BB_CLIPS: Blackboard Extensions to CLIPS

Robert A. Orchard, Aurora C. Diaz

Lab for Intelligent Systems, Division of Electrical Engineering
National Research Council of Canada
Ottawa, CANADA K1A 0R6

NRCC Publication No. 31505

aurora@ai.dee.nrc.ca, bob@ai.dee.nrc.ca

Abstract

This paper describes a set of extensions made to CLIPS version 4.3 [1] that provide capabilities similar to the blackboard control architecture described by Hayes-Roth [2]. There are three types of additions made to the CLIPS shell. The first extends the syntax to allow the specification of blackboard locations for CLIPS facts. The second implements changes in CLIPS rules and the agenda manager that provide some of the powerful features of the blackboard control architecture. These additions provide dynamic prioritization of rules on the agenda allowing control strategies to be implemented that respond to the changing goals of the system. The final category of changes support the needs of continuous systems, including the ability for CLIPS to continue execution with an empty agenda.

Keywords: CLIPS, blackboard, dynamic control

1. Introduction

This paper describes changes that add a blackboard control architecture to CLIPS version 4.3 and enable the operation of continuous systems. This extended version of CLIPS is called BB_CLIPS.

One class of modifications implements changes in the syntax of CLIPS, allowing the facts base to be partitioned into appropriate user defined blackboards and levels within a blackboard. A second class implements changes in CLIPS rules and the agenda manager to incorporate some of the powerful features of the blackboard control architecture. These include modifications that allow for (1) a more detailed description of the features of a rule in its declare section, (2) the use of special rules to manage problem-solving control and strategy decisions, and (3) the use of a combining function to bring together the current control and strategy decisions with the features of the rules to calculate the current priority of each rule on the agenda. A third class of modification implements changes in the functionality of CLIPS to facilitate the operation of continuous systems. These enhancements include (1) the extension of the *run* command to receive other parameters that allow BB_CLIPS to continue executing even with an empty agenda, (2) the addition of *runstart* and *runstop* functions (very much like the exec functions of CLIPS) which are invoked whenever the run command is executed or terminated, and (3)

the addition of a function that, when executed, changes the recency control strategy from most-recent to least-recent.

The use of the above modifications are optional and existing CLIPS programs will execute correctly with no changes. In addition, it should be noted that these modifications add very little runtime overhead (in some cases it is faster than the unmodified CLIPS).

Section 2 describes changes made to CLIPS to implement the blackboard control architecture and discusses the first two types of modification. Section 3 describes the changes that enable the operation of continuous systems. And finally some discussion of the use and future of BB_CLIPS is presented in section 4.

2. Blackboard Architecture

A blackboard-based system consists of three basic components:

1. The knowledge sources which are separate and independent modules of knowledge needed to solve the problem.

2. The global blackboard structure that contains the problem-solving state data. The knowledge sources post changes to the blackboard that incrementally build a solution to the problem. Communication and interaction among the knowledge sources are through this blackboard.

3. The scheduler that supervises knowledge source execution and blackboard access.

In BB_CLIPS each CLIPS rule serves as a knowledge source, its facts base as the blackboard, and its agenda manager as the scheduler.[1]

2.1. Specifying Blackboard Locations

A blackboard-based system is usually organized into one or more blackboards that are partitioned into various levels according to the needs of the application (see Figure 1). The syntax of the facts and patterns in CLIPS has been modified to allow the system designer to clearly specify the two components of the blackboard data; the blackboard entry or relation which is the information content of the data and the blackboard specification which indicates the location within the blackboard structure where this information is stored.

The following (1) illustrates the syntax of a fact that is associated with a particular blackboard and placed at a specified level within that blackboard.

```
(status PUMP1 ON) $in (component_bb pump 100)        (1)
```

The relation *status* contains the information that *PUMP1* is *ON* and this information is found in the *component_bb* blackboard with value *pump* in the component type level and value *100* in the time level.

---

[1]In this document the term *rule* and *agenda manager* are used when talking about BB_CLIPS and *knowledge source* and *scheduler* when talking about the blackboard architecture in general.

In general a blackboard specification has the following syntax:

$in (bb_name level1 ... leveln)

where *$in* is the delimiter separating the relation information and the blackboard specification. The information between the parentheses identifies the name of the blackboard and any sublevels within it.

