NASA/TM-1999-209128

Automated Concurrent Blackboard System Generation in C++

J. A. Kaplan and J. W. McManus
Langley Research Center, Hampton, Virginia

W. L. Bynum
The College of William and Mary, Williamsburg, Virginia

April 1999

NASA/TM-1999-209128

Automated Concurrent Blackboard System Generation in C++

J. A. Kaplan and J. W. McManus
Langley Research Center, Hampton, Virginia

W. L. Bynum
The College of William and Mary, Williamsburg, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

April 1999

Available from:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

Abstract

A Blackboard System is composed of an area of shared memory, referred to as the blackboard, that contains a problem to be solved and a number of different processes, referred to as knowledge sources, that can access and modify the blackboard. Each knowledge source will post a partial solution whenever doing so can contribute to the overall solution of the problem. These partial solutions cause other knowledge sources to update their portions of the solution on the blackboard until eventually an answer is found.

In his 1992 Ph.D. thesis, "Design and Analysis Techniques for Concurrent Blackboard Systems", John McManus defined several performance metrics for concurrent blackboard systems and developed a suite of tools for creating and analyzing such systems. These tools allow a user to analyze a concurrent blackboard system design and predict the performance of the system before any code is written. The design can be modified until simulated performance is satisfactory. Then, the code generator can be invoked to generate automatically all of the code required for the concurrent blackboard system except for the code implementing the functionality of each knowledge source. His tool suite was hosted on a Symbolics LISP Workstation.

The port of the McManus tool suite to the X-Window system on a UNIX® platform is still ongoing. We have completed the port of the source code generator and a simulator for a concurrent blackboard system. The tools take as input the blackboard specification file conforming to the methodology developed by Dr. McManus. The source code generator generates the necessary C++ source code to implement the concurrent blackboard system using Parallel Virtual Machine (PVM) running on a heterogeneous network of UNIX® workstations. The concurrent blackboard simulator uses the blackboard specification file to predict the performance of the concurrent blackboard design. As with Dr. McManus' original tool suite, the only part of the source code for the concurrent blackboard system that the user must supply is the code implementing the functionality of the knowledge sources.

Table of Contents

Introduction

Definition of a Blackboard System

A blackboard system is composed of an area of shared memory, referred to as the blackboard, that contains a problem to be solved and a number of different processes, referred to as knowledge sources, that can access and modify the blackboard. The first reference to the term blackboard was in 1962 by Allen Newell when he wrote:

> 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 judge when he has something worthwhile to add to it. This conception is just that of Selridge's Pandemonium (Selfridge, 1959): a set of demons, each independently looking at the total situation and shrieking in proportion to what they see that fits their natures. (Newell, 1962)

Since that time, blackboard systems have evolved and have been used in multiple applications, including the Hearsay-II speech recognition (Erman, 1980), signal processing with the HASP/SIAP systems (Nii, 1982), errand-planning tasks (Hayes-Roth, 1979), and air combat decision generation in the Paladin System (McManus 1989).

All communication between knowledge sources takes place via the blackboard. The blackboard also contains the partial solution to the problem that the blackboard system is attempting to solve. The blackboard is decomposed into structured units known as blackboard data objects. These blackboard data objects are used as both inputs and outputs for the knowledge sources.

The knowledge source contains the knowledge that can help to solve the problem present on the blackboard. Each knowledge source will post a partial solution whenever doing so can contribute to the overall solution of the problem. These partial solutions cause other knowledge sources to update their portions of the solution on the blackboard until eventually an answer is found.

Description of McManus Tool Suite

In his 1992 Ph.D. thesis, "Design and Analysis Techniques for Concurrent Blackboard Systems", John McManus defined several performance metrics for concurrent blackboard systems and developed a suite of tools for creating and analyzing such systems. These tools allow a user to analyze a concurrent blackboard system design and predict the performance of the system before any code is written. The design can be modified until simulated performance is satisfactory. The code generator can then be invoked to automatically generate all of the code required for the concurrent blackboard system except for the code implementing the functionality of each knowledge source. His tool suite was hosted on a Symbolics™ LISP Workstation.

Description of the COBS Architecture

The Concurrent Object-Oriented Blackboard System (COBS) consists of an object-oriented blackboard data structure, a set of knowledge sources, and a set of knowledge source handlers (McManus, 1992). The blackboard holds the user defined data objects. Each data object contains a type, a value, and a list of knowledge source handlers to notify when the data object is updated. Activation of a knowledge source cannot occur until each of its input conditionals and each of its preconditions hold.

Each data object that the knowledge source uses as input has an *input conditional* that is true only if the data object has been updated since the last knowledge source activation. A *precondition* is a Boolean C expression relating the states of various data objects. When all input conditionals and preconditions of a knowledge source hold, the knowledge source is activated. The knowledge source reads the values of its inputs from the blackboard and resets the input conditionals to false. The knowledge source processes these inputs and performs its calculations.

The *postcondition* of the knowledge source is a Boolean C expression that must hold for the knowledge source to post the results of its computation to the blackboard. If the postcondition of the knowledge source is false, then the results of the computation are discarded.

The knowledge source handlers are responsible for knowledge source activation and remove the need for a centralized blackboard control module. The removal of the centralized blackboard control module allows for concurrent knowledge source execution. The knowledge source handlers use a two-phase locking protocol to maintain consistency on the blackboard, allowing n-reader/one-writer access to the blackboard data objects. Locating the data access functionality within the knowledge source handler removes the need to lock any of the data objects or regions of the blackboard during knowledge source execution. This allows the knowledge source handler to continue evaluating input conditionals and preconditions while the knowledge source executes.

