1962."

Nicol, David M., "The Automated Partitioning of Simulations for Parallel Execution". Doctoral Dissertation, University of Virginia, 1985.

Nii, H. Penny. "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures." AI Magazine, 7 (2), 1986, pp. 38 - 53.

Nii, H. Penny. "Blackboard Systems (Part 2)" AI Magazine, 7 (3), 1986, pp. 15 - 33.

Nii, H. Penny, and Aiello, N. "AGE (Attempt to GEneralize): A Knowledge -Based Program for Building Knowledge-Based Programs." In Proceedings IJCAI-79, pp.645 - 655. Morgan Kaufmann Publishers.

Nii, H. Penny, Aiello, Nelleke, and Rice, James: "Frameworks for Concurrent Problem Solving: A Report on Cage and Poligon." Presented at AAAI-88 Workshop on Blackboard Systems, St Paul Minn. In Blackboard Systems, R. S. Englemore & A. J. Morgan ed; 1988. pp. 475 - 503.

Nii, H. Penny, Feigenbaum, E. A., Anton J. J., and Rockmore, A. J. "Signal-to-Symbol Transformation: HASP/SIAP Case Study." AI Magazine 3, pp. 23 - 25.

Nijholt, A. "Topics in Artificial Intelligence." Memorandum INF-85-9 Twente University of Technology, Department of Informatics, The Netherlands. June 1985

O' Hagan, Michael. "Aggregating Template or Rule Antecedents in Real-Time Expert Systems With Fuzzy Set Logic." ORINCON Corporation, San Diego, CA.

Pang, Grantham K.H. "A Blackboard Control Architecture For Real-Time Control", In Proceedings 1988 American Control Conference, Vol 1, 88CH2601-3, pp. 221 - 226.

Pearson, G. Mission Planning within the Framework of the Blackboard Model. 1985 IEEE. Reprinted , with permission, from Expert Systems in Government Symposium. ed by Kamal N. Karna.

Raulefs, Peter. "Toward a Blackboard Architecture for Real-time Interactions with Dynamic Systems.", in *Blackboard Architectures and Applications*. ed. V. Jagannathan et al. , Academic Press, Inc. 1989.

Reynolds, D., MUSE: a Toolkit for Embedded, Real-Time AI. In *BlackBoard Systems*. ed. R.S.Engelmore and A.J. Morgan. Addison Wesley, 1988.

Rice, James, Nelleke Aiello, and H. Penny Nii. "See How They Run... The Architecture and Performance of Two Concurrent Blackboard Systems.", in Blackboard Architectures and Applications. ed. V. Jagannathan et al. , Academic Press, Inc. 1989.

Schmucker, Kurt J. Fuzzy Sets, Natural Language Computations, and Risk Analysis. Computer Science Press, 1984.

Selfridge, Oliver. "Pandamonium : A Paradigm for Learning" Proceedings of the Symposium on the Mechanization of Thought Processes., pp. 511-529, 1959.

Simon, Herbert A. "Scientific Discovery and the Psychology of Problem Solving" In Models of Discovery, D. Reidel Publishing Company, Boston Mass, 1977.

Stefik, Mark et .al. "The Organization of Expert Systems: A Prescriptive Tutorial." Xerox, Palo Alto Research Centers, January 1982.

Sztipanuvits, J. et al. "Programming Model for Distributed Intelligent Systems", In Proceedings Second Conference On Artificial Intelligence for Space Applications, pp. 363 - 371.

Tanimoto, Steven L. The Elements of Artificial Intelligence, An Introduction using LISP Computer Science Press, Inc. Rockville, 1987.

Tarjan, Robert. "Depth–First Search and Linear Graph Algorithms." SIAM Journal of Computing. Volume 1, Number 2, June 1972.

Tarjan, Robert. "Enumeration of the Elementry Circuts of a Directed Graph." SIAM Journal of Computing. Volume 2, Number 3, September 1973.

Tello, Ernest, R. Object Oriented Programming for Artificial Intelligence Addison-Wesley Publishing Company, Inc. 1989.

Terry, A. "Using Explicit Strategic Knowledge to Control Expert Systems.", In BlackBoard Systems. ed. R.S.Engelmore and A.J. Morgan. Addison Wesley, 1988.

Thorpe, Charles, Anthony Stentz, and Steven Shafer. " An Architecture for Autonomous Vehicle Navigation." In Proceedings Computers in Aerospace Conferences V, October 21-23, 1985. pp. 22 - 27.

Uczekaj, Stephen A. et al. "Autonomous Processing using a Blackboard Chipset." In Proceedings AIAA Computers In Aerospace VI Conference, October 7-9, 1987. pp. 385 - 362.

Vemuri, V. "Simulation of a distributed processing system: A case study" Simulation, May, 1991, pp. 302 - 315.

Williams, M. A., "Hierarchical Multi-expert Signal Understanding": Tech Report ESL-IR201. ESL, Inc.

Zanconato, R. "BLOBS - an Object Oriented Blackboard System Framework for Reasoning in Time", In BlackBoard Systems. ed. R.S.Engelmore and A.J. Morgan. Addison Wesley, 1988.

Zadeh, L.A. "Fuzzy Sets" Information and Control, Vol 8, 1965. pp 338 - 353.

VITA

John William McManus

Born in Stuttgart, [redacted]. Graduated from Groveton High School in Alexandria, Virginia in June 1979. B.A. Randolph-Macon College, 1984. M.S Computer Science, College of William and Mary, 1985 - 86.

The author is currently employed at the Aircraft Guidance and Control Branch of the NASA Langley Research Center in Hampton, Virginia.