N96- 12954

5-11 ~~~

EMBEDDED CLIPS FOR SDI BM/C3 SIMULATION AND ANALYSIS

p. 8

Brett Gossage and Van Nanney
Nichols Research Corporation
4040 South Memorial Parkway
Huntsville, AL 35802

ABSTRACT

Nichols Research Corporation is developing the BM/C^3 Requirements Analysis Tool (BRAT) for the U.S. Army Strategic Defense Command. BRAT uses embedded CLIPS/Ada to model the decision making processes used by the human commander of a defense system. Embedding CLIPS/Ada in BRAT allows the user to explore the role of the human in Command and Control (C^2) and the use of expert systems for automated C^2. BRAT models assert facts about the current state of the system, the simulated scenario, and threat information into CLIPS/Ada. A user-defined rule set describes the decision criteria for the commander. We have extended CLIPS/Ada with user-defined functions that allow the firing of a rule to invoke a system action such as weapons release or a change in strategy. The use of embedded CLIPS/Ada will provide a powerful modeling tool for our customer at minimal cost.

THE PROBLEM

Battle Management, Command, Control and Communication (BM/C^3) systems accomplish the automated control of tactical and strategic military systems. Large-scale BM/C^3 systems such as for the Strategic Defense System (SDS) present several difficult problems. Decision timelines are too short and amounts of information too vast for a human Man-in-the-Loop (MITL) to effectively control or interact with the system without automated decision making or decision support. It is unlikely (and undesirable) that any experience will be gained in actual combat for building a set of rules for an automated SDS decision system. It is also unlikely that the builders of the system will accept full automation of all decision functions. That is, the system designer will require "positive control" of the system by some human commander. Computer simulations of the system are the only currently available method to study these problems. These studies are done in two fundamentally different ways. One is to create simulated command centers with human participants and the other is to use detailed simulations with embedded rules of engagement.

Simulated,"mock-up," command centers with human participants drive real-time displays with discrete simulations or scripts. Separate simulations may generate the scripts independently in non-realtime mode. These scripts have to be generated separately since the run time for full-scale SDS simulations is generally to long for real-time displays. Automated decision software may also be used for decision aids. The main drawback of such studies is that the decisions of the commander cannot affect those parts of the simulation that are run off-line. Thus, the decision loops can only be closed for the more simple parts of the simulation. Closing this loop becomes a Hobson's choice between lowering model fidelity to close the decision loops and leaving some loops open to gain higher model fidelity. However, such simulations provide a means to study the appropriate decision aids and decision criteria for the human commander and provide training for command center personnel.

Another method for studying BM/C^3 decision making is to embed an expert system tool in a simulation of the system of interest. This tool may consist of an inference engine and a rule base.[1] This method allows the closure of all decision loops since running in real-time is not an issue. The main drawback of this method for SDS studies is that no experts exist with the knowledge necessary to derive the rule base. Some the rules can be generated from existing rules of engagement, from experienced SDS simulation engineers, or from personnel who have participated in mock-up SDS command centers. But other rules will have to be generated through experimentation. Rules deemed appropriate in embedded expert system experiments could provide guidance to commanders in mock-up simulators, thus the two methods may complement each other.

THE APPROACH

The requirements for BRAT presented us with several challenging problems. BRAT is required to simulate all phases of SDS operation including peacetime to wartime transition and reaction to failures. The BRAT simulation cannot assume any architecture for the system under study and hence must be able to assemble a simulation from a collection of predefined models. Since the MITL controls the peacetime to wartime transition of the SDS, a BRAT model must be constructed that models the decision processes of the commander. BRAT simulates the system with a large collection of models of varying levels of detail. The BRAT simulation framework integrates these models together employing object-oriented techniques and event graphs.[2] The models capture the physical characteristics of the system, the performance of the automated BM/C^3 functions and the control of the system exercised by the commander. While most of the models can be implemented in procedural code, a model of the commander requires the greater flexibility provided by declarative languages. In BRAT, one model, designated as Command_Defense, accomplishes the simulation of the role of the commander in an SDS system. We have chosen to embed an expert system in the Command Defense model. Command_Defense and its integration with this expert system (CLIPS) are the subjects of the rest of this paper.