Each knowledge source is a highly specialized, independent process consisting of inputs, outputs, input conditionals, preconditions, postconditions, and execution logic. The handler associated with the knowledge source provides blackboard input and output capabilities along with evaluation of the input conditions, preconditions, and postconditions. The user provides the execution logic for the knowledge source. The handler will activate its knowledge source when the input conditionals and the preconditions of the knowledge source evaluate to true (see Figure 1). The handler will make a copy of the input data on the blackboard and send it to the knowledge source. The handler will then reset the input conditionals to false. After receiving this message, the knowledge source will execute its logic. When the knowledge source is done, the output will be sent back to the handler. The handler will test the postconditions. If the postconditions evaluate to true, the handler then updates the output data objects with the results from the knowledge source. If the postconditions are false, the output will be discarded.

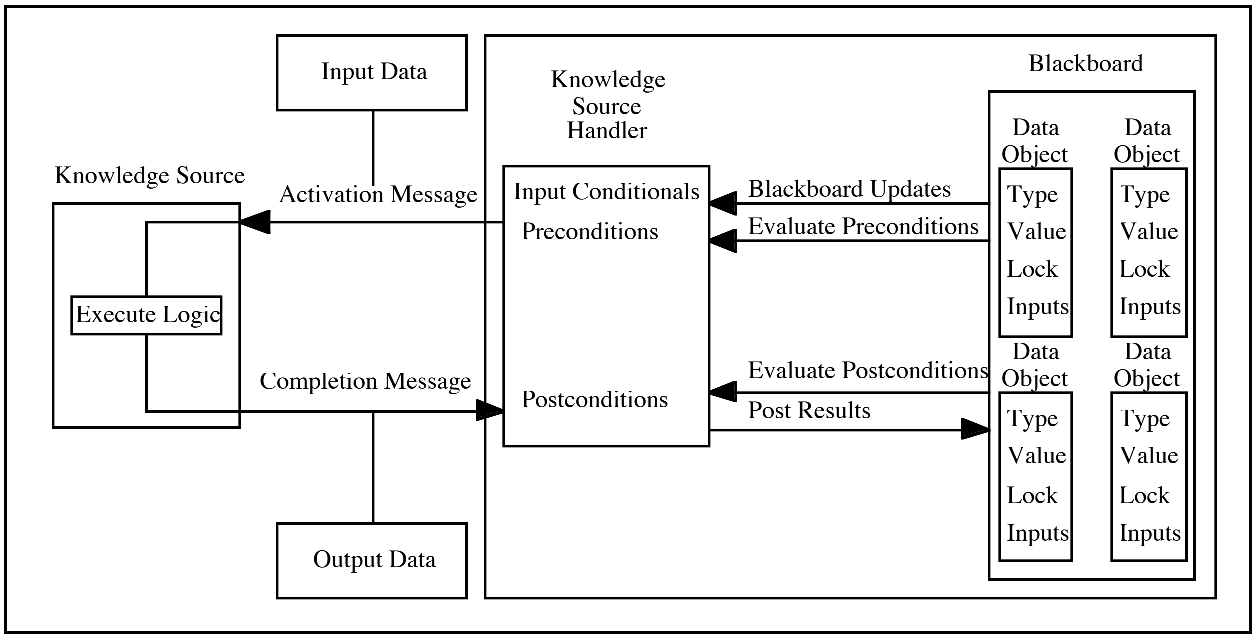


Figure 1 – Activation Cycle for Knowledge Source

Tools

Knowledge Source Connectivity Analyzer

The Knowledge Source Connectivity Analyzer analyzes a proposed blackboard system and outputs a list of metrics detailing the system, including *Knowledge Source Specialization, Knowledge Source Interdependence*, and *Knowledge Source Independence. Knowledge Source Specialization* gives the level of overlap between the task that two separate knowledge sources solve. *Knowledge Source Interdependence* details the functional connectivity between two separate knowledge source. *Knowledge Source Independence* describes the serialization of two knowledge sources and their ability to run in parallel. This system can be used to build a blackboard specification file that is then used by the simulation system and the code generator.

The port of the McManus tool suite to the X-Window system on a UNIX® platform is still ongoing. The underlying code is written in the C and C++ programming languages while the user interface is written using the Tcl/Tk graphical interface library. This tool allows a user to generate and analyze a concurrent blackboard system without having to build the system.

COBS Blackboard System Simulation System

The Concurrent Object-Oriented Blackboard System (COBS) includes a system simulator. This simulator can be used to model a proposed blackboard system before the system is built. The simulator is a discrete event simulation of a fully functional blackboard system. A COBS specification file created by the Knowledge Source Connectivity Analyzer specifies the structure of the system. The simulator runs the blackboard system with the exception of the functionality of the knowledge source. The COBS Blackboard System Simulator is implemented in Common Lisp on a Symbolics™ workstation.

The COBS Code Generator

The COBS Blackboard System Code Generator generates all necessary support software, not including the code for the knowledge sources, for a concurrent blackboard system as specified by a COBS specification file created by the Knowledge Source Connectivity Analyzer. This software includes all required variables, blackboard data objects, blackboard locking mechanisms, knowledge source handlers and their associated functionality to handle input conditionals, preconditions, and postconditions. The user must provide the actual source code for the knowledge sources, although the code generator does provide a generic interface for the user-provided knowledge source code to communicate with the knowledge source handlers. The COBS Blackboard System Code Generator produces code in Common Lisp using the Common Lisp Object System and the Common Lisp Interface Manager and is portable to any computer that supports the Common Lisp Standard.

