MARBLE - a System for Executing Expert Systems in Parallel

Leonard Myers, Coe Johnson and Dean Johnson.

ICADS Research Unit
California Polytechnic State University
San Luis Obispo, California 93407

April 20, 1990

Summary.

This paper details the MARBLE 2.0 system which provides a parallel environment for cooperating expert systems. The work has been done in conjunction with the development of an intelligent computer-aided design system, ICADS, by the CAD Research Unit of the Design Institute at California Polytechnic State University[1].

MARBLE (Multiple Accessed Rete Blackboard Linked Experts) is a system of CLIPS shells that execute in parallel on a ten processor, shared-memory computer. Each shell is a fully functional CLIPS expert system tool. A copied blackboard is used for communication between the shells to establish an architecture which supports cooperating expert systems that execute in parallel.

The design of MARBLE is simple, but it provides support for a rich variety of configurations, while making it relatively easy to demonstrate the correctness of its parallel execution features. In its most elementary configuration, individual CLIPS expert systems execute on their own processors and communicate with each other through a modified blackboard. Control of the system as a whole, and specifically of writing to the blackboard is provided by one of the CLIPS expert systems, an expert control system.

Introduction.

The MARBLE project is a framework for executing simultaneous CLIPS expert systems in a tightly-coupled shared-memory parallel computer environment. Specifically, MARBLE modifies CLIPS 4.3[2] to implement a blackboard system[3,4] for control of narrowly focused expert systems that execute in parallel. The system is specifically intended to provide a platform for experimentation in the development of techniques for cooperative problem solving with multiple expert systems.

Cooperative problem solving approaches are of interest primarily for their promise to simplify the complexity of developing solutions to large ill-defined problems and because the use of multiple problem-solving agents can be mapped to parallel hardware architectures with the expectation of reducing execution time.

The Blackboard Model.

The design philosophy of MARBLE is based on the following hypotheses:

1. *Large expert systems might be better engineered in the future as groups of independently developed specialized systems.*
2. *The control of cooperating expert systems can itself*

be implemented as an expert system.

The two above ideas are quite simply based on the attempt to model a committee of experts, or a person who is advised by several experts. The expertise of the individual expert is distinct, and the manner in which the group operates is independent of the individual areas of expertise. Since committee work is common in human problem solving, it should be possible to model cooperative machine expert systems.

A chairperson, or project leader, focuses attention to specific sub-problems and maintains order. It is assumed that the meeting area provides a blackboard on which all important information is recorded. The chairperson uses the blackboard to provide a description of the current statements accepted by the team and to focus the attention of the team to the issues that must be considered to solve the problem. Team members are not permitted to communicate directly with each other. They must direct their comments to the leader, who uses the blackboard to provide the communication. Often, the blackboard is described as holding the current state of the solution to the problem and a history of its contents can be used to analyze the problem solving techniques of the team.

Although the blackboard model has been used in many projects over the last decade, the implementation of cooperation is in its infancy[5]. Therefore, it is important to develop a platform for experimentation with various approaches.

The basic architecture of MARBLE is illustrated in figure 1. The chairperson is replaced by the control expert system, and the team members are replaced by specific domain expert systems. The blackboard can be read by any expert system, but only the control expert system is permitted to modify it. In the simplest context, the control expert system examines suggestions from the other experts and summarizes the collective wisdom.

The PEBBLE Predecessor.

The MARBLE project follows the development of PEBBLE (Parallel Execution of Blackboard Linked Experts)[6]. PEBBLE is an initial attempt at executing multiple expert systems in a shared-memory parallel computer system under the blackboard model. It uses the C programming language to implement a simple expert system shell language in which the expert systems access a shared-memory blackboard. Communication between the experts is handled through action descriptors, which are small tables that protect their information from mutual access errors.

By compiling the PEBBLE expert system language and building a dependency graph from the conditions used in the production rules, efficient execution is obtained. PEBBLE also demonstrates the effectiveness of the action descriptor approach, but the limitations of its pattern matching make it inefficient to use in

the development of large expert systems.