To meet the BRAT requirements for modeling the role of the commander and the rules a commander would use to operate the system, we chose to imbed an inference engine in the Command_Defense model. It was further decided that a forward chaining engine would be appropriate since the BRAT simulation is an "event driven" environment.[3] Command_Defense is one of many models that are required for BRAT, so it was not feasible within cost or time constraints to implement an inference engine of our own. CLIPS provided the ideal solution since cost was zero. Also, CLIPS is designed to be embedded in other software which lowered the risk associated with interfacing to stand-alone expert system tools. The major work that remained then was to design and build the interfaces for asserting facts about the system state to CLIPS and to extend CLIPS with user-defined functions that allow rule firings to cause changes in the simulated system state and the current engagement strategy.

IMPLEMENTATION

The BRAT simulation executive and its models are implemented in Ada. As a proof-of-concept we implemented a prototype Command_Defense model using the C-Pragma interfaces provided with the C version of CLIPS. (The Ada version was not available at that time). While this was successful, it caused problems when ported from one environment to another since different Ada compilers implement the C-pragmas differently. A second solution which solved the portability problems was to build a fact file from the Ada code and then execute CLIPS through the operating system. The CLIPS rules wrote all commands generated as a result of rule firings to a file read in by the model when CLIPS terminated. This solution was also unsatisfactory since the process was much slower than a fully embedded design. When CLIPS/Ada became available, the model was redesigned to accommodate it. This resulted in the loss of some CLIPS features such as **bsave** and **bload** which are not now available in the Ada version. The added portability and ease of integration made the switch worthwhile, however.

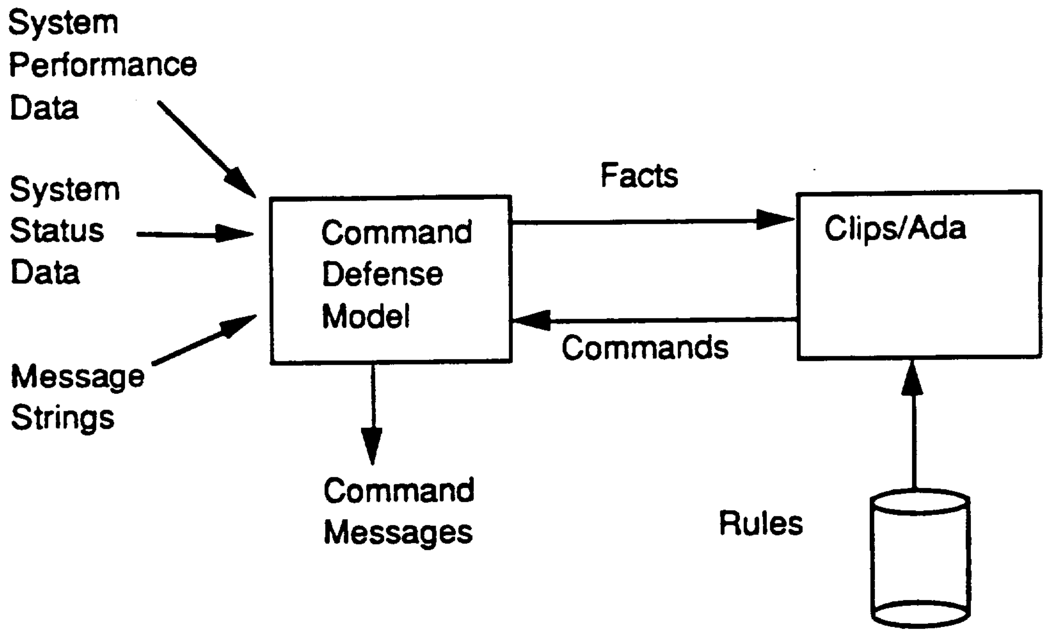

Figure 1. Model Interfaces.

The interfaces to the Command_Defense model occur through three routes. (See Figure 1) The first is the definition of the rules by the user. This is accomplished in the BRAT user interface in a text editor or in the CLIPS stand-alone program. The latter is probably preferable since the user can take advantage the CLIPS system to test the rules before their use by the model. The second interface is the assertion of facts about the current state of the system into CLIPS. This accomplished by converting system information into fact strings and asserting them into CLIPS. The rules bind quantitative system information to variables by pattern matching these facts. The third interface is through the extension of CLIPS with Right Hand Side (RHS) functions. These RHS functions pull information from the CLIPS buffers and insert it into global package data structures. The model reads these global data structures when new commands are to be sent to other models through the simulated communications system.

