N96- 12955

p. 24

Embedding CLIPS-based Expert Systems in a Real-time Object-oriented Simulation

Patrick McConagha and Joseph Reynolds
Tracor Applied Sciences, Inc.
6500 Tracor Lane
Austin, Texas 78725

1.0 INTRODUCTION

This paper describes our continuing work embedding CLIPS-based expert systems into the System Test Environment (STE)[1]. We are embedding simple, compact rule engines in STE to simulate the actions of Naval platform commanders and equipment operators. Our eventual goal is to implement expert system modules that will replace all human participants and some of the equipment present in the simulation.

This paper will briefly describe STE and then discuss its structure and implementation in more detail. Next, we will consider how expert systems could enhance STE's current capabilities. This will be followed by the examination of a specific CLIPS-based expert system model to be embedded in STE. Finally, a summary of our experience and a discussion of anticipated work on this project will close this paper.

2.0 AN OVERVIEW OF STE

So that the reader will understand the environment into which the CLIPS-based expert systems are to be embedded, we will now briefly describe STE. This discussion will be rather short and high-level. A more complete description of STE can be found in [1], from which the following description has been condensed.

[1] Our work on STE was sponsored by Mr. Steve McBurnett of the Integrated Warfare Branch, Code 5570 of the Naval Research Laboratory (NRL) under Contract # N00014-88-C-2175.

STE is not a simulation in itself but rather a simulator. The purpose of STE is to supply data describing the kinematics, equipment, and operation of Naval assets thereby simulating the "real world". This data provides an environment in which to develop and test operational equipment for the Navy. STE can be considered a test bed on which a large range of simulation experiments will be run.

The initial application of STE was to provide data to stimulate a prototype Anti-Submarine Warfare (ASW) decision aid, called TABS, under development at NRL. A typical configuration of STE for testing TABS is shown in Figure 1. Although STE can and will support testing of a range of experimental equipment, work to this point has been directed toward the requirements of TABS. This paper will address applications of expert systems and issues present in this first application of STE.

2.1 STE Structure

The functional requirements imposed on STE were similar to those for any large-scale simulation test bed. These requirements included the following.

- Modularity - STE must readily accept any extensions needed to provide an acceptable environment to the equipment under test. This means STE must be able to generate all data needed to stimulate a piece of equipment and must deliver that data to that equipment as it would receive it in its operational environment.

- Flexibility - Simulation operators must be able to substitute models with various levels of fidelity as required by the equipment under test.

- Speed - STE must run in real time and take advantage of hardware resources available at NRL.

There were other requirements levied on STE, but the three outlined above are all we need to consider. These requirements resulted in an object-oriented design for STE.

STE objects were designed based on the low-level objects in the Object-Oriented Support Library (OOPS) [2]. The following OOPS objects provided the bases from which all STE objects are derived:

- Movable objects - This category includes platforms such as ships, aircraft, torpedoes, etc. as well as other "movable" objects like minefields, storms, convoy perimeters, and land masses. These objects can move and can have equipment objects (see below) attached to them. Land masses do not move, but they are useful as navigation hazards and where land-based forces, such as aircraft, must be considered.

- Equipment objects - This category includes sensors (sonar, radar, etc.), weapons, communications gear, and ship and equipment commanders. Equipment objects are attached to movable objects by the scenario.

- Environment objects - These objects model the operational environments for sonar, radar, etc. as those environments affect the various pieces of equipment.

- Launcher objects - These objects can create new instances of objects as the simulation progresses. For example, a helicopter launcher creates a new helicopter object and attaches to it any radars, sonars, radios, or other equipment objects specified by the scenario.

- Operator objects - These objects serve as translators between STE and entities in the outside world. These entities can be humans sitting at a console or equipment under test.

- Internal Communication objects - This category includes objects used internally by STE to control data exchange and communication between other simulation objects.

- Miscellaneous objects - This category includes low-level objects such as random number generators used by STE to control the simulation.

One of the obvious benefits of an object-oriented design is that although objects share a common structure, they are very much independent. As long as their interfaces conform to what is expected from specific objects, ships for example, implementation of the ship model is wholly contained in the ship object. In fact, two ships in the same scenario could be modeled quite differently. A ship that controls local air traffic could be modeled at a high level of fidelity while another ship that launches helicopters is simply modeled as a movable platform with a helicopter launcher object attached to it. With this in mind, one or more expert systems can be introduced into this structure in place of algorithmic models or in place of models that require human response. We have done this by replacing the specified models with simple embedded CLIPS-based expert systems. Specific applications of expert system models will be discussed in section 3.