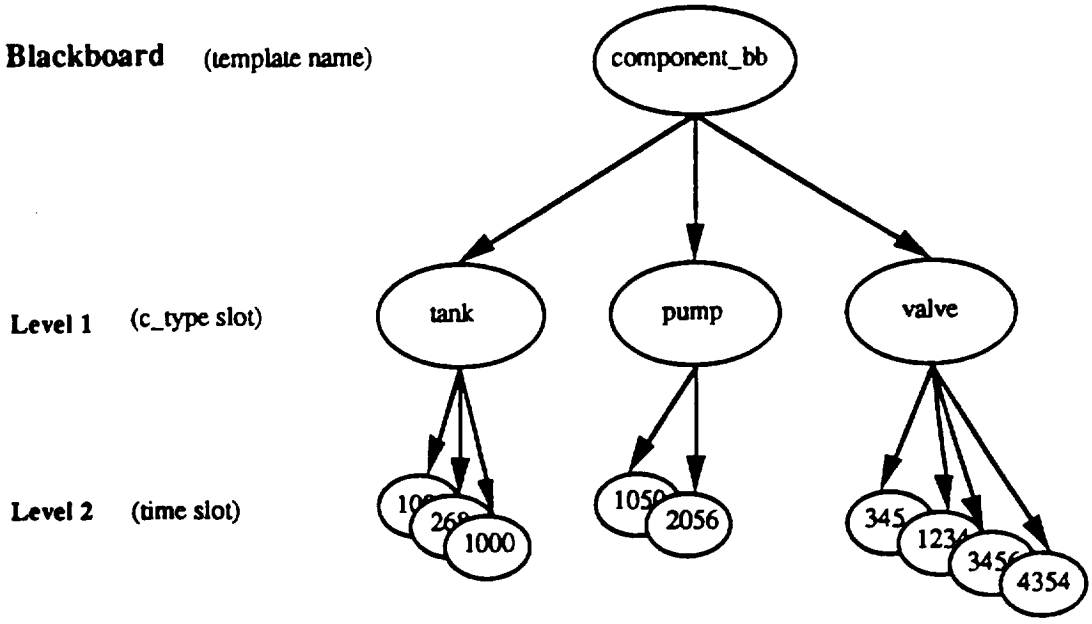

Figure 1 – Blackboard Structure

With CLIPS version 4.3, templates may be used to describe a relation more fully. Similarly, in BB_CLIPS 4.3, a template can be used to describe the relation and another to describe the blackboard specification. Consider the following template definitions:

```
(deftemplate status (field c_instance (type WORD)) (field has_value (type WORD)))
(deftemplate component_bb (field c_type (type WORD) (field time (type NUMBER)))
```

Fact (1) above may be rewritten, given the above template definitions, as:

```
(status  (c_instance PUMP1)  (has_value ON))
         $in (component_bb (c_type pump) (time 100))                    (2)
```

No distinction is made between templates used to describe relations and those used to describe blackboard specifications. Any operation that is valid for a relation template is valid for a blackboard specification template. Thus, to change the blackboard specification and one of the relation slots for fact (2) above, the following modify command could be used:

(modify ?fact_id (has_value OFF) $in (time 200))

This modify command retracts the old fact (status PUMP1 ON) $in (component_bb pump 100) and asserts the new fact (status PUMP1 OFF) $in (component_bb pump 200). The fact must have been previously bound to ?fact_id.

For a single fact, template and non-template relations and blackboard specifications may be mixed[2]. The modify command may be used only for templates, therefore, given a fact that has a non-template relation and a template blackboard specification, only the slot values in the blackboard specification may be *modified.*

2.2. Blackboard Control Architecture Features

The blackboard architecture has been implemented in many different ways. One such implementation, developed at the Knowledge Systems Lab at Stanford University, allows the system to reason about and explicitly represent control decisions on knowledge source firing. It is called the blackboard control architecture [2]. This allows for the unification of goal-directed and data-directed control which forms the relationship between actions and results that is needed in order to make intelligent control decisions [3].