The powerful pattern matching capability of CLIPS and the ready availability of C-language source code make it an attractive candidate for replacement of the PEBBLE language. The use of CLIPS will also permit this research to focus on the cooperation of expert systems, rather than the continued development of the language itself. Thus MARBLE is born as the PEBBLE framework with CLIPS shells replacing the PEBBLE language shells.

The guiding principal in incorporating CLIPS into a PEBBLE-like configuration of parallel processing is to make the change as transparent to as much CLIPS code as possible. This is necessary in order to reliably make changes in the C code, which is an intricate fabric of interrelated functions and data structures, and to provide a platform that will make it possible to easily update the system with expected future versions of CLIPS.

Shaping MARBLE from PEBBLE.

Since the use of a blackboard is central to the intended application in the ICADS system, the primary problem is to implement a blackboard with CLIPS expert systems. The initial approach attempted to modify the CLIPS shell so that each expert could access the blackboard as an additional fact list kept in shared memory. This is complicated by the intimate connection between the fact list and the Rete network. As the coding changes to accomplish this transition were made, it became apparent that it would be necessary to make such basic alterations to CLIPS that it would jeopardize the ability to convenient replace the modified CLIPS shells with new versions.

It is important to understand why the PEBBLE shells cannot be directly replaced by CLIPS shells. In PEBBLE the facts are organized in a hashed table, similar to that of a symbol table for a compiler language. The rules reference the symbol table to obtain the addresses of variables used in their conditions. The blackboard facts are kept in a separate symbol table that is allocated in shared memory. Since all blackboard entries are uniquely defined by a "bb" prefix in their names, it is easy to make all of the references to blackboard values use the special symbol table while all other references use the symbol table that is local to the processor on which the rules are being executed.

In contrast to the organization of facts in PEBBLE, the facts in the CLIPS system are kept in a highly linked structure that specifically provides components to speed the execution of pattern matching. Each fact points to every condition with which it matches.

The rules in CLIPS are used to generate a pattern

network[7]. A node in the network represents the basic pattern of any fact that would satisfy a condition of a rule. When a fact matches a pattern, it is then further examined to bind the variables that may be used in more specific relations that must hold. After a fact is added to the working memory, or fact list as it is called in CLIPS, the fact is "pushed" through the pattern network. During this process "tokens" that represent matches of the fact with the patterns for the conditions in the rules are generated and distributed in the network. As a result, the network stores a knowledge of the "matches" that have been made at any particular point in time.

The nodes are arranged in a manner so that each path in the network represents the set of conditions that are necessary to fire a rule. That is, if all of the nodes in a path were to have their conditions satisfied, the terminal node would identify a rule whose conditions have all been met. In essence, the network "remembers" what conditions have been met up to any particular point in time and processes new facts from that partial match. Thus new facts "add" to the partial match information and may result in the completion of requirements for a rule to fire.

This algorithm provides a very efficient way of determining the effect on the rules that should be produced when a fact is asserted. However, in order to obtain this efficiency the technique has deposited "memories" of the fact within the pattern network that represents the rules. If a fact is deleted, these "memories" must be removed; and in order to make the removal efficient, it is necessary to have pointers from the facts into the areas of the network where the "memories" are kept.

In order for MARBLE to provide a blackboard architecture similar to that used in PEBBLE, the domain expert programs and the control expert must execute their own CLIPS systems on their own processors. But this presents a real problem with respect to how separate CLIPS systems can share the facts that would be on the blackboard. For example, suppose rule 1 in one domain expert references the fact, "(bb wall 2 thickness 8)". Let us suppose that rule 2 in another domain expert references the same fact. If the fact is asserted onto the blackboard, then both of the domain systems need to "push" the fact through their respective pattern networks. This means that the address of the fact must be available to both domain systems. By using shared memory, the address could certainly be available to both. But the fact must also point into both of the networks to preserve the CLIPS code that keeps track of the matches that have been "remembered" in the network. This requires that the pattern network be in shared memory as well.