2.2 STE Implementation

STE was written in C++, an object-oriented programming language based on C. It runs on a 128 node Butterfly parallel processor with human interfaces implemented on Sun workstations networked with the Butterfly[2]. The current version of STE provides the simulated environment for the initial TABS prototype. It has been able to satisfy the real time speed requirements of TABS, providing data faster than TABS can process it.

2 Sun is a trademark of Sun Microsystems, Inc., Butterfly is a trademark of BBN Advanced Computers, Inc.

3.0 USING CLIPS IN STE

CLIPS-based expert systems will be used to automate decision making in STE. These embedded expert systems will replace models that currently require a response from an operator sitting at a console. In some cases, an embedded expert system could replace an algorithmic model or a table look-up model. Any object in STE whose function can be described by a set of rules, however fuzzy, is a candidate for an embedded expert system.

The benefits gained from this effort include the ability to rapidly develop prototype "experts" for specific STE objects in the CLIPS standalone environment. Enhancements to initial implementations of these experts will likewise be a relatively straightforward task. Similarly, "tweaking" the system by reprogramming experts provides a valuable means of studying various effects of different actions taken under similar situations. These trade-off studies are a major part of STE's functionality. Finally, considering a specific function from a rule-based perspective may lead to insights that help us build better algorithmic models.

Objects in STE that are candidates for an expert system model include the following:

- Platform Commander - A human in command of a ship, airplane, or other platform. A platform commander receives data from equipment on his platform and operational orders from his superiors in the chain of command. He must then determine how to best use his platform and the equipment attached to it to carry out his orders.

- Asset Commander - Examples include a Battle Group, Task Force, or ASW commander. This object differs from a platform commander in that an asset commander issues orders and receives feedback from other commanders. An ASW commander, for example, might have frigates, destroyers, and several ASW aircraft at his disposal. In carrying out his orders, he controls these assets by issuing commands to each of the platforms' commanders.

- Equipment Operators - These commander objects operate specific equipment. For example, a sonar operator receives data from his sonar equipment and reports sonar contacts up the chain of command.
- Specific Functions of Equipment - This is where an embedded expert system replaces a traditionally algorithmic function. The track correlator example in section 4 is an example of this application.

To illustrate the application of embedded expert systems in STE consider the following scenario. A task force is leaving port and steaming to its assigned patrol area. The ASW Commander for the task force is ordered to protect the task force from hostile submarines en route to the patrol area. Assets at his disposal include frigates, destroyers, aircraft, and a variety of equipment on each of these platforms. Figure 2 shows the relationships between some of the STE objects that exist in this scenario. Objects that could possibly be replaced by expert system models are so marked. This example is simplistic but it serves to illustrate the breadth of possible applications of expert systems in STE.

4.0 AN EXPERT SYSTEM MODEL FOR A TRACK CORRELATOR

As our first investigation into expert system applications in STE, we implemented a rudimentary track correlator model. This particular object was chosen mainly because its functionality in STE was well understood. Secondly, the track correlator model in place in STE was a very simple one; almost any new model would have been an improvement.

A typical track correlator is a sequential algorithm that does the following. Given a list of established tracks and a set of new sensor reports, the correlator tries to match each new report to an existing track. A new track is created if a new report doesn't correlate with any of the existing tracks. Finally, existing tracks that do not match new reports are dropped. This process is repeated each time a new set of reports is received.

This is a simplified explanation of a track correlator. Specific issues such as how "closely" a new report must match an existing track, what to do when a new track matches more than one existing track, under what circumstances a new track is created, and how old a track must be before it is dropped vary between applications. Nevertheless, the basic functionality of a track correlator is straightforward.

