KNOWLEDGE BASE RULE PARTITIONING DESIGN FOR CLIPS

Joseph D. Mainardi and Dr. G.P. Szatkowski

GENERAL DYNAMICS
Space Systems Division

5001 Kearny Villa Road
San Diego, CA 92138
(619) 496-7093

ABSTRACT

This describes a knowledge base (KB) partitioning approach to solve the problem of real-time performance using the CLIPS AI shell when containing large numbers of rules and facts. This work is funded under the joint USAF/NASA Advanced Launch System (ALS) Program as applied research in expert systems to perform vehicle checkout for real-time controller and diagnostic monitoring tasks.

The Expert System advanced development project (ADP-2302) main objective is to provide robust systems responding to new data frames of 0.1 to 1.0 second intervals. The intelligent system control must be performed within the specified real-time window, in order to meet the demands of the given application. Partitioning the KB reduces the complexity of the inferencing Rete net at any given time. This reduced complexity improves performance but without undo impacts during load and unload cycles. The second objective is to produce highly reliable intelligent systems. This requires simple and automated approaches to the KB verification & validation task. Partitioning the KB reduces rule interaction complexity overall. Reduced interaction simplifies the V&V testing necessary by focusing attention only on individual areas of interest.

Many systems require a robustness that involves a large number of rules, most of which are mutually exclusive under different phases or conditions. The *ideal* solution is to control the knowledge base by loading rules that directly apply for that condition, while stripping out all rules and facts that are not used during that cycle. The *practical* approach is to cluster rules and facts into associated "blocks". A simple approach has been designed to control the addition and deletion of "blocks" of rules and facts, while allowing real-time operations to run freely. Timing tests for real-time performance for specific machines under R/T operating systems have not been completed but are planned as part of the analysis process to validate the design.

BACKGROUND

The Air Force and NASA have recognized that our nation's current suite of launch vehicle systems has a number of problems making them inadequate for the projected needs after the late-1990's. High costs of above $2,000/lb of payload delivery, a low reliability, poor resiliency (standdowns of many months for current expendables), and limited launch rate capacity are reasons behind the joint USAF/NASA effort for an operational ALS and Shuttle 'C'. These will serve the commercial and DoD mission models beginning in 1998. In order to meet the goals of $300/lb and launch rates as high as 50 missions annually, these systems and their associated ground operations segment must be made as autonomous as possible, while at the same time improving reliability and safety. Under the ALS Program, a study was initiated to explore the use of knowledge-based system (KBS) techniques for the purpose of automating the decision processes of these vehicles and all phases of the ground operations segment by assessing the feasibility, benefits, and risks involved.

An expert decision aid is a software approach to solving particular problems that are constantly changing over time and are complex or adaptive in behavior, the opposite of an analytical problem that is basically deterministic. Examples of these types of problems are: the re-scheduling of a vehicle checkout due to a damaged cable; or, determining if a system is indeed faulty given conflicting sensor readings. These heuristic problems require a depth of knowledge and experience (art rather than science) to form solutions quickly. Expert systems embody that collection of knowledge and experience in modular pieces that are rules and facts that describe the proper thought process for a given set of circumstances arrived at by any path. It is this modular independence that makes expert systems attractive. The incremental improvement of knowledge and experience can be built and tested readily without re-testing the rest of the software system, unlike conventional software that is difficult to maintain in a day to day changing environment.

INTRODUCTION

This work was funded under a joint effort on the part of NASA and USAF for the Advanced Launch System (ALS) program, as applied research in expert systems. The implementation of robust, real-time expert systems is considered to be an important technological addition to the design and development of the ALS vehicle. In a document released by the Aerospace Industries Association of America (AIAA) in April 1989, Artificial Intelligence technology was identified as one of the eight key technologies for the 1990's. This selection was founded on the basis of greatest potential payoff, broadest application base, and highest leverage. With the success of non-real-time expert systems already established, it was important to produce real-time expert systems that would exhibit the benefits of the technology within the scope of the ALS program.