Problems of currently existing solutions

There are currently several existing systems that auto-generate code for use in blackboard systems (Ho, 1991;Ho, 1994; Hewett, 1993; and McManus, 1992) that create a serial implementation of a blackboard system. These serial implementations do not exploit the parallelism inherent in the blackboard paradigm. These systems were validated using a simulation of a parallel system on a serial processor.

McManus's COBS system is designed to be executed on "a centralized shared memory parallel processor hosting the blackboard data structure", but was not available for use. (McManus, 1992) The system was instead tested on a Symbolics™ Workstation by multi-tasking the knowledge source handlers. Simulations were used to predict the behavior of the system in an ideal configuration. The inherent parallelism of the separate knowledge sources was never exploited.

The possible processing power available through the network has expanded with the proliferation of fast, cheap networks. Continuing to implement a serial version of a parallel system is no longer necessary. Dai lists the two main benefits of parallelism (Dai, 1993). First, by operating the knowledge sources in parallel, they can share the workload more evenly and significantly improve the system performance. Second, a real-time task can be sub-divided into several tasks to guarantee response time.

Description of new tool - Blackboard Generator (BBG++)

High-Level Description

The Blackboard Generator in C++ (BBG++) is an auto-code generator for a C++ concurrent blackboard system. The generated code includes all necessary source code for the blackboard system, with the exception of the knowledge source functionality provided by the user. The source code handles all necessary communication, construction of the blackboard, initialization of the blackboard data objects, locking semantics for the blackboard data objects, an opportunistic activation scheme for the knowledge source handlers, and a Makefile to compile the system in the UNIX® environment.

Automated Generation

The file describing the system must be in the format used by the McManus tool suite (see the related section below for a detailed description). The auto-code generator has the ability to output either a fully functioning system, or a simulation of the system. The source code produced by the BBG++ allows for distributed execution of the knowledge sources. The blackboard and the knowledge source handlers must exist on the same machine. This is necessary because UNIX® semaphores are used as the locking semantics for the blackboard data objects. These semaphores are implemented in the kernel of the UNIX® operating system (Stevens, 1992). The execution of these knowledge source handlers may be parallelized, however, by executing them on a multi-processor shared memory machine where the memory for the machine can be mapped into each process's address space. Examples of such machines include a Silicon Graphics, Incorporated Onyx or a dual-processor Intel clone. Both of these systems can run variants of the UNIX® operating system.

The system produced from the auto-code generator must be compiled into executable form. This compilation will occur after the user has specified the functionality of the knowledge sources. After compilation, the system can be executed.

The main blackboard host process begins execution and reads the specification file. The blackboard is created and sized. Semaphores are allocated for each blackboard data object. The knowledge source handlers are created and initialized. This initialization includes the creation of the blackboard data, along with the necessary information to access the semaphores associated with the blackboard data objects.

The simulation of the concurrent blackboard system models the activation of the knowledge sources using a discrete event simulation. All the necessary communication and activation of knowledge sources is performed. The functionality of the knowledge sources need not be implemented in the simulation, since their execution times will be estimated. In determining the performance of the system, the simulation uses only the estimated execution times of the knowledge sources. The time necessary to lock and unlock the blackboard data objects and communication time between nodes is not considered in computing execution time. This time is assumed to have been included in the execution time of the knowledge sources. Simulations of the system can vary dramatically from running the functional system if network loading changes dramatically between simulation and actual execution.

Distributed Execution of Knowledge Sources on a Heterogeneous Network of Workstations

The knowledge sources for the system are spawned to different machines across a heterogeneous computing environment using Parallel Virtual Machine (PVM) (Geist, 1994). The knowledge source handler, executing on the blackboard host computer, sends an activation message to the knowledge source. Multiple knowledge sources can be spawned on the same machine if the user anticipates that they will not activate at the same time. Specialized sensor knowledge sources can be spawned onto machines that are directly attached to the interfacing specialized hardware. Some or all of the knowledge sources can be executed on the blackboard host computer, if desired. The actual layout of the knowledge sources onto individual processors is best left to the user, since the user has first hand knowledge of the problem domain. In Figure 2, the user has placed two knowledge sources on the same machine while separating the other two onto their own individual hosts.