4.1 The Track Correlator Model

Our initial implementation of an expert system track correlator is shown in Figure 3. This program defines four templates that are used by the expert system. The **sim-time**[3] template defines the fact that maintains the current simulated time and time step. Since STE is an event-driven simulation, the time step is not necessarily a constant value but represents the simulated time that has elapsed since the CLIPS rule engine was last called. The **new-report** template defines the format of facts that contain new sensor reports. A sensor report consists of current information about the sensor itself (e.g.position) and information about the detected target such as bearing. A sensor report can contain much more information about the target, but this information varies between types of sensors (active sonar, passive sonar, radar, etc.). Sensor position is useful when trying to localize the target's position; it was not considered in this example. The **current-track** template defines the facts that identify established tracks. A **current-track** fact contains a contact number and a list of times at which a report was received on this target. The **contact** template defines facts that contain the actual data from each specific sighting of a target. A **contact** fact contains the same information as a **new-report** fact with the exception of sensor position. If sensor position were considered in this model, a **contact** fact would contain an estimate of the target's position derived from the sensor's position and its report on the target.

[3] Boldface words name fact templates, facts, or rules. Fixed-width font words denote function or constant names.

This model contains three rules; one to perform each basic function of a track correlator. The first rule, **extend-track**, tries to correlate a new sensor report with an existing track. This rule compares target information in the new report to information contained in the most recent **contact** fact for a given track. An external function, `same_target`, is called to make the comparison. For this simple model only relative bearing of the target is considered. A higher fidelity test could easily be implemented in `same_target` which would then require more arguments to be passed from CLIPS (report times and target characteristics), but the structure of this rule would be essentially the same.

When this rule fires, the **new-report** fact is removed from the fact list and replaced by a **contact** fact. The outside world is notified of the continuing track via another external function call `same_track`. Finally, the **current-track** fact is modified to incorporate the newest contact with the target.

The second rule, **make-new-track**, creates a new track when a sensor report does not match an existing track. It fires when there does not exist a **contact** fact in the fact list that correlates with the new report. The `same_target` test is used as a predicate function inside a negated pattern to perform this test. As in the **extend-track** rule, the **new-report** fact is replaced by a **contact** fact in the fact list when this rule fires. The outside world is notified of the track creation via a call to the external function `new_track`. Finally, a **current-track** fact is created with a unique track number and asserted. The track number is derived from a track counter fact that is initialized in a `deffact` statement.

The last rule in this model, **lost-track**, fires when no new report is received for an existing track. After **extend-track** and **make-new-track** have fired for each of the extended and new tracks, respectively, **lost-track** simply checks if the most recent contact in an existing track was received before the start of the current CLIPS execution cycle. The **sim-time** fact used in this rule is updated before each execution cycle by the calling program. When this rule fires, it simply reports the loss of contact by calling the external function `no_contact`. Discontinued tracks are not removed from the fact list in this model.

4.2 Running the Track Correlator Model

The 'C' program shown in Figure 4 was used to demonstrate the execution of the expert system track correlator model. The program first opens a data file that contains time and bearing information. Next, it initializes CLIPS, loads the rule base, and resets CLIPS. It then works through the data file building and asserting the **sim-time** fact containing the current simulated time and time step, building and asserting **new-report** facts for each bearing given at the current time (a negative bearing in the data files represents an execution cycle where no new reports are received), runs CLIPS, and retracts the **sim-time** fact. The **sim-time** fact is asserted using the `assert` command so that it may be retracted later. The **new-report** facts are asserted via the more efficient `add_fact` mechanism.

The program listing in Figure 4 also contains the declaration for the external functions called by the track correlator (in `usrfuncs`) and the functions themselves. The `same_target` function simply compares the two parameters and returns `TRUE` if they are within a specified tolerance. Otherwise it returns `FALSE`. The `same_track`, `new_track`, and `no_contact` functions simply print informative messages to the screen.

A sample data file and execution output is shown in Figure 5. Several test data sets were executed to examine the performance of this track correlator model under a wide variety of operating environments. These tests were run on a 20 mHz, 80286-based personal computer. Sample execution times are shown in Tables 1 through 5. Each table shows the time, in seconds required to complete a single iteration of the main loop of the 'C' driver program (see Figure 4). The different number of tracks represent the number of targets being tracked by the system. This value increases as more targets enter the scenario. The maximum number of contacts represents the maximum number of times the system has detected a specific target. This value generally increases as the length of the simulation increases. The number of new reports represents the number of sensor reports received in the current execution cycle. It increases with the number of targets present at the current simulated time.

Not surprisingly, execution time increases with an increase in the number of tracks, contacts, and new reports. While this seems reasonable, the amount of increase was unexpected. Further analysis of the model revealed several improvements which might improve performance.

The **extend-track** rule was relatively straightforward. Maintenance of track information in the fact list was costly. A better implementation might have the `same_track` function update an external database where track histories are stored. The `same_target` test could then access the database to determine track continuity. This would be useful as the need for a more sophisticated correlation test is realized.

The **make-new-track** rule was a little more confusing. The use of a predicate function within a negated pattern circumvented the CLIPS rule that `and` constraints were not allowed inside a negated pattern. This implementation, however, resulted in numerous calls to the `same_target` function. In fact, since the **make-new-track** rule did not limit its correlation attempts to just the most recent **contact** fact for each target, the assertion of a **new-report** fact resulted in a call to `same_target` for each **contact** fact in the fact list. This means that `same_target` was called once for each **current-track** fact and once for each **contact** fact in the fact list each time a **new-report** fact was asserted. With three current tracks consisting of four contacts each and only two new reports, `same_target` would get called seven times when the first report is processed and nine times when the second report is processed (the first report either lengthened an existing track or established a new one).

The initial implementation of the **lost-track** rule was poor. It was activated for every track maintained in the fact list at the beginning of each execution cycle. Because of the salience declaration, activations of **extend-track** fired and removed activations of **lost-track** for those tracks that were extended in the current execution cycle. **lost-track** was modified and the salience declaration was replaced with a (not (new-report)) constraint. Along with minor changes to **extend-track** (retraction of the **new-report** fact was delayed until the track was updated) and the test program (assertion of the new **sim-time** fact was delayed until after all **new-report** facts were asserted), this change ensured that **lost-track** would not be activated unnecessarily. However, this "improvement" actually resulted in slightly LONGER execution times. A seemingly obvious improvement to the model resulted in a degradation of performance.

5.0 CONCLUSIONS

We have successfully implemented a low-fidelity model of a track correlator using CLIPS. This model takes advantages of many of the features CLIPS offers for embedded expert systems. More importantly, the experience gained while working on this model will allow us to design and implement better models for a wide range of functions within STE. We plan to continue our work developing and improving these models. The track correlator we examined in this paper may not ever be used in an STE simulation, but it has demonstrated that simple rule-based models will have a place in the real-time, object-oriented environment of STE.

We have ported CLIPS to a Sun workstation and to the Butterfly computer at NRL. The track correlator model has been run successfully on both. The next major task ahead of us is to modify CLIPS so that multiple expert systems can run concurrently on the Butterfly. From there we can integrate working expert system models into STE.

TABLE 1
Execution times with zero tracks

number of new reports	0	1	2	5
execution time	.02	.05	.06	.17

TABLE 2
Execution times with 1 track

		number of new reports	
		0	1
maximum number of contacts	1	-	.05
	2	-	.05
	3-5	-	.05
	6-10	.03	.06

TABLE 3
Execution times with 2 tracks

		number of new reports		
		0	1	2
maximum number of contacts	1	-	-	.08
	2	-	-	.06
	3-5	-	-	.11
	6-10	-	.09	.13
	15	.04	-	-

TABLE 4
Execution times with 3-5 tracks

maximum number of contacts	number of new reports: 0	1	2	3	5
1	-	-	-	-	.33
2	-	-	-	.11	.38
3-5	.06	.11	.16	.16	.59
6-10	-	-	.28	-	.97
15	-	.18	-	-	-
20	.05	.24	.39	-	-

TABLE 5
Execution times with up to 49 tracks

maximum number of contacts	number of new reports: 2	3	4	5	6	7	8	9	10
35	1.92	2.93	3.73	5.61	7.47	9.50	12.30	15.90	20.80
49	-	-	-	-	13.95	-	-	-	-

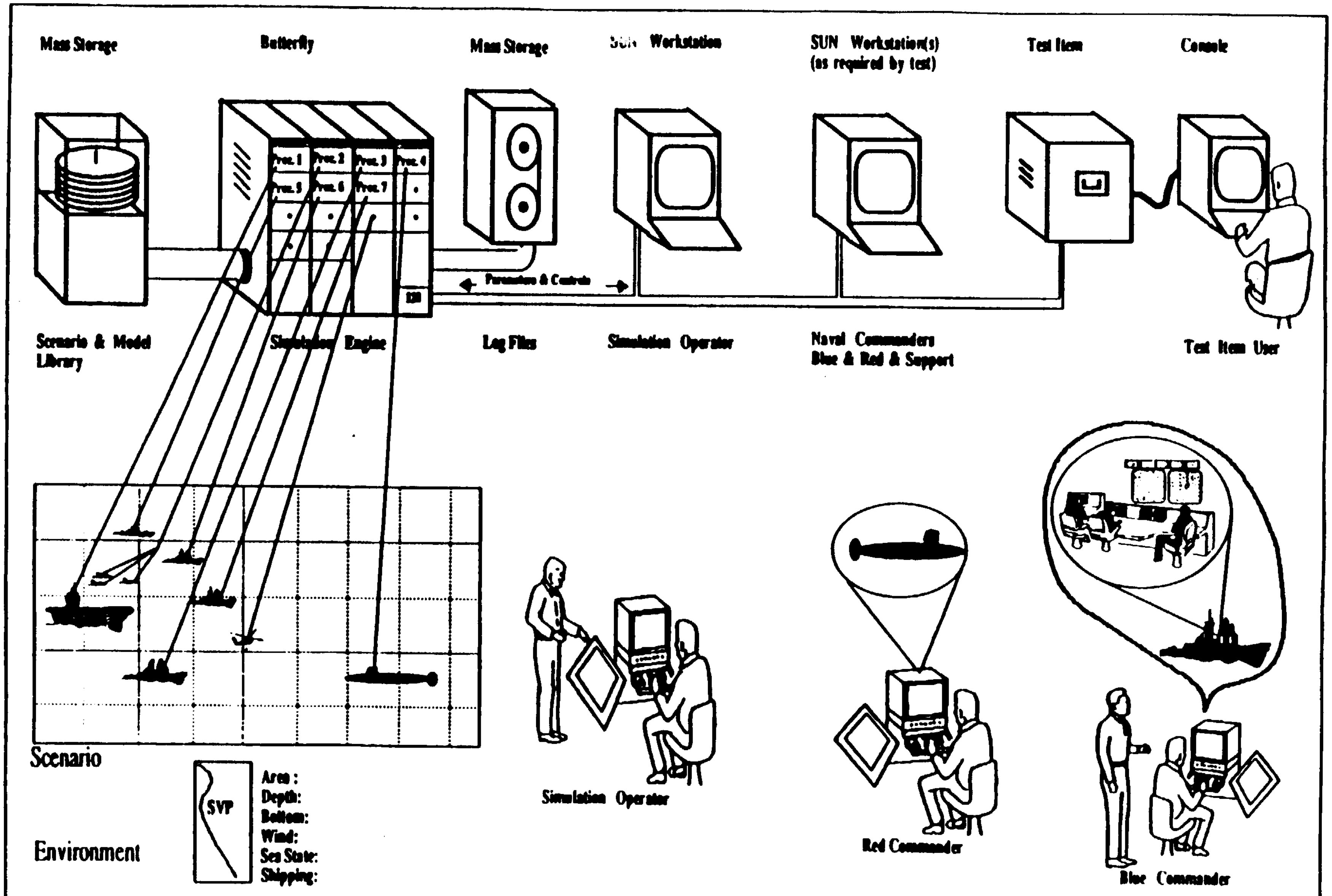
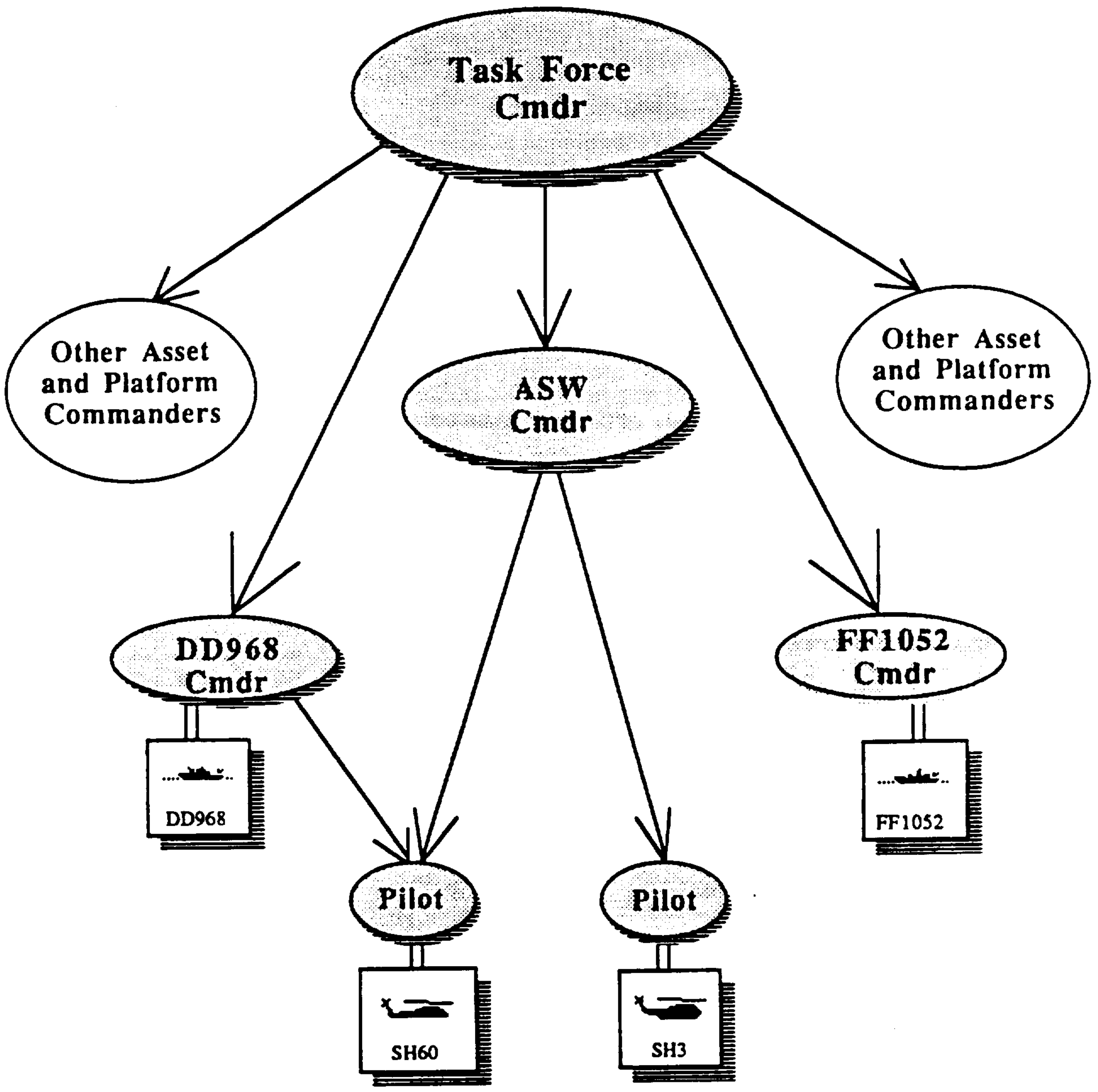


Figure 1. STE Configuration



Shaded objects could be modeled with an Expert System

Figure 2 - A Sample STE Scenario

```
:  File: corlater.clp
:  Programmer: Pat McConagha
;
:  This program implements a simple track correlator that takes
;  new sensor reports and integrates them into a list of
;  current tracks. It will be embedded in an application that
;  calls CLIPS once per execution cycle with new sensor reports.

;
:  The following fact templates are used:
;

(deftemplate sim-time "current simulated time and time step"
  (field cur-time
    (default ?NONE)
    (type NUMBER))
  (field time-step
    (default ?NONE)
    (type NUMBER)))

(deftemplate new-report "a new sensor report"
  (field report-time
    (default ?NONE)
    (type NUMBER))
;  (field sensor-lat          Sensor position not used in this model
;    (default ?NONE)
;    (type NUMBER)
;    (range -90.0 90.0))
;  (field sensor-long
;    (default ?NONE)
;    (type NUMBER)
;    (range -180.0 180.0))
  (field target-bearing
    (default ?NONE)
    (type NUMBER)
    (range 0.0 360.0))
  (multi-field other-info       : Specific target characteristics
    (default ?NONE)             ; dependent on the sensor
    (type ?VARIABLE)))

(deftemplate current-track "track information"
  (field contact-num
    (default ?NONE)
    (type NUMBER))
  (multi-field times             : Times at which contact was made
    (default ?NONE)
    (type NUMBER)))
```

Figure 3 - An Expert System Track Correlator

```
(deftemplate contact "specific information from each contact"
  (field contact-num
    (default ?NONE)
    (type NUMBER))
  (field time
    (default ?NONE)
    (type NUMBER))
  (field target-bearing
    (default ?NONE)
    (type NUMBER)
    (range 0.0 360.0))
  (multi-field other-info        ; Specific target characteristics
    (default ?NONE)              ; dependent on the sensor
    (type ?VARIABLE)))


;
;  Initial facts
;
(deffacts initial-conditions
  (last-track-number 0))


;
; Define the rule for extending an existing track.
; A track is extended if bearings match between a new
; report and an established contact
;

(defrule extend-track
  ?report <- (new-report (report-time ?time)
                         (target-bearing ?bearing)
                         (other-info $?other))
  ?track <- (current-track (contact-num ?num)
                           (times ?last-time $?times))
  (contact (contact-num ?num)
           (time ?last-time)
           (target-bearing ?last-bearing))
  (test (same_target ?bearing ?last-bearing))    ; Simple test
=>                                               ; to match bearings
  (retract ?report)
  (same_track ?num ?bearing ?time)
  (modify ?track (times ?time ?last-time $?times))
  (assert (contact (contact-num ?num)
                   (time ?time)
                   (target-bearing ?bearing)
                   (other-info $?other))))
```

ORIGINAL PAGE IS
OF POOR QUALITY

Figure 3 (Cont'd)

```
;
; Define rule for creating a new track
; A new track is created if a new report does not match the
; bearing of a known track
;

(defrule make-new-track
  ?report <- (new-report (report-time ?time)
                         (target-bearing ?bearing)
                         (other-info $?other))
  (not (contact (target-bearing ?old-bearing&:   ; No known contact
       (same_target ?old-bearing ?bearing))))    ; on new bearing
  ?num <- (last-track-number ?n)
=>
  (retract ?report ?num)
  (bind ?n (+ ?n 1))
  (new_track ?n ?bearing ?time)
  (assert (last-track-number ?n))
  (assert (current-track (contact-num ?n)
                         (times ?time)))
  (assert (contact (contact-num ?n)
                   (time ?time)
                   (target-bearing ?bearing)
                   (other-info $?other))))


;
; Define rule for droping a track
; Don't remove it from fact list, just report that it wasn't detectec
;   during this execution cycle

(defrule lost-track
  (declare (salience -50))
  (current-track (contact-num ?num)
                 (times ?last-time $?))
  (sim-time (cur-time ?t)
            (time-step ?delta-t))
  (test (<= ?last-time (- ?t ?delta-t)))
=>
  (no_contact ?num ?t))
```

Figure 3 (Cont'd)

```
/* File:          main.c
   Programmer:    Pat McConagha

   This program demonstrates a rudimentary expert system
   track correlator implemented in CLIPS.
 */

#include <stdio.h>
#include "clips.h"

#define DATAFILE "contacts.dat"
#define RULESFILE "corlater.clp"

main ()
{
FILE *datafp;
float sim_time, cur_time, brng;
char time_string[50], report_string[50];
struct fact *time_fact, *new_fact;

/*  Both new reports and current track information
    are maintained in the CLIPS fact list.  */

  /* open the data file that contains new reports */
  datafp = fopen(DATAFILE, "r");

  if (datafp == NULL)
      {
      printf("Couldn't open data file.\n");
      exit (1);
      }

  init_clips();
  load_rules(RULESFILE);
  reset_clips();

  fscanf(datafp, "%f%f", &sim_time, &brng);

  cur_time = 0.0;


  /* outer loop iterates through the data file
     calls CLIPS shell once per time interval. */
  while (!feof(datafp))
      {
```

Figure 4 - The 'C' Track Correlator Driver

```
/* build and assert the current time-keeping fact */
sprintf(time_string, "sim-time %f %f", sim_time,
                      sim_time - cur_time);
time_fact = assert(time_string);

cur_time = sim_time;

do
    {

    if (brng >= 0)   /* a negative bearing simulates */
                     /* no new reports during the    */
                     /* current execution cycle      */
        {
        /* build and add a new data fact */
        new_fact = get_el(3);

        add_element(new_fact, 1, WORD, "new-report", 0.0);
        add_element(new_fact, 2, NUMBER, NULL, sim_time);
        add_element(new_fact, 3, NUMBER, NULL, brng);

        if (add_fact(new_fact) == NULL)
             printf("Error adding a data fact.\n");
        }

    fscanf(datafp, "%f%f", &sim_time, &brng);
    }
while ((!feof(datafp)) && (sim_time == cur_time));


run(-1);

retract_fact(time_fact);

printf("\n");
}
```

Figure 4 (Cont'd)

```
/* define functions called from CLIPS */
usrfuncs()
{
int same_target(),
    same_track(),
    new_track(),
    no_contact();

define_function("same_target", 'i', same_target, "same_target");
define_function("same_track", 'v', same_track, "same_track");
define_function("new_track", 'v', new_track, "new_track");
define_function("no_contact", 'v', no_contact, "no_contact");
}


#define epsilon 1.0e-3
int same_target()
{
  float brngl, brng2;
  double fabs();

  brngl = rfloat(1);
  brng2 = rfloat(2);

  if (fabs(brngl-brng2) < epsilon)
     return(TRUE);

  return(FALSE);
}


int same_track()
{

int con_num;
float brng, time;

con_num = rfloat(1);
brng = rfloat(2);
time = rfloat(3);

printf("New report for contact # %3d on
        bearing %5.1f at time %5.1f \n",
        con_num, brng, time);
return(0);
}
```

Figure 4 (Cont'd)

```
int new_track()
{

int con_num;
float brng, time;

con_num = rfloat(1);
brng = rfloat(2);
time = rfloat(3);

printf("Starting new track for contact # %3d on "
        "bearing %5.1f at time %5.1f\n",
         con_num, brng, time);
return(0);
}


int no_contact()
{

int con_num;
float time;

con_num = rfloat(1);
time = rfloat(2);

printf("No report for contact # %3d at time %5.1f\n",
        con_num, time);
return(0);
}
```

Figure 4 (Cont'd)

Program Input

```
1 45
1 195
2 45
2 72
3 195
3 45
3 213
4 72
4 321
4 195
6 45
7 -1
8 72
8 213
```

Program Output

```
Starting new track for contact #   1 on bearing 195.0 at time    1.0
Starting new track for contact #   2 on bearing  45.0 at time    1.0

Starting new track for contact #   3 on bearing  72.0 at time    2.0
New report for contact #   2 on bearing  45.0 at time    2.0
No report for contact #   1 at time   2.0

Starting new track for contact #   4 on bearing 213.0 at time    3.0
New report for contact #   2 on bearing  45.0 at time    3.0
New report for contact #   1 on bearing 195.0 at time    3.0
No report for contact #   3 at time   3.0

New report for contact #   1 on bearing 195.0 at time    4.0
Starting new track for contact #   5 on bearing 321.0 at time    4.0
New report for contact #   3 on bearing  72.0 at time    4.0
No report for contact #   4 at time   4.0
No report for contact #   2 at time   4.0

New report for contact #   2 on bearing  45.0 at time    6.0
No report for contact #   4 at time   6.0
No report for contact #   1 at time   6.0
No report for contact #   5 at time   6.0
No report for contact #   3 at time   6.0

No report for contact #   4 at time   7.0
No report for contact #   1 at time   7.0
No report for contact #   5 at time   7.0
No report for contact #   3 at time   7.0
No report for contact #   2 at time   7.0

New report for contact #   4 on bearing 213.0 at time    8.0
New report for contact #   3 on bearing  72.0 at time    8.0
No report for contact #   1 at time   8.0
No report for contact #   5 at time   8.0
No report for contact #   2 at time   8.0
```

Figure 5 - Execution of a Sample Data File

REFERENCES

1. Cohen, Neil and J. Reynolds. 1990. "System Test Environment: A Real-Time, Man-In-The-Loop Fleet Simulator to Support Testing of Developmental Equipment." In *Proceedings of the SCS Multiconference on Object-Oriented Simulation* (San Diego, CA, Jan. 17-19). Society for Computer Simulation, San Diego, CA, 23-27.

2. Gorlen. Keith, *OOPS*, Public Domain Software Library, National Institutes of Health