Placing the pattern networks and the blackboard into shared memory is not a very difficult task. But the C functions that implement CLIPS expect for the fact list to be one highly-linked structure. Since the blackboard facts need to be in

shared memory, this implies that the local facts must also be in shared memory; or there could be two fact lists, one of which is in shared memory while the local fact list could be in local memory. In fact, an attempt to implement MARBLE with a shared memory blackboard, separate from a local fact list was attempted. The approach was abandoned as a series of changes to CLIPS code became necessary - changes that would compromise the ease with which new versions of CLIPS could be used for the system.

Instead of implementing the blackboard as one shared fact list, each expert shell now keeps a copy of the blackboard in its own fact list. At first, it may seem that this approach would produce a system inferior to the traditional blackboard model, in which the experts examine a common blackboard. However, just as distributed databases have achieved advantages over traditional localized databases, it will be noted later that there are some major conceptual advantages to the copied blackboard approach over the common blackboard implementation.

The implementation of MARBLE requires some basic changes to the manner in which CLIPS shells run. In order to simply the communication between the CLIPS shell that executes the control expert system and the CLIPS shells that execute the domain expert systems, all CLIPS data structures are stored in shared memory. Also, the run loop in each CLIPS shell is modified so that an exec function is called prior to the first rule firing, as well as immediately after each rule firing. In the event that an expert reaches a point where its agenda is empty, a special exec function is called repeatedly. The exec function invokes C functions that examine the action descriptor for the shell. As a result, every domain CLIPS shell checks for communication from the control expert, and the control expert CLIPS shell checks for communication from each domain expert after firing at most one rule.

Action Descriptors.

Action descriptors provide the link between the control expert system and the domain expert systems. Each domain system has an action descriptor in which it in can receive a request from the control unit, send a request to control and record its status to allow control to monitor its activities.

The interaction between the control expert system and the domain expert systems is configured by a finite-state machine diagram to make certain that there can be no deadlocks or uncontrolled interference. Whenever an action is initiated, the action descriptor is modified to show the state change that has occurred. When the action is finished, the action descriptor is again changed. Simple checking of the action descriptor guarantees that a change is proper, or the change is postponed. The control and the domain systems have their own areas within the action descriptor to permit concurrent action from both expert systems. The fields of the action descriptor and the

possible values they represent are shown in Table 1.

MARBLE Architecture.

When the MARBLE system is started, it will do some initialization and then fork CLIPS loaders to each of the processors. In particular the action descriptors for the domain expert systems are initialized to indicate that they are "IDLE". The system waits until all of the loaders are ready to execute and then it begins to execute only the loader for the control expert system. The loader prompts the user for the filename of the CLIPS control expert. As a matter of form, the control expert must contain rules that use the function "activate" to initiate the loading of domain experts.

The control expert can start domain experts at any time. The function "any_idle" will tell the control expert if there are any CLIPS shells available for a new domain expert.

Loaders.

A loader is a modified CLIPS program. There are three versions of loaders, as follows:

* domain loader
* control loader
* I/O loader

The domain loader is a CLIPS shell that has been modified to examine the action descriptor of the processor on which the loader is executing. It will examine the action descriptor before each execution of an action on its CLIPS agenda. This makes certain that the domain expert system will pay immediate attention to the requests that come from control. If there are no items on the CLIPS agenda, the domain loader will continually examine the action descriptor, waiting for instructions from control. When the control system wishes to have the domain loader execute a domain expert system, it places the filename of the CLIPS domain ruleset into the action descriptor for the domain and then changes the action descriptor to indicate its wish for the domain loader to execute the domain ruleset. The domain loader reads the request in the action descriptor and executes a standard CLIPS "load" of the rules. Then, if the file loads without error, the domain loader changes the action descriptor to indicate it is beginning the execution of the domain expert system and executes a standard CLIPS startup, by asserting a CLIPS initial-fact. After this assertion, the domain loader runs as an enhanced CLIPS shell with a few new commands and the transparent examination of the action descriptor prior to the execution of each CLIPS command.