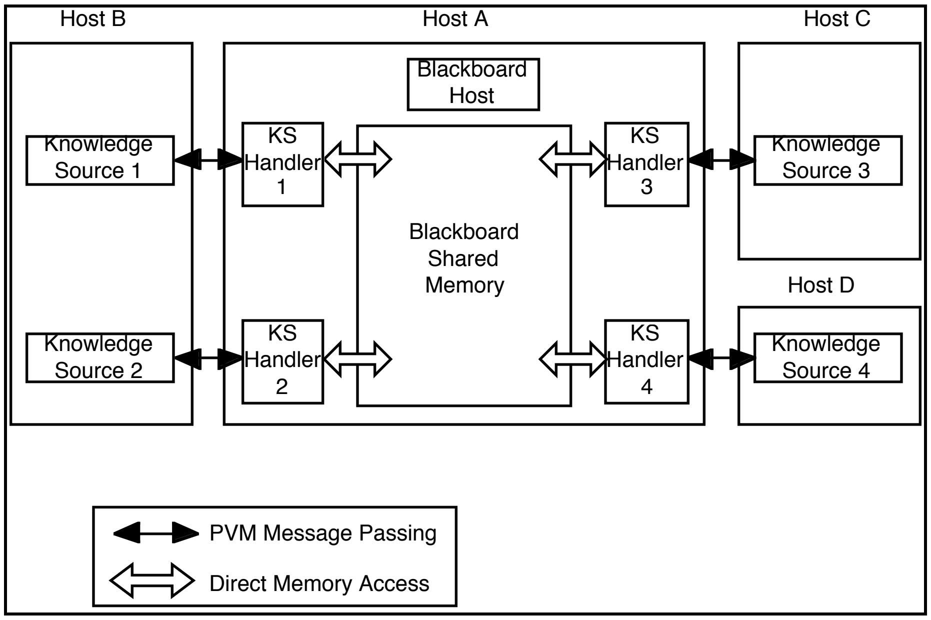


Figure 2 - Simple Distributed Blackboard System

Distributed execution of the knowledge sources allows each knowledge source to execute as quickly as possible on the most appropriate hardware. A knowledge source that is most appropriately executed on a vector supercomputer can be executed on that platform to take advantage of the inherent specialization available on that platform. Knowledge sources that are distributed applications themselves can be further subdivided to run on parallel machines, communicating using PVM.

With the proliferation of inexpensive, powerful workstations, workstation clusters offer both a cost-effective alternative to batch processing and an easy entry into parallel computing (Kaplan, 1994). By using PVM as both a spawning and communication mechanism, the knowledge sources can be distributed across a large number of commercially available hardware platforms connected by many different high-speed networks. This gives the user maximum flexibility when configuring a blackboard system for a specific application. In Figure 3, a complicated blackboard system is shown that incorporates actuators, sensors, and a knowledge source that serves as a front end to a parallel-processing machine.

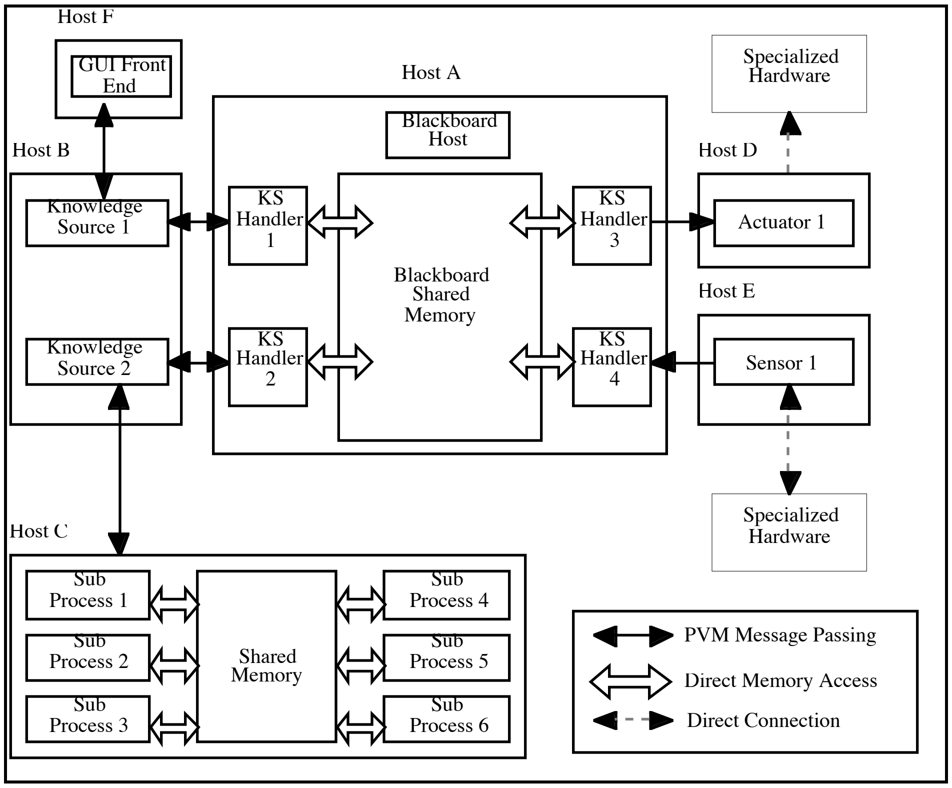


Figure 3 - Complex Blackboard System

Tool Suite

The port of the McManus Tool Suite from the Symbolics workstation in LISP to the X-Window System on UNIX® is still ongoing (see Figure 4). The tool suite currently has the ability to perform the following functionality on an existing blackboard specification file: view or edit an existing Knowledge Source (see Figure 5); view or edit an existing blackboard data object (see Figure 6); view a connectivity graph (see Figure 7); view analysis metrics; view elementary circuits in the blackboard; or write a new specification file. Work will continue on the tool suite with an emphasis on added functionality.

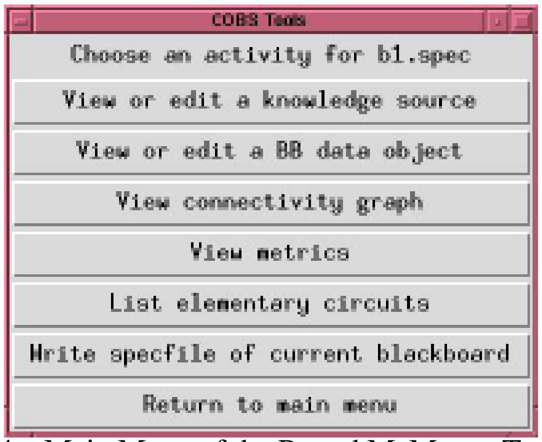
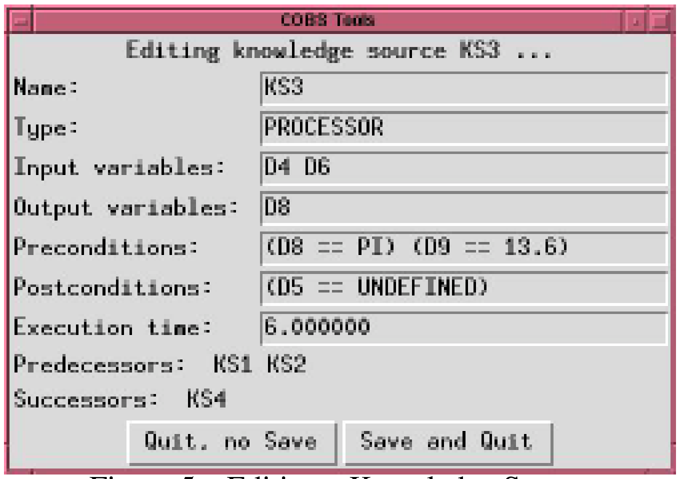