In attempting to select a candidate expert system development tool to implement robust, real-time production systems, speed was the most obvious factor considered. A leading candidate was the "C Language Integrated Production System" (CLIPS), developed at the Artificial Intelligence Section (AIS) at NASA/Johnson Space Center. Based on the fact that CLIPS is written in C (considered one of the faster programming languages), and its lean implementation, it was determined that it held the highest potential of all tools reviewed, for use in real-time expert systems applications. Another factor was that of software flexibility. CLIPS was chosen because of the nature of the tool. It is public domain software so therefore the source code can be modified. Its C implementation permitted easy imbedding into the application system and easy integration with most graphics libraries and other languages (for ex. LISP). And finally, it provided a hardware independent inference vehicle, allowing development on lap-tops and run-time multi-system integration on high performance workstations and multi-processor systems. Overall, the belief was that CLIPS would be the best available tool to realize robust, real-time expert system implementations within the ALS program.

One of the major problems facing robust, real-time expert systems is that there have not been enough successful systems to make a significant impact on the commercial or military community. This was the same type of problem that initially hindered the success of non-real-time expert systems. An earlier attempt at creating a modification of CLIPS for use in real-time applications, the Portable Inference Engine (PIE) by T. Le and P. Homeier of the Aerospace Corporation, has not yet been widely accepted by the CLIPS user community.

The approach that was chosen for the ALS program was to create a rule partitioning approach that would not require modification of the actual CLIPS code, but would instead be application-specific CLIPS code that could be used to reduce the knowledge base (the number of active rules and facts). This is effectively knowledge-base "control blocking". The examination of rules that do not fire during an expert system application's cycle is the unfortunate overhead that expert systems typically carry. There a only a few expert system development tools that implement the concept of knowledge base rule blocking, where production rules are organized into logical arrangements to facilitate better control of the execution of those rules. An example of this approach is found in IBM's Expert System Environment (ESE). The challenge was to create a similar rule partitioning approach in CLIPS. This would then permit the development of robust, real-time controller applications for use in the ALS program.

APPROACH

The approach taken to achieve real-time response using CLIPS was to control the number of production rules that would appear in a real-time window (0.1 to 1.0 seconds) during any cycle, without any significant degradation in performance - i.e. continual real-time operations. The method that was selected was one that would remove blocks of production rules and associated facts that were not being used during a real-time cycle and would add any blocks of production rules and facts that would be required during that same cycle. This is a procedure that is automated in some expert system development tools, but this automation has overhead that does not permit expert systems to perform in real-time. As an example, ESE uses "Function Control Blocks" to control rule sets. This is an effective method of controlling production rule examination, yet it does not permit real-time applications to be implemented.

In developing the CLIPS approach, overhead costs were unavoidable. The first cost was that each production rule would carry the overhead of an associated fact, in order to facilitate production rule removal. Each production rule is assigned a fact whose value is the rule name, which is then referenced by an **excise** statement. It was determined that this was the simplest and most efficient approach. The next cost was that production rule removal requires a CLIPS rule that will remove both the production rule and the associated fact. By using an **excise** statement to remove the rule and a **retract** statement to remove the fact (a pointer to the rule), all traces of the production rule are removed from the real-time cycle. It was determined that this was, again, the simplest and most efficient approach.

Another factor contributing to the overhead is that each production rule partition should (conceptually) be stored in an external file; I/O calls to external files are a major contributor to system performance degradation. A partial solution to this problem is to group together blocks of production rules that have common characteristics, and store the grouped rules in fewer external files. This will greatly reduce the number of I/O calls required in a real-time cycle, and therefore speed up the execution of the expert system application. To further reduce the overhead of excessive I/O calls, production rule partitions should be organized in an efficient manner. This can be done using a variety of different methods. Using mutual exclusivity of production rule partitions is very important in reducing I/O calls. By using mutually exclusive production rule partitions, all extraneous test level rules (see below) can be avoided. When organizing test level rules, attempt to organize production rule partitions in a manner that will not attempt to load duplicate production rules. This task can be performed by analyzing all conditions that are used to load each production rule, categorizing those rules, and putting all of the similar production rules in categorized external files. For rule removal, the simple rule is to remove only the production rules that are not needed for the upcoming real-time cycle. In other words, don't leave a production rule in the rule base if it's not needed during that cycle; it can be loaded at the next appropriate cycle.