MARBLE User Functions.

MARBLE also requires the addition of several new user functions to the CLIPS language:

bb_assert, for the domain experts;
also, activate, any_idle, promote_fact, force_promote, bb_retract and exit_marble, for the control expert.

The functions used to affect the content of the blackboard are bb_assert, promote_fact, force_promote and bb_retract. These four functions use a new parser which is a modified version of the CLIPS assert parser. This allows them to be called with the same syntax as the standard CLIPS assert. When a domain expert wishes to suggest a fact for the blackboard, it calls bb_assert with the fact as an argument. This new command sets the domain_action field of the action descriptor for the domain to REQUEST_ASSERT and places a pointer to the fact into the dargl field. When the control system inspects the action descriptor of the domain expert, it will perform the standard assertion code, using the address of the fact in the shared memory used by the domain expert, and assert the fact to the fact list, with "bb_consider" as the first argument. Facts beginning with "bb_consider" are only under consideration for posting to the blackboard.

By using the status, control_action and domain_action values of the action descriptor as a triple to identify the state of the action descriptor, a finite state transition graph can be constructed to show the valid sequences of operations. For example, in figure 2 when a domain expert is running with no communication pending, the state is 300. If the domain expert executes a bb_assert, the action descriptor will be changed to 301. This provides the request to the control expert. Then it is possible that the control expert will make a request for the domain to copy a value from the blackboard, before the control performs the domain request and changes the domain_action value. Thus the action descriptor might become 311. If the control did not make such a request, it would perform the bb_assert action and then reset the domain_action value back to 300.

By constructing the entire finite state transition graph from the point of view as to what should be possible, it is relatively easy to verify the code responsible for performing the actions associated with the action descriptor states. It is particularly important in the parallel environment to provide a proof of the conceptual plan to prevent invalid interactions between the various processes. In effect, the values in the action descriptors are used as semaphores to provide mutual exclusion in critical areas.

The control expert uses rules that evaluate the facts with "bb_consider" in their first fields, to determine if they should be promoted to the blackboard. If so, the control expert must choose between using the force_promote function and promote_fact. Both functions replace the fact with one that has a first field value of only "bb" and set the action descriptors of all active domain experts, to tell them to copy the new blackboard fact. The control_action value is set to ASSERT_BB and a pointer to the

fact to be copied into the domain fact lists is placed into the cargl field. The functions differ in the form of the fact they send to be copied. Force_promote points to a fact beginning with "bb", while promote_fact sends a fact whose first field is "idt_consider". The first field value will tell the domain experts whether they must immediately assert the blackboard fact, or if they can delay in accepting it.

It is natural that the control expert system should "decide" what should be placed on the blackboard. In fact, a major reason for the control expert is to arbitrate between the domain experts when they make different recommendations for values of the same entity. However, it may at first seem unusual that this privilege is also extended to the domain experts. But consider the following! It is often the case that with a team of human experts, even after agreement has been reached by the group as a whole, an individual may continue to think differently. Moreover, if a domain expert feels that a particular attribute should have a specific value, important advice might be lost if the domain expert were forced to override its opinion. Control can ignore the continual suggestion of a value , but if the domain expert is "turned off" by a forced value, the control expert would not be receiving the best advice. Furthermore, the system can insulate itself from a cascade of trivial changes by allowing the domain experts to determine when to update their values of a blackboard fact. In the design environment, it is very possible that small changes should be ignored in the beginning phases of the design work, and that the domains can execute rules to identify different tolerances to use as the design progresses. It is also possible that when a major change is made in a drawing, the domains may be able to recognize this event and delay in accepting a quickly changing sequence of values for a blackboard fact until it is stable.

If the control expert uses force_promote exclusively, the domain experts will keep a very current copy of the blackboard. Remember that each domain expert can execute no more than one CLIPS action before checking its action descriptor, so the response is immediate. Also, since each processor will independently execute the assertion code to incorporate the fact into its own fact list, the entire process takes just a little over the time it would take to assert the fact into one fact list.