COBS Tools
Choose an activity for b1.spec
View or edit a knowledge source
View or edit a BB data object
View connectivity graph
View metrics
List elementary circuits
Write specfile of current blackboard
Return to main menu

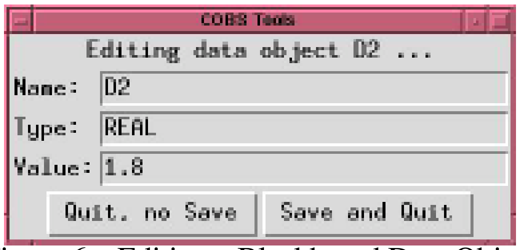
Figure 4 – Main Menu of the Ported McManus Tool Suite



Editing knowledge source KS3 ...

Name:	KS3
Type:	PROCESSOR
Input variables:	D4 D6
Output variables:	D8
Preconditions:	(D8 == PI) (D9 == 13.6)
Postconditions:	(D5 == UNDEFINED)
Execution time:	6.000000
Predecessors:	KS1 KS2
Successors:	KS4

Figure 5 – Editing a Knowledge Source



Editing data object D2 ...

Name:	D2
Type:	REAL
Value:	1.8

Figure 6 – Editing a Blackboard Data Object

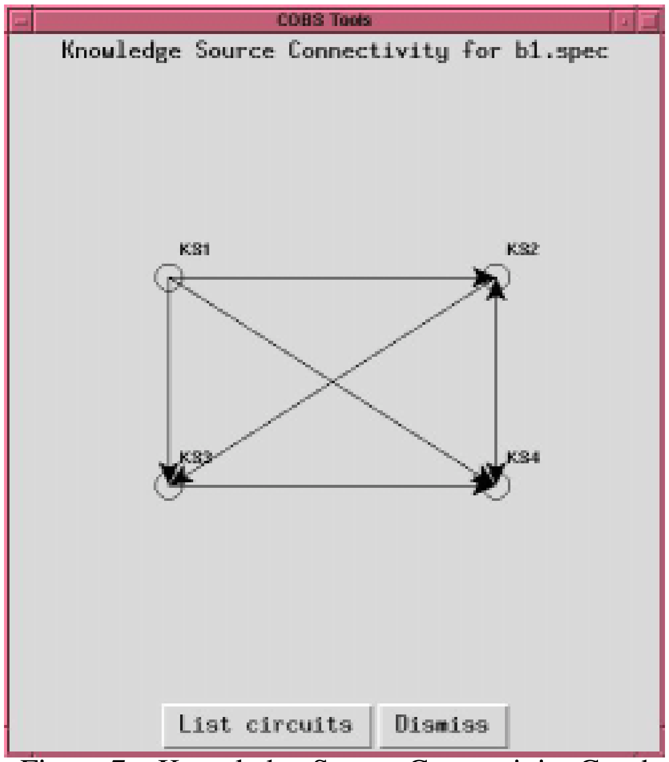


Figure 7 – Knowledge Source Connectivity Graph

Blackboard Description File

The Blackboard Specification completely describes the setup of the blackboard system (see Figure 8 for an example). The file begins with header information, included in a comment, that details the generation date and name of the specification file. The header information is followed by detailed information regarding the knowledge sources. The blackboard data objects information is given next. The final element of the specification file is a listing of processors that this blackboard can utilize during execution. This final section was added to support the additional functionality of distributed knowledge source execution.

The knowledge source section starts with the total number of knowledge sources in the system. Following this information is a detailed description of each knowledge source in order. The information about the knowledge source is given in the following order, which is interspersed with comments to improve the readability of the file: knowledge source name, type of the knowledge source, input variables, output variables, preconditions, postconditions, execution time, update interval, and processor tag. The knowledge source name can be any alphanumeric string, but must be unique with respect to other knowledge sources. The input and output variables are lists of blackboard data objects. The lists of preconditions and postconditions must be given in terms of Boolean expressions in C; usually relating the blackboard data objects. The execution time is used for the simulation facility and is measured in whatever units the user chooses. The units used should be consistent between the knowledge sources. The update interval is only meaningful if the knowledge source is a sensor and describes how often the sensor should be activated. The rest of the knowledge sources are activated when all of their input conditionals and preconditions hold. The last item of information about the knowledge source is the location of the desired processor for execution, which is a number corresponding to one of the machines listed lower in the specification file.

The blackboard data object section begins with the total number of blackboard data objects in the system. Following this information is a detailed description of each blackboard data object. The information is given in the following order, which is interspersed with comments to improve the readability of the file: blackboard data object name, type of the blackboard data object, and the initial value. The blackboard data object name can be any alphanumeric string, but must be unique with respect to other blackboard data objects. There are three types of blackboard data objects: REALS, which are floating point values, INTEGERS, and STRINGS consisting of numbers and letters. A STRING type is limited to 80 characters by default, but the default can be adjusted. If the initial value is unspecified, it should be given as UNDEFINED.