Rules for the Command_Defense model are divided into three basic types: time-based, relational, and free-form. Time-based rules fire on or after a given simulation time has been reached (see Listing 1). The time fact is bound and then reasserted to allow other time-based rules to fire. A lower salience rule eventually binds the time fact and retracts it. This allows the user to cause system actions to occur such as releasing weapons 300 seconds after simulation start. Frame-based rules use information generated as Ada records by other BM/C^3 models and sent to Command_Defense in messages. Free-form rules can follow any syntax desired and allow the user to define external string messages (such as those generated by external simulations) as Left Hand Side (LHS) patterns for firing rules.

```
(defrule release "A rule to release weapons at time t"
   (current time ?simtime) ;bind ?simtime to current time
   (release-time ?rtime)   ;bind ?rtime to release time
   (test (>= ?simtime ?rtime)) ; if time >= release t
   ( not(time1-past))          ; and rule not fired yet
=>                             ; then
   (assert (command RELEASE_WEAPONS))  ;release weapons
   (assert (time1-past))
)

(defrule retract-time " so other time-based rules fire"
   (declare(salience -1)) ; lower salience so that all
   ?timefact <- (current time $?); rules for current
                                 ; time fire first
=>
   (retract ?timefact)           ; retract time fact
)
```

Listing 1. Example of time-fired rule.

The user is responsible for creating and maintaining the rule-base for the Command_Defense model. Without detailed knowledge of the available fact patterns and RHS function syntax, this task could overwhelm the user and cause errors in rule execution. To ease this burden and assure proper use of the model, "defexternal" and "defrelation" statements provided to the Cross Reference Style Verification (CRSV) utility to assure rule validity [4]. The CRSV tool uses defexternals to assure that RHS function names and arguments are correct. An example defexternal is given is Listing 2. This definition assures that only the available weapon target assignment optimization modes are selected. Defrelations assure that LHS patterns for rules are consistent with the facts asserted by the model. An example defrelation is given in Listing 3. This definition assures that the user does not define a rule for which no valid fact pattern will exist. It also helps to assure that the proper variable bindings will occur.

```
(defexternal SET_OPT_MODE
   (true-function-name SET_OPT_MODE)
   (min-number-of-args 1)
   (max-number-of-args 1)
   (assert ?NONE)
   (retract ?NONE)
   (return-type NUMBER)
   (argument 1
      (type WORD)
      (allowed-words
         PREFERENTIAL_ASSET_BASED
         PREFERENTIAL_TARGET_BASED
         SUBTRACTIVE))
)
```

Listing 2. Example defexternal for CRSV.

```
(defrelation threat-data
   (min-number-of-fields 3)
   (max-number-of-fields 5)
   (field 1
      (type WORD)
      (allowed-words threat_class asset_class))
   (field 2
      (type NUMBER)
      (range 1 ?VARIABLE))
   (field 3
      (type WORD)
      (allowed-words count))
   (field 4
      (type NUMBER)
      (range 1 ?VARIABLE))
)
```

Listing 3. Example defrelation for CRSV.

All information about the state of the simulated system is input to CLIPS through facts. Current time is always asserted on each execution of the model. Simulated messages are sent to Command_Defense by other models and are received as Ada records or as strings. The Threat_Assessment and System_Performance models summarize available system information in data records and transmit them in simulated messages to the Command_Defense model. These records contain summary information for system element operational status, weapon system performance, assets threatened, and missile launch fields. Record fields are converted to strings and concatenated with appropriate description strings. For example, the fact (data threat_class 1 count 20) provides the type and number of threat objects of a given class currently detected. The rule base uses this threat information to make inferences about the objective and intent of an attack. The model asserts string messages directly so the user is responsible for assuring that the rules are consistent with them. String message facts allow the user to define arbitrary scenarios for a simulation run. These message facts are defined in an input exogenous event file along with a message arrival time for input to the simulation through event generators.

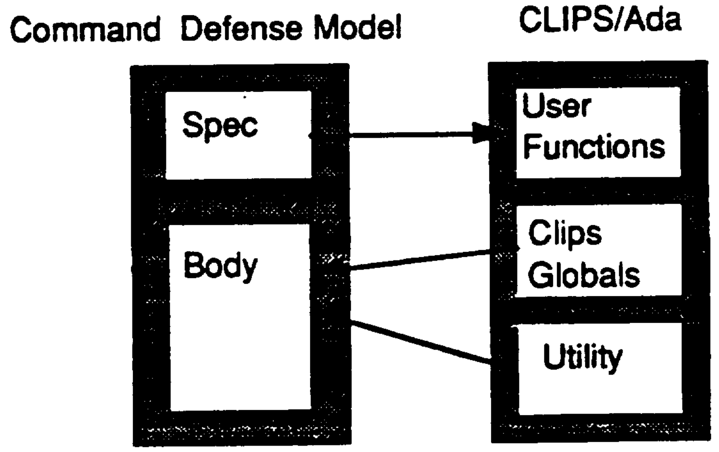

Figure 2. Software Architecture

The RHS functions added to CLIPS which change defense strategies, select types of assets to defend, specify weapon withhold, release weapons, or send strategy change messages. The software architecture for exporting these functions to CLIPS is shown in Figure 2. These functions are exported to CLIPS throughout the model Ada specification file while the code for the functions is kept in the model code body file. Each function pulls the function parameters from the CLIPS buffers and places them in a global strategy variable. When the SEND_STRATEGY_MESSAGE RHS function is invoked, the current strategy is sent to weapon control models. W e expect to continue to expand the number of RHS functions as the BRAT simulation grows. An example of a RHS function which sets the weapon withhold percentage is shown in Listing 4.

```
function SET_WITHHOLD
   (The_Problem : in CLIPS_GLOBALS.Test)
    return CLIPS_GLOBALS.Real is

----------  constants SET_WITHHOLD  --------------------

Check_Value : FLOAT_TYPE_PKG.Float_Type := 0.0;

----------- exceptions SET_WITHHOLD  ------------------

Probability_Out_Of_Bounds_Error : exception;

use FLOAT_TYPE_PKG;
begin
Check_Value := UTILITY.GET_FLOAT_ARGUMENT(The_Problem,1);
if (Check_Value > 1.0) OR (Check_Value < 0.0) then
   raise Probability_Out_Of_Bounds_Error;
end if;
Percent_Withhold := Check_Value;
return 0.0;
------------Exception ---------------------------------
   when Probability_Out_Of_Bounds_Error =>
      BRAT_ERROR_PKG.Log_Error
        ("Invalid probability retrieved from
        CLIPS buffer");
      Raise BRAT_ERROR_PKG.Cc_Function_Error;
end SET_WITHHOLD;
```

Listing 4. Example RHS function.

STATUS AND FUTURE PLANS

As of this writing the Command_Defense model is undergoing integration testing with the BRAT Simulation Executive. Time-based rule firings have been tested in a prototype simulation. The use of defrelation and defexternal statements in the User Interface for rule verification has been defined. All interfaces have been successfully tested and verified.

CONCLUSIONS AND RECOMMENDATIONS

Embedding CLIPS in Command_Defense has proven to be straight-forward, so long as both the model and the CLIPS version are written in the same language. The loss of the **bload** and **bsave** features in the Ada version restricted our ability to build simulations with multiple instances of Command_Defense models. Simulated systems with multiple commanders require multiple model instances for studying devolution of control when primary C2 nodes are lost. An added feature that would be useful in this regard is for **bsave** and **bload** to include the fact list along with the rules. This would allow saving the models perception of the system at a given time to a binary file for fast reloading later. We expect the rule base for the model to expand over time as more users take advantage of its capabilities. We will be defining a baseline set of rules to be delivered with the BRAT product that the user can modify as needed. This may also involve the addition of more RHS functions to CLIPS. In sum, embedding CLIPS in the Command_Defense model has proven to be a powerful, easy-to-use, and cost effective choice for the BRAT project.

REFERENCES

[1] Mitchel, Robert R. "Expert Systems and Air Combat Simulation." AI Expert 4(9). (September 1989): 38-43.

[2] Daniel, Robert S., Gossage, Brett N., Barnett, Gene A. "The Battle Management Requirements Analysis Tool Simulation Environment." Presented at the 1989 Summer Computer Simulation Conference, Austin TX. Nichols Research Corporation.

[3] Baker, Louis. Artificial Intelligence With Ada. McGraw-Hill, New York. 1989.

[4] CLIPS Reference Manual. Version 4.3. Houston, Texas; NASA Johnson Space Center. June 1989.

** The opinions expressed in this paper are those of the authors and do not reflect any official position of U.S. Army Strategic Defense Command (USASDC) or Nichols Research Corporation. The work is supported under USASDC contract DASG60-88-C-0069.