When a new domain expert system is loaded, it copies all of the blackboard values into its fact list. The control does not execute control rules until the copy is completed to guarantee the agreement of the blackboard contents between control and the domain, and to prevent any contamination of the blackboard. Thereafter, the blackboard assertions take place with a single fact at a time. Thus, the execution of the system is only slowed appreciably by the loading of new expert systems.

When a domain expert finishes its work, it performs a CLIPS

halt. Then its CLIPS loader returns to the IDLE status, awaiting the loading of a new domain expert. When the control expert is finished, it calls the exit_marble function, which commands each domain process to exit. Exit_marble makes certain that all of the other processes have been killed before it exits.

Conclusions.

MARBLE has been used to implement a multi-person blackjack simulation in which the players execute in parallel. The design has also been used as a model for a distributed version of the blackboard that is currently being used with three networked computers for the first ICADS prototype system[8].

The most important result is that MARBLE provides a platform for experimentation in the development of techniques for synthesizing the efforts of concurrent expert systems. Moreover, the parallel environment provides this platform without the use of the complex scheduling algorithms that are needed in most blackboard systems. In addition, the use of shared memory eliminates the need for message passing, common to distributed blackboard systems.

When a CAD workstation that can execute the specific drawing system used in the ICADS prototype is added to the parallel system on which MARBLE runs, MARBLE will be used to execute the ICADS prototype with a greater degree of concurrence than the current networked system can provide.

References.
1. Pohl, J., A. Chapman, and L. Myers; 'ICADS: An Intelligent Computer-Aided Design Environment'; Proc. of ASHRAE Symposium on Artificial Intelligence in Building Design, St. Louis, IL., June 1990.

2. NASA; 'CLIPS Architecture Manual (Version 4.3)'; Artificial Intelligence Section, Lyndon B. Johnson Space Center, NASA, May 1989.

3. Hayes-Roth, B.; 'A Blackboard Architecture for Control'; Artificial Intelligence, Vol. 26, 1985.

4. Nii, H.P.; 'Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures'; The AI Magazine, Summer 1986.

5. Klein, M.; 'Conflict Resolution in Cooperative Design'; Thesis, Computer Science Dept., University of Illinois, Urbana, IL., 1990.

6. Myers, L. Cheng, Erikson, Nakamura, Rodriguez, Russett and Sipantzi; 'PEBBLE: Parallel Execution of BlackBoard-Linked Experts'; Proc. SURF Conference, Newport Beach, CA., Sept. 1988.

7. Forgy, C.L.; Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem'; Artificial Intelligence, Vol. 19, No. 1, 1982

8. Myers, L. and J. Pohl, 'ICADS: DEMO1 - A Prototype Working Model'; Fourth Eurographics Workshop on Intelligent CAD Systems, Paris, France, April 1990.

FIELD	DESCRIPTION
status	current process status
control_action	action requested by control
domain_action	action requested by domain
proc	process id
carg1	fact pointer argument from control
carg2	string argument from control
darg1	fact pointer argument from domain

FIELD USAGE

FIELD	VALUE	DESCRIPTION	USE
status			
	-1	ERROR	error identification
	0	IDLE	free for new use
	1	READY_TO_LOAD	load sequence flag
	2	LOADING	domain is loading carg2 file
	3	RUNNING	domain is executing CLIPS
	4	STALLED	domain agenda is empty
	5	BB_COPY	domain requests blackboard
	6	HAS_EXITED	domain process is dead
control_action			
	-1	ERROR	error identification
	0	NONE_CURRENT	no current control command
	1	ASSERT_BB	control is sending new fact
	2	RETRACT_BB	control requests retraction
	3	COMMAND_EXIT	control commands an exit
domain_action			
	-1	ERROR	error identification
	0	NONE_CURRENT	no current domain request
	1	REQUEST_ASSERT	domain requests BB assert
	2	unused	(domains do not request retraction)
	3	DONE	domain CLIPS has exited

TABLE 1.ACTION DESCRIPTORS

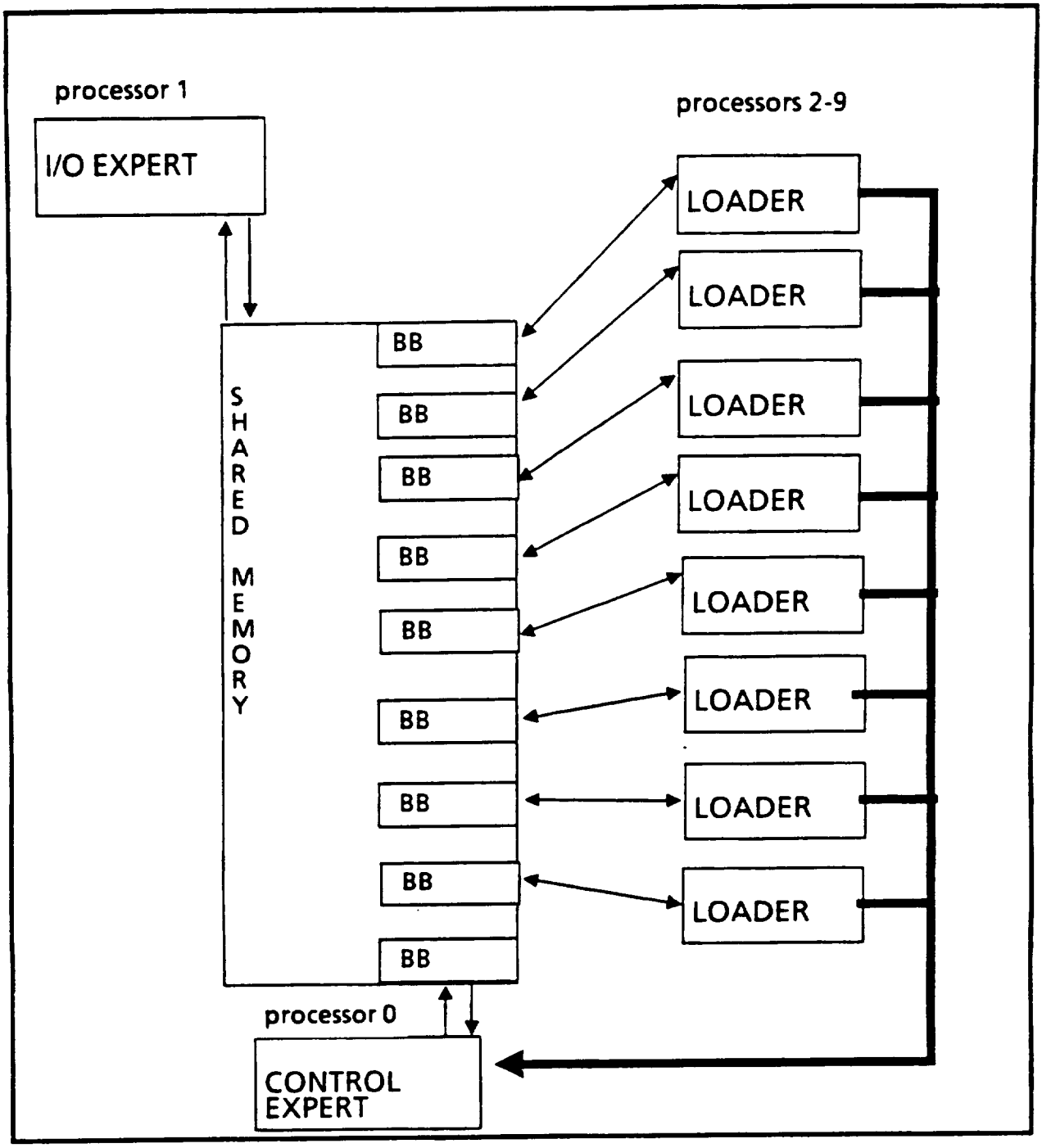

Figure 1: MARBLE Architecture

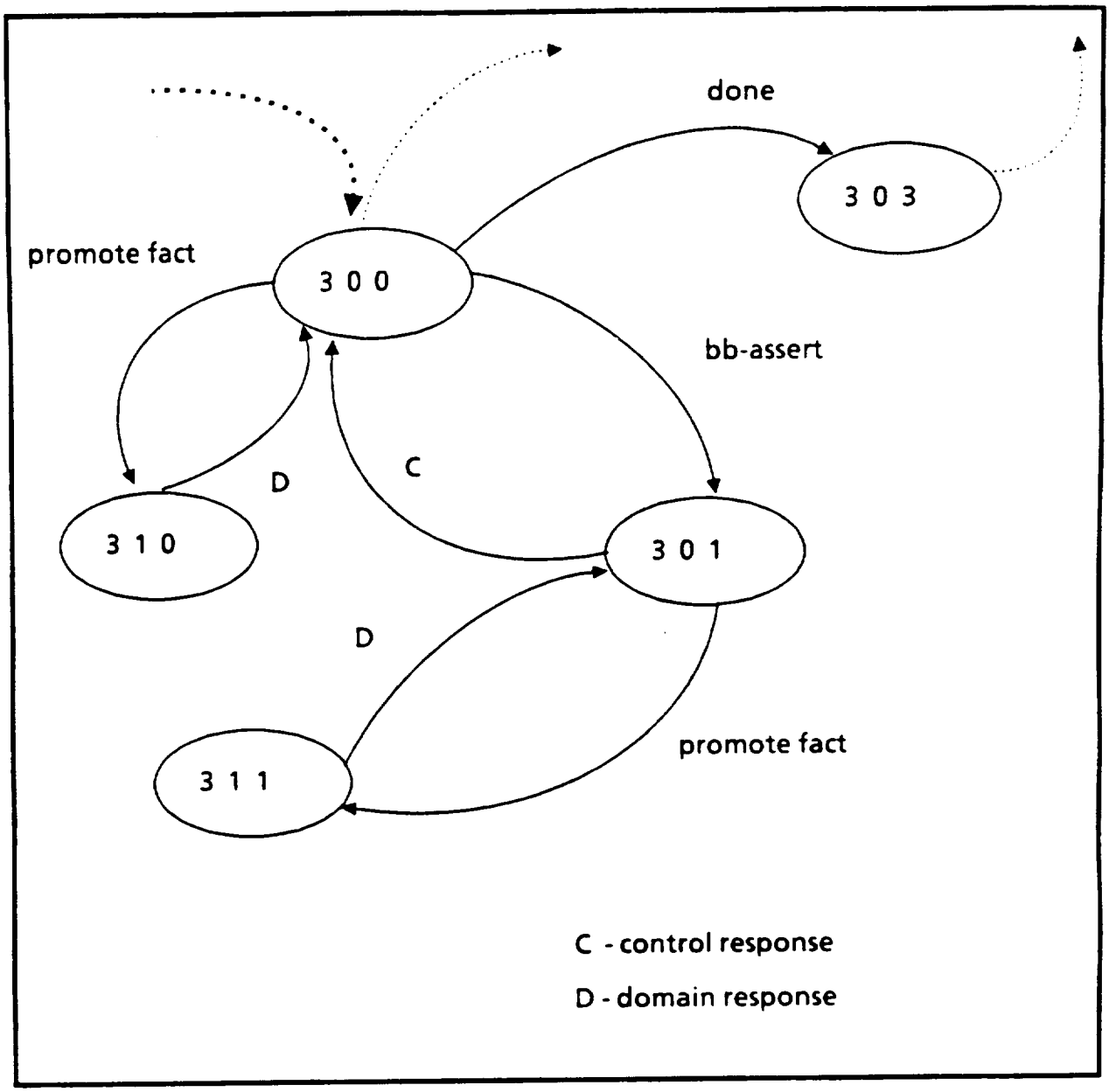

Figure 2: Partial State Transition Graph