The processor section begins with the total number of machines that are available for use by the system. Following this information is a listing of the Internet address of the machines. These machines must be available through a Local Area Network and must have the Parallel Virtual Machine software correctly installed and accessible.

```
# Blackboard specification file: full_test.spec
# produced by the COBS system  Fri Apr 10 15:40:27 1998

# number of knowledge sources
1
# KS name
 S1
# type
 SENSOR
# input variables
 none
# output variables
 D1
# preconditions
 none
# postconditions
 none
# execution time
 2.000000
# update interval
 1.0
# processor tag
 0

# number of data objects
1
# DO name
 D1
# type
 REAL
# value
 1.0

# number of machines
1
#machine name
processor1.larc.nasa.gov
```

Figure 8 – A Sample Blackboard Specification File

Parser

A PERL script known as parser is used to move the user from the Blackboard Analysis tools to a functioning system. The script generates the source code for different pieces of the blackboard system: the blackboard host executable, the knowledge source handlers, the framework for the knowledge sources, and the Makefile.

The user may also generate the simulation if desired. Generating the simulation will alter the Makefile, but not affect the actual code that is generated. The source code for the simulation will either be enabled or disabled at compile time, depending upon the user's wishes.

The parser is passed the name of the blackboard configuration file as a command line argument. If the user also appends the keyword "all" on the command line, a normal executable system will be generated with the simulation non-active. If the keyword "all" is not included, the user will be asked a series of questions to determine which pieces of the system should be generated.

Once the user has entered in his or her preferences, the parser will read the specification file and obtain all necessary information. A directory will be created as a storage repository for the blackboard currently being generated. All template files for modification will be copied into this directory. These files are:

ks.cpp	One copy for each knowledge source will be copied into the directory and given the name of the knowledge source according to the specification file and appended with ".cpp"
kshandler.cpp	One copy for each knowledge source will be copied into the directory and given the name of the knowledge source according to the specification file and appended with "-Handler.cpp".
Semaphore.cpp	File for Semaphore object used for Blackboard Data Object Locking
Semaphore.hpp	Header file for Semaphore object used for Blackboard Data Object Locking
SharedMemory.cpp	File for Shared Memory Object used to access Blackboard
SharedMemory.hpp	Header file for Shared Memory Object used to access Blackboard
Ioutil.hpp	Header file for I/O object used by SharedMemory and Semaphore objects
Ioutil.cpp	File for I/O object used by Shared Memory and Semaphore objects
bb_pvm3.h	Header file with definitions of message passing identifiers
structures.h	Header file containing specifications of data structures in Shared Memory
LinkedList.cpp	File for Linked List Object used by the Blackboard Simulation System
LinkedList.hpp	Header file for Linked List Object used by the Blackboard Simulation System
Clock.cpp	File for clock object used in the Blackboard Simulation System
Clock.hpp	Header file for clock object used in the Blackboard Simulation System
Queue.cpp	File for an Event Queue object used in the Blackboard Simulation System
Queue.hpp	Header file for an Event Queue object used in the Blackboard Simulation System
Table.cpp	File for a table of Clocks objects used in the Blackboard Simulation System
Table.hpp	Header file for a table of Clocks objects used in the Blackboard Simulation System
assume.hpp	Header file used for assert statements
Templates.cpp	File used to specify all template specifications
Simulation.hpp	Header file used in the Blackboard Simulation System

The first file to be modified is bb_host.cpp. The file is responsible for bringing the entire system into existence, either the simulation or the actual running system. The only modification necessary to this file is to place the proper location of the specification file into the system. The template file has the keyword SPECFILE inside of it and this keyword is replaced with the actual location.

The next file to be modified is the Makefile. The Makefile template is sparse, having two lines for building the bb_host, two lines for the knowledge sources, and two lines for the knowledge source handlers. The bb_host lines require no modifications. The lines for the knowledge sources and the knowledge source handler from the template must be duplicated for each knowledge source in the system. The proper names must also be inserted into the Makefile. The Makefile that is eventually written will be able to build the entire system once the knowledge sources have been given functionality by the user. The Makefile will have to be edited by hand by the user if the user adds additional software to the knowledge sources that must be included at compile time.

A number of compiler variables will be set in the Makefile if the user has requested construction of a simulation. These compiler variables will cause certain code to be included in the final compiled version of the software. The blackboard simulation system runs the same code as the normal system does, but additional messages and handshaking between the knowledge source handlers and a simulation module are included. This simulation module keeps track of the execution times and manages the event queues that determine when the knowledge sources execute.

Once the blackboard host file and the Makefile have been generated, the knowledge source handlers are modified. These files are the most customized files in the system, because the knowledge source handlers have to receive messages about their data objects in the shared memory being updated. Storage space is created in the knowledge source handler for storing the values of each of the data objects. Once this has been accomplished, the preconditions are created within the knowledge source handler. The postconditions are then created in the same manner and inserted into the knowledge source handler code.

Once the files have been written, the user can add the desired functionality to the knowledge sources. This functionality can be as simple or as complex as the user needs. It can take advantage of platform specific operations, such as vector processing or graphics. The necessary header files and libraries must be included in both the knowledge source software and in the Makefile so a proper compilation can be performed. The user may also link in other programming languages, again with the stipulation that the compile is done properly.