Finally, each test level check, for both production rule block removal and loading, is in itself a rule. These test level rules can be reduced in number by finding conditions that are common within the application's testing conditions and incorporating them into fewer test level rules. When used across an entire expert system application, this type of control will greatly reduce the number of test level rules. Unfortunately, the test level rules cannot be removed. Therefore, if a test level rule will not remove at least three production rules, it is probably not efficient to execute that test level rule.

The test level rules must be executed at the beginning of each cycle. Therefore, they should be assigned the highest salience possible. In addition, these test level rules should be ranked, with the basis of the ranking taking on the characteristics of the application. For example, a system that has mutually exclusive blocks of production rules should have the mutual exclusivity test level rules examined first, and then examine the explicit test level rules. Test level rules remove subsets of production rules loaded during the previous cycle, and add production rules needed for the current real-time cycle. The test level rules can also turn on and off any facts generated by test level rules. This will further control production rule loading and removal. These facts can also be used to preserve rules that must stay resident in the production rule-base for multiple cycles. Typically, a test level rule with a salience of 10000 will assert facts that will have an effect on test level rules with a salience of 9999 and below. The rules with the salience of 9999 will then assert facts that will have an effect on test level rules with a salience of 9998 and below. This procedure will continue until all test level rules have been examined. The production rules that remain in the knowledge base for that cycle will then execute.

Figure 1 gives an example of loading and removing a "control block" (a collection of rules and facts).

CLIPS Knowledge Base Partitioning Scheme

Example: actual CLIPS code, loaded into a file (remove_rulea.clp)

```
(deffacts x (x 0))
(deffacts rulea_facts (rulea a0) (rulea a2))
(defrule a0 (x 0) => (fprintout t "Answer is 0" crlf))
(defrule a1 (x 1) => (fprintout t "Answer is 1" crlf))
(defrule a2 (x 2) => (fprintout t "Answer is 2" crlf))
(defrule a3 (x 3) => (fprintout t "Answer is 3" crlf))
(defrule a4 (x 4) => (fprintout t "Answer is 4" crlf))
(defrule remove_rulea (?kptr <- (rulea ?kname) =>
  (fprintout t "Going to remove rule " ?kname crlf) (excise ?kname) (retract ?kptr))
```

Procedure:

```
>(load "remove_rulea.clp")
>(reset)
>(facts)
f-0  (initial-fact)
f-1  (x 0)
f-2  (rulea a0)
f-3  (rulea a2)
>(rules)
a0
a1
a2
a3
a4
remove_rulea
```

[continued next column]

```
>(run)
Going to remove rule a2
Going to remove rule a0
2 rules fired
>(facts)
f-0  (initial-fact)
f-1  (x 0)
>(rules)
a1
a3
a4
remove_rulea
```

Figure 1 —CLIPS Knowledge Base Partitioning Scheme

CONCLUSIONS

There is significant interest in developing and implementing robust, real-time expert systems applications for the ALS Program. Real-time expert systems show great promise in reducing costs for ALS vehicle launches. With small rule sets of twenty to fifty rules and a real-time operating system, the performance of the rule partitioning approach was well within the limits of the one second real-time window. Future timing tests performed for large rule sets under a real-time operating system, will demonstrate the ability of CLIPS to perform in real-time.

There are additional benefits of the knowledge base rule partitioning approach. Using the approach outlined above, the application of generic verification and validation techniques for expert systems are more likely to succeed than with traditional expert systems. By efficiently grouping the production rules, modular testing and documenting of modifications and enhancements are easier to perform than non-modularized expert systems applications. In addition, each partitioned production rule set can be independently verified and validated. This is a significant advantage over expert systems with non-partitioned production rules, and can ultimately lead to lower expert system maintenance and enhancement costs. When compared to non-partitioned rule bases, modularized production rule bases are easier to document, and subsequently the documentation is better. Better documentation normally results in lower maintenance and enhancement costs. There is strong evidence that the ability to verify and validate expert systems will be a major factor in their expanded use in the military and commercial community.