The blackboard control architecture separates knowledge sources into two types. The first is used to solve the domain problem and knowledge sources of this type are called domain knowledge sources. The second deals with solving the control problem; that is, to determine which of the potential actions (rule firings) to perform at each point of the problem-solving cycle. These are called control knowledge sources and they embody the strategy and control knowledge or meta-level knowledge of the system. There are also two types of blackboards. One type are called domain blackboards and contain decisions made when solving the domain problem. The other hold decisions made when solving the control problem and are referred to as control blackboards. Also there is a single scheduler that supervises knowledge source execution and blackboard access for both types of knowledge source and blackboard. The scheduler decides which knowledge source to execute and considers (1) the features of the knowledge sources which have been triggered and are currently on the agenda, (2) the decisions that have been posted on the control blackboard(s), and (3) some combining or integration function to determine current priorities for the knowledge sources on the agenda.

In BB_CLIPS there is no difference in the syntax that distinguishes domain and control rules. Also, the organization of both the control and domain blackboards are left to the system designer. The next subsections describe additions made to CLIPS that allow flexible and dynamic prioritization of rules.

2.2.1. Declare Section

Standard CLIPS allows a static *salience* to be specified in the declare section of a rule definition. This is used to order the rules found on the agenda. In BB_CLIPS, the declare section is enhanced to allow a more detailed specification of the *features* of a rule. Feature

[2]Each template may have only one multifield slot. For a fact with a template relation and a template blackboard specification, the template relation may have one multifield slot and the template blackboard specification may also have one multifield slot.

values may be integers, elements of a predefined set (e.g. low, alarm), or a blackboard specification (e.g. $in (interface_bb operator_cmd)).

Consider the following declare section of a rule:

```
(declare                                                    (3)
        (salience 100)
        (problem alarm)
        (efficiency low)
        (importance 5)
        (focus $in (interface_bb operator_cmd))
)
```

This declares that the rule belongs to the set of rules dealing with the alarm problem and that it has a salience of 100, a low efficiency, an importance of 5 and will produce a blackboard entry in the operator_cmd level of the interface_bb blackboard. This interpretation is determined by the system designer, as are the features that are needed for the problem at hand.

The declare section of each rule is validated when the rule is loaded. The rule compiler will check the syntax of a feature and ensure that the values for each feature are allowable. Therefore, each feature must be identified by the system designer in a file containing declaration definitions for each feature that is to be allowed in the rules. This file is compiled and linked with BB_CLIPS providing the predetermined set of features[3]. The system designer specifies the feature names and the valid values that these features may take. For a feature of type *integer* this means defining a valid range; for a feature of type *set* this means enumerating the valid set members; and for a feature of type *blackboard specification*, no validation is done because the blackboard organization is determined dynamically. Below is part of such a feature declaration.

```
struct declare_template valid_declarations [] =
  {
        {"salience", SALIENCE_FEATURE, &salience_range ,NULL},
        {"reliability",INTEGER_FEATURE, &reliability_range, NULL},
        {"efficiency", SET_FEATURE, NULL, &efficiency_set},
        {"focus", BB_SPEC_FEATURE, NULL, NULL},
        {"problem", SET_FEATURE, NULL, &problem_set},
        {"prob_type", SET_FEATURE, NULL, &prob_type_set},
        {"sub_type", SET_FEATURE, NULL, &sub_type_set},
  };

struct set_descriptor efficiency_set =
        { 3, efficiency_set_mem};

charptr efficiency_set_mem[] =
        {"low", "medium", "high"};
```

[3] This is similar to the method for adding user defined functions to CLIPS. The authors acknowledge that it would have been more flexible to allow the features to be dynamically created and loaded when BB_CLIPS starts up and this could be considered at some future date. Similarly the combining function used to determine dynamic priorities would also have to be attached to BB_CLIPS at runtime (this is more difficult).

2.2.2. Control and Intercept Rules

As stated earlier, there are separate knowledge sources that post control or metalevel decisions on the control blackboard. These decisions are taken into account when the scheduler is deciding which knowledge source to invoke, thereby providing dynamic prioritization of knowledge sources. For example, a decision on the control blackboard might specify that knowledge sources with efficiency of low or medium be given a certain weight. The scheduler when calculating priorities, will use this weighting factor attached to the efficiency feature for any knowledge sources that are currently triggered and for future knowledge sources as they become triggered. Later, should this control decision be retracted, the priorities of any triggered knowledge sources with the efficiency feature are recalculated immediately and future knowledge source priorities will also be adjusted.

In BB_CLIPS decisions posted on the control blackboard are asserted in much the same way as decisions posted on the other non-control blackboards. In addition, however, some *intercept rules* need to be included which when fired invoke procedures to store these decisions in a separate data structure which is available to the agenda manager. The assertion of the control decision:

```
(efficiency 100 == low medium) $in (control_bb policy)                    (4)
```

might, for example, cause the following intercept rule to be instantiated and added to the agenda.

```
(defrule intercept_cf_set                                                 (5)
        (declare (salience MAX_SALIENCE))
  ?f <- (?feature_name ?wt ?func $?val) $in (control_bb policy)
=>
        (set_cf_set ?f ?feature_name ?wt ?func $?val)
)
```

The intercept rule (5) above calls the external function set_cf_set[4] that ensures that the function (?func) is valid for the set type feature (?feature_name) and that the values given for the feature ($?val) are valid for the set feature. If all checks are passed, the weighting factor for the feature (?wt) is stored in a data structure used by the agenda manager when calculating the priorities of the rules on the agenda.

Intercept rules usually have a maximum salience so that they are executed immediately. Once the intercept rule illustrated in (5) is executed, all rules in the current and succeeding agendas that declare either a low or medium efficiency are given priorities that take into account the control decision made in (4) – until this control decision is retracted. The next two rules are examples of intercept rules for the integer and blackboard specification features.

```
(defrule intercept_cf_int
        (declare (problem intercept))
  ?f <- (?feature_name ?wt ?func $?val) $in (control_bb policy)
=>
        (set_cf_int ?f ?feature_name ?wt ?func $?val)
)
```

[4] There are predefined external functions to handle integer, set, and blackboard specification features. These are set_cf_int, set_cf_set, and set_cf_BBspec respectively.

```
(defrule intercept_cf_BBspec
        (declare (problem intercept))
  ?f <- (?wt $?BBspec) $in (control_bb focus ?type)
=>
        (set_cf_BBspec ?f ?type ?wt $?BBspec)
)
```

There are predefined functions associated with integer and set features. For integer type features these are <, <=, >, >=, ==, !=, IN_RANGE, and NOT_IN_RANGE. For set type features these are == and !=. The operation of these may be changed or new functions may be added by modifying appropriate files. Only functions previously defined as valid for the different feature types may be used in facts asserted by the control rules to reason about the features. For instance, if a control rule concludes that rules with low or medium efficiency should have a weighting factor of 100, given the *current state of the problem*, then it could assert a fact of the form illustrated in (3). This fact makes use of the efficiency set feature and the == (equality) function which has been predefined for set type features.

2.2.3. Combining Function

The agenda manager in BB_CLIPS uses the feature declarations of a rule and control decisions plus some predefined combining function to determine a priority for a rule. The features of a rule are set when the rule is loaded and can be changed only by modifying the rule definition and then reloading it. Control decisions are posted on the control blackboards and are trapped by user-defined intercept rules (as explained in the previous section). Upon execution of one of the functions set_cf_int, set_cf_set, or set_cf_BBspec, the priorities of the rules currently on the agenda are recalculated to incorporate the new control decision.

A predefined function is used to combine the control decisions and the features of the rules on the agenda to determine the priority of a rule. Consider the following control decisions:

```
(problem 500 == alarm)  $in  (control_bb policy)
(200 interface_bb operator_cmd)  $in  (control_bb focus strategic)
(efficiency 100 == low medium)  $in  (control_bb policy)
(importance 10 IN_RANGE 0 5)  $in  (control_bb policy)
(importance 20 IN_RANGE 6 10)  $in  (control_bb policy)
```

If the combining function adds the weights assigned to the set and blackboard specification features and adds the product of the value of the integer features and the weight assigned to these, then a rule in the agenda with the declaration shown in (3) will have a priority of:

$$500 + 100 + 100 + (5 * 10) + 200 = 950.$$

The above combining function is defined in a file that is provided and may be modified by the system designer as necessary to fit the problem at hand.

3. Continuous Operation and Other Additions

This section describes further extensions made to CLIPS to address the needs of continuously operating systems and to provide other features that were found to be useful.

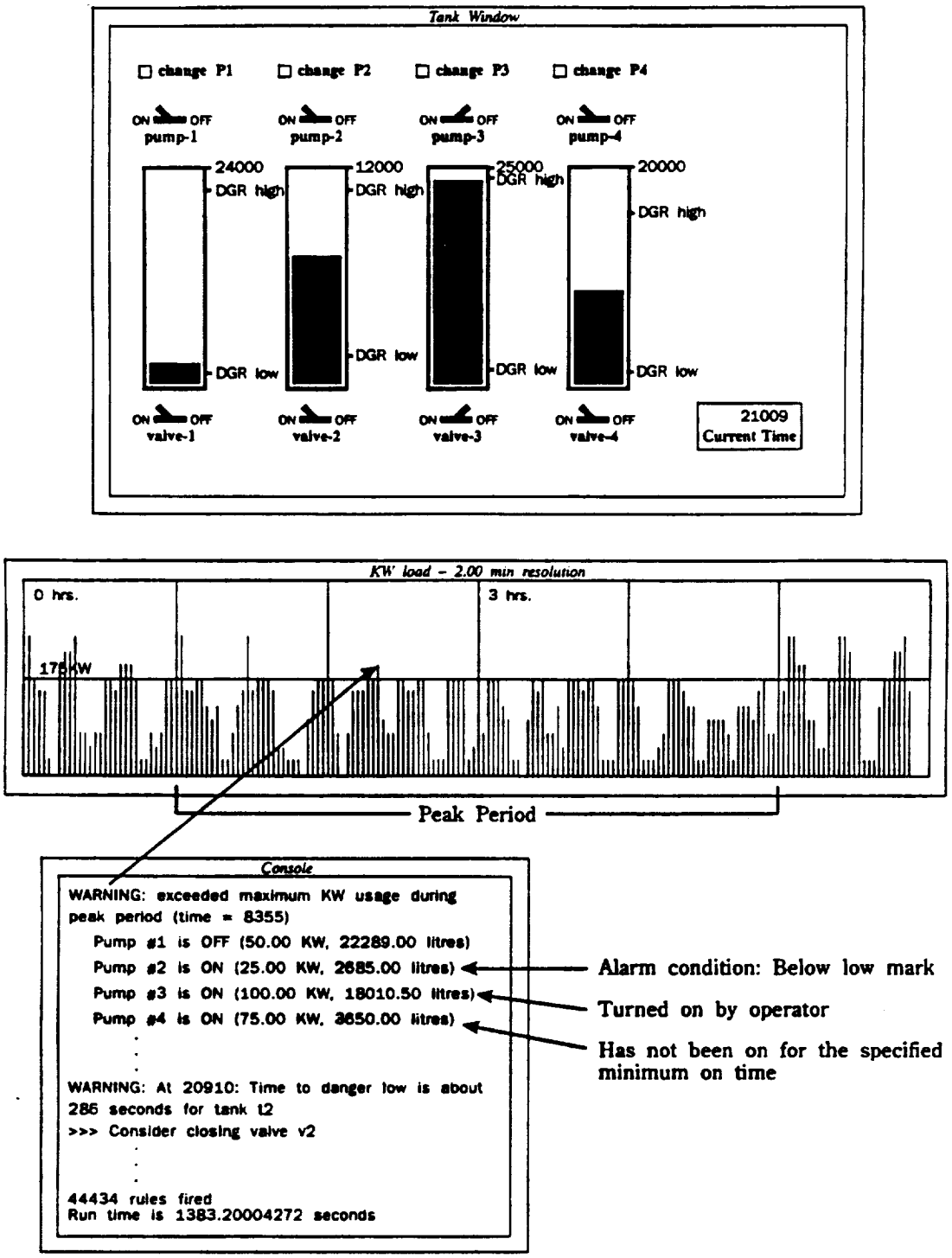

Figure 2 — Test Program

3.1 The Run Command

Normally, CLIPS terminates when the agenda is empty. For real-time systems (or any continuously operating system) there is need for a mechanism that allows the inference engine to idle, waiting for events to occur without executing a dummy idle rule. In BB_CLIPS, the run command was extended to receive any of the following parameters:

A positive integer n.
BB_CLIPS will run until n rules have executed or until the agenda is empty, whichever comes first. e.g. (run 10)

-1.
BB_CLIPS runs until the agenda is empty. e.g. (run -1)

-2.
BB_CLIPS runs forever (in an idle state if no rules are on the agenda). e.g. (run -2)

A negative integer -n (less than -2).
BB_CLIPS runs until n rules have executed (in an idle state if no rules are on the agenda). e.g. (run -10)

The *halt* function or a keyboard intercept (e.g. control-C) may halt the execution of CLIPS at any time.

3.2 Runstart and Runstop Functions

A list of external functions that are executed at the end of each cycle of the inference engine (i.e. after each rule firing) can be created. This is done with the *add_exec_function* of CLIPS. In certain cases, however, it is useful to be able to execute special routines on entry or exit from the run command. The *runstart* and *runstop* functions of BB_CLIPS provide such a capability. Consider the situation where a simulation is being done and a clock driven by the time of day is used to keep track of the simulated time. When the system is stopped (when n rules have been fired after the (run -n) command or a control-C interrupt occurs, for example), the simulated clock should not advance. When the system continues, the clock should resume from where it left off when the system was stopped. In this case the addition of a runstart and a runstop function will allow the appropriate adjustments to be performed.

A function is added to the list of functions to be invoked when the run command is executed by calling the add_runstart_function and it can be removed from this list by calling the remove_runstart_function. Similarly, a function is added to the list of functions called when the run command is terminated by calling the add_runstop_function and removed with the remove_runstop_function[5]. The following are examples of calls to these four functions.

```
add_runstop_function("haltTimer",haltTimer);
add_runstart_function("continueTimer",continueTimer);
remove_runstop_function("haltTimer",haltTimer);
remove_runstart_function("continueTimer",continueTimer);
```

[5] These external functions must have been previously defined as user functions [1].

3.3 Recency Control Strategy

If there are a number of rules on the agenda with the same salience, CLIPS chooses the last rule that was added to the agenda for execution (thus implementing a *most-recent-first* control strategy). It has been found that for some systems it is more important to execute the first of the rules added to the agenda (i.e. execute the least recent, as opposed to the most recent). In BB_CLIPS this is done by invoking the *set_most_recent_first* function on the right hand side of a rule with an argument of true or false (the system default is true)[6]. The following is an example of a rule that will set the agenda manager to give preference to rules (within the same priority grouping) added least recently to the agenda.

```
(defrule change-recency
  =>
    (set_most_recent_first false)
)
```

4. Discussion

The additions described in this paper have proved useful in practice. A test program was constructed which simulates a series of tanks being filled by turning pumps on and emptied by opening valves. The system monitors the tank levels trying to keep the tanks below some high level mark and above some low level mark, raising alarms when these conditions are violated. It also has to plan the use of the pumps such that the total power consumption at any given time during peak periods in the day remains below some predetermined value (this is to avoid surcharges by the power company). Additional functionality was developed to complete the program. This included: (1) a simulator written in C to control the reading of level sensors in the tanks and to control actuators which turn pumps on and off and open and close valves as required; (2) a graphical interface using the NeWS [4] system on Sun microcomputers (see Figure 2); and (3) a suitable blackboard structure to partition the problem (partially shown in Figure 1). A detailed discussion of this problem can be found in [5].

Other ways to provide the features described in this paper are being considered. For example, allow the dynamic specification of rule features and the combining function rather than requiring the creation of a separate version of BB_CLIPS for each problem specific set of features; use a special assert function (*control_assert*) to handle assertions into the control blackboard rather than the assert function and the intercept rules described herein; and allow the dynamic specification of an agenda selection function which currently always selects the highest rated rule on the agenda.

Future work may involve determining how to most effectively use CLIPS in a multiprocessor environment and in collaboration with other expert and non-expert systems in a multi-paradigm environment.

---

[6] Calling the set_most_recent_first function has the same effect as executing an intercept rule in that it causes the reordering of the agenda to occur. This, however, causes some problems for the current BB_CLIPS implementation. It does not keep information that determines when a rule is added to the agenda. When the current agenda is reordered, some rules that were previously at different priorities may now have the same priority and it is not possible to determine which rule was added first to the current agenda. Subsequent agenda additions, though, are prioritized properly.

References

[1] Artificial Intelligence Section. *CLIPS Reference Manual, Version 4.3*. Lyndon B. Johnson Space Center, August 1989.

[2] B. Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence*, 26:251-321, 1985.

[3] V.R. Lesser, D.D. Corkill, R.C. Whitehair, and J.A. Hernandez. Focus of Control Through Goal Relationships. In *IJCAI*, pages 497-503, 1989.

[4] Sun Microsystems. *NeWS Manual*. 1989.

[5] A.C. Diaz, R.A. Orchard. A Prototype Blackboard Shell Using CLIPS. Submitted to the *Fifth International Conference on AI in Engineering*, 1990.