Blackboard Host Software

The Blackboard Host Software (BHS), bb_host.cpp, brings the entire system into existence. The BHS must be started on the machine that will host the blackboard and the knowledge source handlers. It is recommended that this machine be a shared memory multiprocessor machine. It creates the shared memory used for the blackboard, allocates the semphores used to lock the blackboard data objects, spawns the knowledge sources and their handlers, and connects the knowledge sources and their handlers. The BHS deallocates the shared memory and semaphores when requested when the blackboard is shut down. The BHS is also responsible for running the simulation if the user has requested the blackboard simulation system. The BHS does not interfere with the actual running of the system once it has been started. The only message that the BHS will accept during execution is a request to shut down the system.

During the simulation, the blackboard host is responsible for queuing all events in the system, dispatching knowledge sources for execution and receiving a message when they return. As an integral part of the simulation, the BHS insures that events occur when they are supposed to. It is possible for distributed knowledge sources to get out of during the simulation without a tight control strategy.

The BHS begins by reading the specification file and storing the information obtained from the file. Some of this information is stored in global variables to be used by the signal handler that is called if the system is shutdown in an unexpected way. The signal handler is registered on several signals. The BHS enrolls in PVM and records its PVM identification number. All machines that are specified in the specification file are enrolled in the PVM machine being used for the blackboard system.

The BHS attempts to resolve any problems that might occur during the setup process. These problems include adding a machine that is already in the virtual machine, attempting to add a machine that does not have PVM installed correctly, attempting to access a machine that does not exist or a to which access is not allowed. These errors will not cause the system to stop, however. The BHS will not be able to start the knowledge sources properly on these remote machines.

After starting the virtual machine, the size of the blackboard is calculated and the blackboard is created. Semaphores are allocated for each blackboard data object. The BHS spawns the knowledge sources on the remote machines. The knowledge source handlers are also spawned on the same host as the BHS. Messages are sent to the knowledge sources informing them of the process identification numbers of their handlers. Messages are also sent to the knowledge source handlers informing them of the process identification number of their knowledge source.

The BHS builds a list of the process identification numbers of all the knowledge source handlers interested in each blackboard data object. The knowledge source handlers use this list when updating a blackboard data object. The knowledge source handler, upon updating a blackboard data object, sends a message to each knowledge source handler in the list informing the handler that the blackboard data object has been updated.

The data objects are mapped into the shared memory. Each knowledge source handler is sent a message describing its data objects and the semaphore keys that the handler will need to lock each of its data objects. A message is also sent to each knowledge source detailing the type of data objects the knowledge source will receive for inputs and outputs.

The last step in starting the system is to initialize those data objects with a defined initial value. Update messages are sent to the knowledge source handlers whose knowledge sources use the initialized data objects as inputs. The system begins to execute only when the input conditionals and preconditions of one or more knowledge sources are met. Satisfaction of input conditionals and preconditions requires that some subset of the blackboard data objects must have values, either through initialization or sensor postings to the blackboard.

The blackboard simulation system is executed if the user has requested it during the parser execution. This will cause additional code within the BHS to be executed. The knowledge source handlers and knowledge sources will still execute, but the blackboard simulation system will activate each knowledge source instead of the knowledge source handlers.

During the simulation, the actual code of the knowledge source does not have to execute. Acceptable substitutions may be used. For example, instead of having a knowledge source perform a complex computation, an approximating function can be substituted in its place to generate realistic output. These realistic outputs are necessary for knowledge source activation. Blackboard data objects will be locked and unlocked as if the real system were running. The running time of the system will be generated from the execution time of the knowledge sources given in the specification file. The actual running time of the knowledge source during execution is not used.

Knowledge Source Handler Software

The knowledge source handler software is spawned by the BHS on the blackboard host machine. After being enrolled in PVM and accepting a process identification number, the knowledge source handler retrieves the identification number of its parent, which is the BHS. The knowledge source handler waits for a message from the BHS detailing the knowledge source to be handled, the key to access the blackboard data structure in the shared memory, the number and type of blackboard data objects, and the input and output data objects of the knowledge source.

The knowledge source handler, after receiving these messages, maps the shared memory segment into its address space and looks for all of the semaphore keys for the blackboard data objects that its knowledge source will use. Once this has been accomplished, input and output buffers are allocated for the blackboard data objects that will be sent to and received from the knowledge source. The knowledge source handler then informs the BHS that it is ready for operation.

The knowledge source handler will now enter its operational loop. The handler evaluates the input conditionals of the knowledge source. If all are true, the handler will check the preconditions. If the preconditions have been met, the blackboard data objects are read from the blackboard and sent to the knowledge source with a message indicating that the knowledge source should begin execution. If the preconditions have not been met, all of the input conditionals are reset to false. After these checks are made, the handler will check to see if it has received any messages. An update message indicates that one of the input data objects of the knowledge source has been updated. A message from the knowledge source indicates that the knowledge source associated with the handler has completed its computation and is ready to post results back to the blackboard. If the postconditions are true, then the results sent from the knowledge source will be written onto the blackboard. If there are no messages, the handler will return to the top of the operational loop.

Knowledge Source Framework Software

Each knowledge source is spawned by the BHS on the host specified by the user in the blackboard specification file. The knowledge sources obtain their process numbers and the process number of the BHS from a PVM library function call. Each knowledge source then waits for a message from the BHS providing the PVM process number of the knowledge source's handler. The BHS then sends a message that gives the knowledge source its proper number of inputs and outputs. The knowledge source uses this information to set up all of its PVM input and output buffers for passing information to and from its handler.

Every knowledge source is implemented through the same code template. The code template requires modification by the user to implement the knowledge source functionality. Once a knowledge source has finished its initialization sequence, it waits for a message to arrive from its handler containing input data. The data will be unloaded from the message and placed into the input buffer. The knowledge source will then call a specific routine, userRoutine. This routine is the location for all of the user code that encompasses the logic in the knowledge source, and execution must begin here. Additional code in other modules may be used by the knowledge source, but additional modules might require new header files to be included during compilation, library files to be included during linking, and modifications to the Makefile.

Future Work

The system as it exists is a powerful tool for the development of concurrent blackboard systems. It is user friendly and flexible in allowing a user to configure a concurrent blackboard system to accomplish a desired goal.

Many additional features could still be developed to make the system more flexible. Array blackboard data objects should be simple to implement and would add a great deal of power to the system. Users interested in graphical or signal processing, in addition to many other problem domains, could use these data objects.

A specific graphical user interface to be used for the generated blackboard system could be added. Currently, XPVM can be used by the user to see the message traces and which hosts are in execution in the parallel machine. While this does give the user some insight into the system, a more intricate user interface would help. This interface could show the user the layout of the shared blackboard segment, the state of the blackboard data objects, and additional information about the states of the knowledge source handlers. The user could determine why a knowledge source is not executing by examining the graphical interface. This interface could also be used during simulation execution to help the user tune the blackboard system properly before execution.

References:

Dai, Haihong, John G. Hughes and David A. Bell. “A Distributed Real-Time Knowledge-Based System and its Implementation using Object-Oriented Techniques. IEEE Proceedings of International Conference on Intelligent and Cooperative Information Systems, 1993 pp. 23-30.

Erman, L.D., Hayes-Roth, F., Lesser, V.R., and Reddy, D.R. (1980) The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. In Computing Surveys 12, pp. 213-253, 1980.

Geist, Al, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam, “PVM: Parallel Virtual Machine”, The MIT Press, Cambridge, Massachusetts, 1994.Hayes-Roth, Barbara A Blackboard Model of Control, Heuristic Programming Project, Report No. HPP-83-38, June 1983.

Hewett, Michael and Rattikorn Hewett “A Language and Architecture for Efficient Blackboard Systems” In: Proceedings of The Ninth Conference on Artificial Intelligence for Applications, March 1-5, 1993, pp. 34-40.

Ho, Cheng-Seen. “Development of a blackboard shell with context blackboard-based control loop” IEEE Proceedings-E, Vol. 138, No. 1, January 1991.

Ho, Cheng-Seen and Chieh-Hsin Keng. “Development of a generalized knowledge source structure for blackboard systems” Knowledge Based System, Volume 7 Number 1, March 1994.

Kaplan, J.A., and Michael L. Nelson “A Comparison of Queueing, Cluster, and Distributed Computing Systems. NASA TM-109025 (Revision 1), June 1994.

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 89-3525 pp. 852-858.

McManus, John W. (1992) “Design and Analysis Techniques for Concurrent Blackboard Systems,” Doctoral Dissertation, The College of William and Mary, April 1992.

Newell, A. (1962) Some problems of the basic organization in problem-solving programs. In: Proceedings of the Second Conference on Self-Organizing Systems. Yovits, M.C., Jocobi, G.T. and Goldstien, G.D. (eds), pp. 393-423, Spartan Books.

Nii, H.P., Feigenbaum, E.A., Anton, J.J. and Rockmore, A.J. (1982) Signal-to-symbol transformation: HASP/SIAP case study. AI Magazine 3, pp. 23-25.

Selfridge, O (1959) Pandemonium: a paradigm for learning. In: Proceedings of Symposium on the Mechanization of Thought Processes, pp. 511-29, HMSO, London.

Stevens, Richard W. Advanced Programming in the UNIX® Environment, Addison-Wesley, Reading, Massachusetts, 1992.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (*Leave blank*)	**2. REPORT DATE** April 1999	**3. REPORT TYPE AND DATES COVERED** Technical Memorandum

4. TITLE AND SUBTITLE Automated Concurrent Blackboard System Generation in C++	**5. FUNDING NUMBERS** 992-30-11-01
6. AUTHOR(S) J. A. Kaplan J. W. McManus W. L. Bynum	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-17823
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** NASA/TM-1999-209128

11. SUPPLEMENTARY NOTES
William L. Bynum - The College of William and Mary, Williamsburg, VA; Joseph A. Kaplan - NASA LaRC; John W. McManus - NASA LaRC

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified-Unlimited Subject Category 61 Distribution: Standard Availability: NASA CASI (301) 621-0390	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

In his 1992 Ph.D. thesis, "Design and Analysis Techniques for Concurrent Blackboard Systems", John McManus defined several performance metrics for concurrent blackboard systems and developed a suite of tools for creating and analyzing such systems. These tools allow a user to analyze a concurrent blackboard system design and predict the performance of the system before any code is written. The design can be modified until simulated performance is satisfactory. Then, the code generator can be invoked to generate automatically all of the code required for the concurrent blackboard system except for the code implementing the functionality of each knowledge source.

We have completed the port of the source code generator and a simulator for a concurrent blackboard system. The source code generator generates the necessary C++ source code to implement the concurrent blackboard system using Parallel Virtual Machine (PVM) running on a heterogeneous network of UNIX® workstations. The concurrent blackboard simulator uses the blackboard specification file to predict the performance of the concurrent blackboard design. The only part of the source code for the concurrent blackboard system that the user must supply is the code implementing the functionality of the knowledge sources.

14. SUBJECT TERMS Blackboard, C++, Parallel Virtual Machine (PVM), Artificial Intelligence Auto-Generated Code, Distribution System	**15. NUMBER OF PAGES** 24
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL