1995-113226

N95- 19642

ADDING INTELLIGENT SERVICES TO AN OBJECT ORIENTED SYSTEM.

Bret R. Robideaux and Dr. Theodore A. Metzler

LB&M Associates, Inc.
211 SW A Ave.
Lawton, OK 73505-4051

(405) 355-1471

robidb@lbm.com metzlert@lbm.com

S17-61
34077
P- 10

Abstract

As today's software becomes increasingly complex, the need grows for intelligence of one sort or another to become part of the application- often an intelligence that does not readily fit the paradigm of one's software development.

There are many methods of developing software, but at this time, the most promising is the Object Oriented (OO) method. This method involves an analysis to abstract the problem into separate 'Objects' that are unique in the data that describe them and the behavior that they exhibit, and eventually to convert this analysis into computer code using a programming language that was designed (or retrofitted) for OO implementation.

This paper discusses the creation of three different applications that are analyzed, designed, and programmed using the Shlaer/Mellor method of OO development and C++ as the programming language. All three, however, require the use of an expert system to provide an intelligence that C++ (or any other 'traditional' language) is not directly suited to supply. The flexibility of CLIPS permitted us to make modifications to it that allow seamless integration with any of our applications that require an expert system.

We illustrate this integration with the following applications:
1. An After Action Review (AAR) station that assists a reviewer in watching a simulated tank battle and developing an AAR to critique the performance of the participants in the battle.
2. An embedded training system and over-the-shoulder coach for howitzer crewmen.
3. A system to identify various chemical compounds from their infrared absorption spectra.

Keywords - Object Oriented, CLIPS

INTRODUCTION

The goal of the project was to develop a company methodology of software development. The requirement was to take Object Oriented Analysis (done using the Shlaer-Mellor method) and efficiently convert it to C++ code. The result was a flexible system that supports reusability, portability and extensibility. The heart of this system is a software engine that provides the functionality the Shlaer/Mellor Method (Ref. 2 & 3) allows the analyst to assume is available. This includes message passing (inter-process communication), state machines and timers. CLIPS provides an excellent example of one facet of the engine. Its portability is well-known, and its extensibility allowed us to embed the appropriate portions of our engine into it. We could then modify its behavior to make it pretend it was a natural object or domain of objects. This allowed us to add expert system services to any piece of software developed using our method (and on any platform to which we have ported the engine).

OBJECT ORIENTED METHODS

The principal concept in an OO Method is the object. Many methods include the concept of logically combining groups of objects. The logic behind determining which objects belong to which grouping is as varied as the names used to identify the concept. For example, Booch (Ref. 4) uses categories and Rumbaugh (Ref. 5) uses aggregates. Shlaer/Mellor uses domains (Ref. 3). Objects are grouped into Domains by their functional relationships; that is, their behavior accomplishes complementary tasks.

Fig. 1 shows that the operating system is its own domain. It is possible the operating has very little to do with being OO, but it is useful to model it as part of the system. Another significant domain to note is the Software Architecture Domain. This is the software engine that provides the assumptions made in analysis. The arrows denote the dependence of the domain at the beginning of the arrow upon the domain at the end of the arrow. If the model is built properly, a change in any domain will only affect the domains immediately above it. Notice there are no arrows from the main application (Railroad Operation Domain) to the Operating System Domain. This means a change in operating systems (platforms) should not require any changes in the main application. However, changes will be required in the domains of Trend Recording, User Interface, Software Architecture and Network. The extent of those changes is dependent on the nature of the changes in the operating system. But this does stop the cascade effect of a minor change in the operating system from forcing a complete overhaul of the entire system.

While CLIPS could be considered an object, the definition of domains makes it seems more appropriate to call it a domain.

THE SOFTWARE ENGINE

The portion of the software engine that is pertinent to CLIPS is the message passing. The event structure that is passed is designed to model closely the Shlaer/Mellor event.

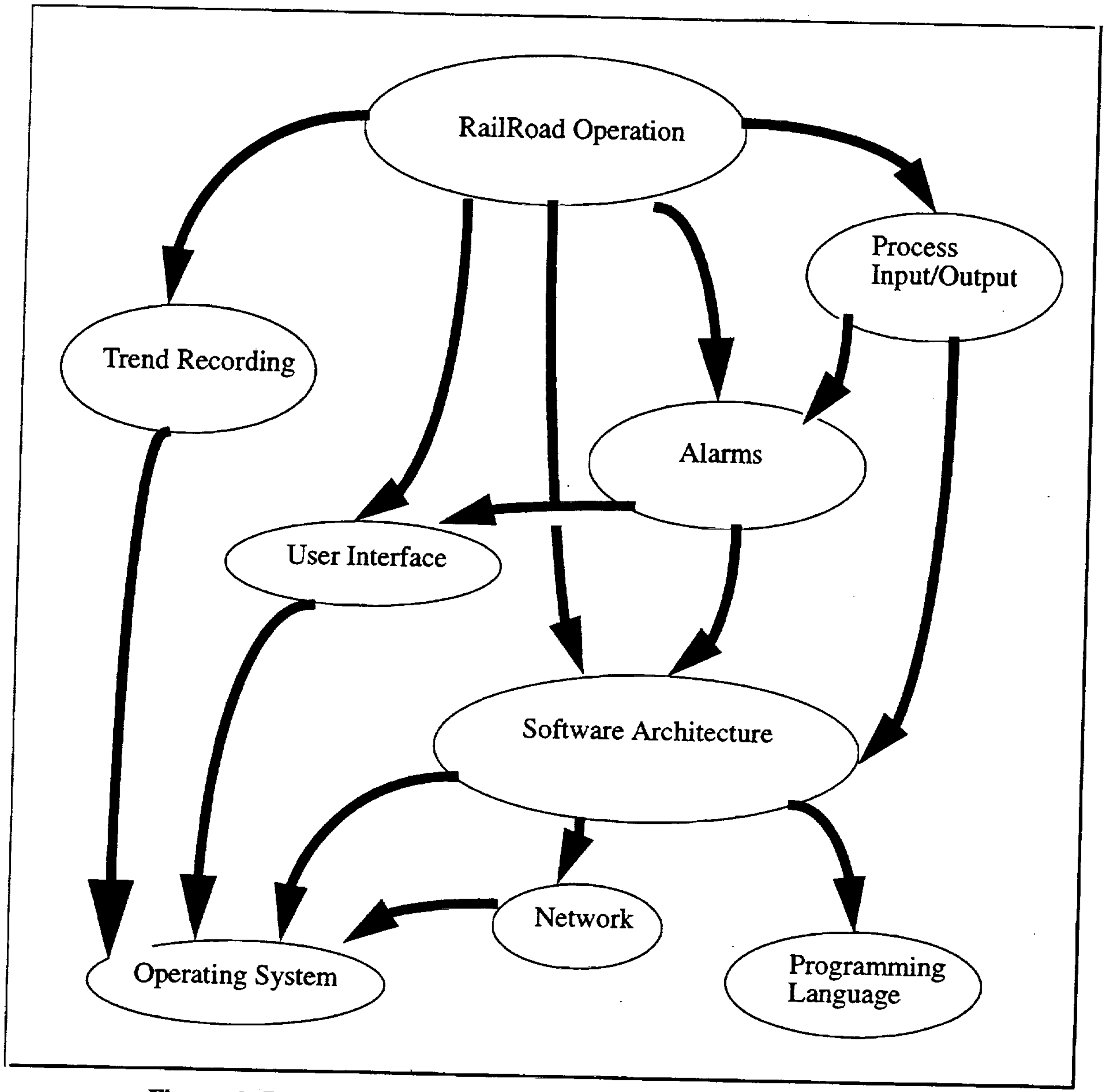

Figure. 1: Domain Chart for the Automated Railroad Management System
Copied from page 141 (Figure 7.4.1) of Ref 3

We have implemented five (5) kinds of events:

1. Standard Events: events passed to an instance of an object to transition the state model
2. Class Events: events dealing with more than one instance of an object
3. Remote Procedure Calls (RPCs): An object may provide functions that other objects can call. One object makes an RPC to another passing some parameters in, having some calculations performed and receiving a response.
4. Accesses: An object may not have direct access to data belonging to another object. The data other objects need must be requested via an accessor.
5. Return Events: the event that carries the result of an RPC or Access event

The event most commonly received by CLIPS is the Class Event. CLIPS is not a true object. Therefore it has no true instances, and Standard Events are not sent to it. CLIPS is more likely to make an RPC than to provide one. It is possible that an object may need to know about a certain fact in the fact base, so receiving an Access is possible. Should CLIPS use an RPC or Access, it will receive a Return Event. RPCs and Accesses are considered priority events so the Return Event could be handled as part of the function that makes the call.

The events themselves have all of the attributes of a Shlaer/Mellor event, plus a few to facilitate functionality. The C++ prototype for the function Send_Mesg is:

```
void Send_Mesg (char* Destination,
                EventTypes Type,
                unsigned long EventNumber,
                char* Data,
                unsigned int DataLength = 0,
                void* InstanceID = 0,
                unsigned long Priority = 0);
```

The engine is currently written on only two different platforms: a Sun 4 and an Amiga. CLIPS, with its modifications, was ported to the Amiga --along with some applications software that was written on the Sun-- and required only a recompile to work. This level of portability means that only the engine needs to be rewritten to move any application from one platform to another. Once the engine is ported, any applications written using the engine (including those with CLIPS embedded in them) need only be recompiled on the new platform and the port is complete. Companies are already building and selling GUI builders that generate multi-platform code so the GUI doesn't need to be rewritten.

The Unix version of the Inter-Process Communication (IPC) part of the engine was written using NASA's Simple Sockets Library. Since all versions of the engine, regardless of the platform, must behave the same way, we have the ability to bring a section of the software down in the middle of a run, modify that section and bring it back up without the rest of the system missing a beat. In the worst case, the system will not crash unrecoverably.

MODIFICATIONS TO CLIPS

Since the software engine is written in C++, at least some part of the CLIPS source code must be converted to C++. The most obvious choice is main.c, but this means the UserFunctions code must be moved elsewhere since they must remain written in C. I moved them to sysdep.c.

The input loop has been moved to main.c because all input will come in over the IPC as valid CLIPS instructions. The stdin input in CommandLoop has been removed, and the function is no longer a loop. The base function added to CLIPS is:

(RETURN event_number destination data)

This function takes its three parameters and adds them to a linked list of structures (see Fig. 2). When CommandLoop has completed its pass and returns control to main.c, it checks the linked list and performs the IPC requests the CLIPS code made (see Fig. 3). RETURN handles the easiest of the possible event types that can be sent out: Class Events. To handle Standard Events, some way of managing the instance handles of the destination(s) needs to be implemented. To handle Accesses and RPCs some method of handling the Return events needs to be implemented. The easiest way is to write a manager program. Use *RETURN* to send messages to the Manager. The event number and possibly part of the data can be used to instruct the Manager on what needs to be done.

```
RETURN ()
  {
    extract parameters from call

    allocate a new element for the list

    fill in the structure

    add structure to the list

  }
```

Figure. 2: Pseudocode for function RETURN

```
main ()
  {
    init CLIPS

    begin loop

      Get Message

      set command string

      call CommandLoop

      check output list

      if there are elements in the structure
        make the appropriate send messages

    end loop
  }
```

Figure 3: Pseudocode for main ()

Sending instructions and information into CLIPS requires a translator to takes Shlaer/Mellor events and convert them to CLIPS valid instructions. In Shlaer/Mellor terms this construct is called a Bridge.

With these changes CLIPS can pretend it is a natural part of our system.

EXAMPLES

This system has been successfully used with three different projects. An After Action Review Station for reviewing a simulated tank battle, an Embedded Training System to aid howitzer crewmen in the performance of their job and a chemical classifier based on a chemical's infrared absorbtion spectra.

The After Action Review (AAR) station is called the Automated Training Analysis Feedback System or ATAFS. See Figure 4 for its domain chart. Its job is to watch a simulated tank battle on a network and aid the reviewer in developing an AAR in a timely manner. It does so by feeding information about the locations and activities of the vehicles and other pertinent data about the exercise to the expert system. The expert system watches the situation and marks certain events in the exercise as potentially important. ATAFS will then use this information to automatically generate a skeleton AAR and provide tools for the reviewer to customize the AAR to highlight the events that really were important.

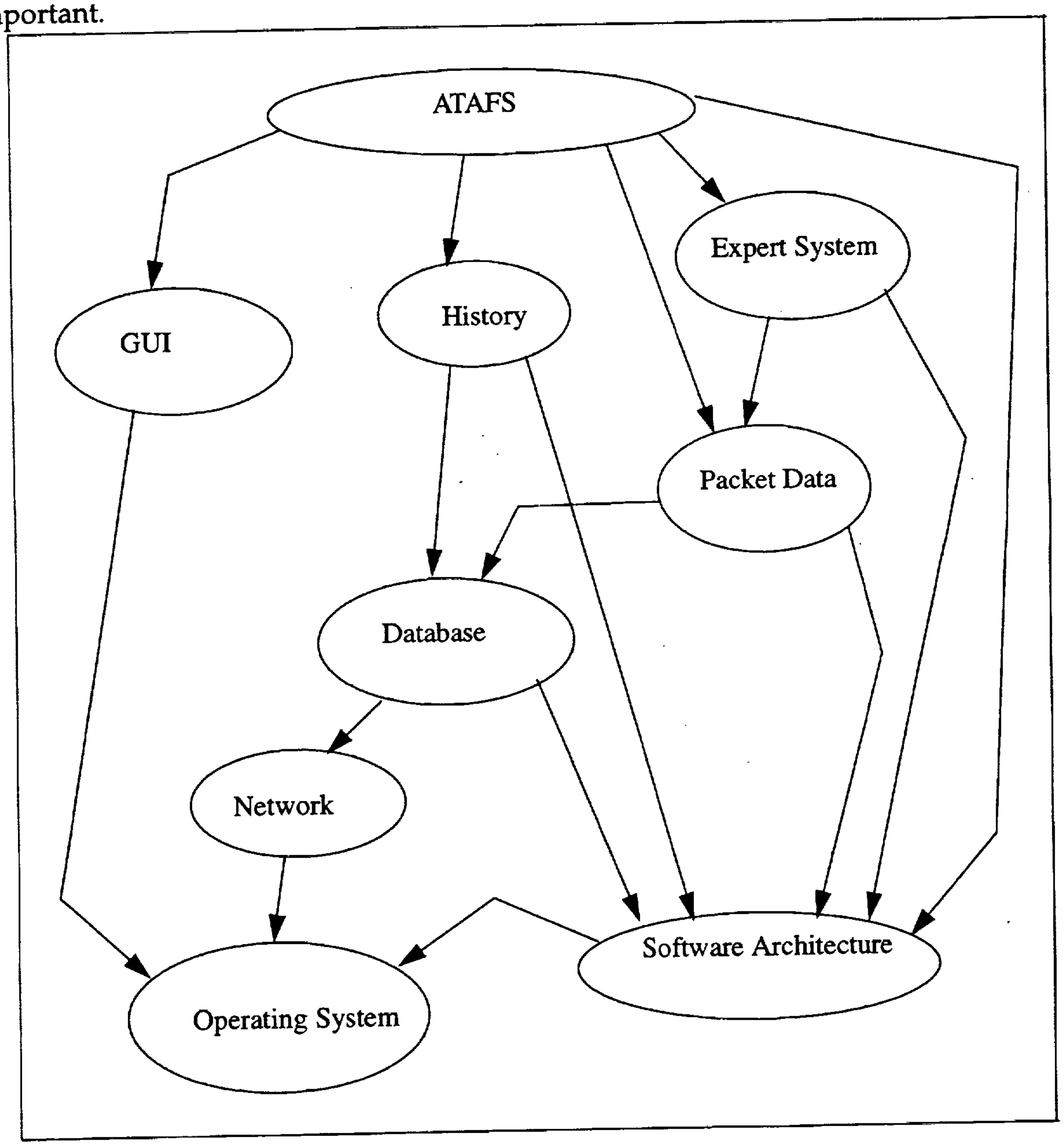

Fig. 4: ATAFS Domain Chart

In an early version of the system, ATAFS was given the location of a line-of-departure, a phase-line and an objective. During the simulated exercise of four vehicles moving to the objective, CLIPS was fed the current location of each vehicle. CLIPS updated each vehicle's position in its fact base and compared the new location with the previous location in relation to each of the control markings. When it found a vehicle on the opposite side of a control mark than it previously was, it *RETURN*ed an appropriate event to ATAFS. Upon being notified that a potentially interesting event had just occurred, ATAFS is left to do what ever needs to be done. CLIPS continues to watch for other events it is programmed to deem interesting.

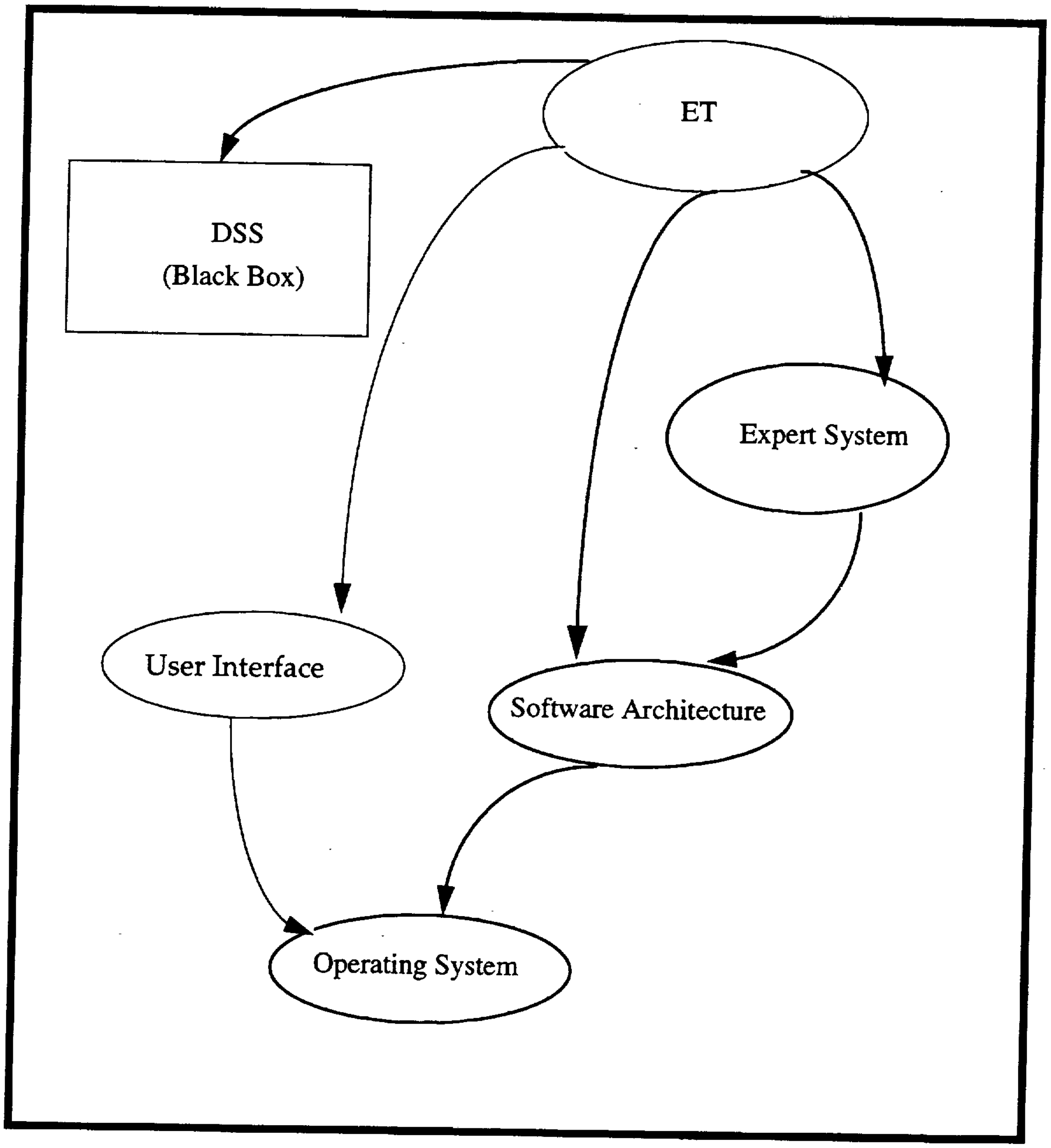

Fig 5: ET Domain Chart

Embedded Training (ET) is a coaching and training system for howitzer crewmen. See Figure 5 for its domain chart. It provides everything from full courseware to simple reminders/checklists. It also provides a scenario facility that emulates real world situations and includes an expert system that watches the trainee attempt to perform the required tasks. During the scenario the expert system monitors the trainee's activities and compares them to the activities that the scenario requires.

It adjusts the user's score according to his actions and uses his current score to predict whether the user will need help on a particular task. All of the user's actions on the terminal are passed along to CLIPS, which interprets them and determines what the he is currently doing and attempting to do. Having determined what the user is doing, CLIPS uses its *RETURN* in conjunction with a manager program to access data contained in various objects. This data is used by CLIPS to determine what level of help, if any, to issue the user. The decision is then *RETURN*ed to ET which supplies the specified help.

The chemical analysis system (Ref. 6) designed by Dr. Metzler and chemist Dr. Gore takes the infrared (IR) spectrum of a chemical compound and attempts to classify it using a trained neural net. In the course of their study, they determined the neural net performed significantly better when some sort of pre-processor was able to narrow down the identification of the compound. CLIPS was one of the methods selected. The IR spectrum was broken down into 52 bins. The value of each of these bins was abstracted into values of strong, medium and weak. Initially three rules were built: one to identify ethers, one for esters and one for aldehydes (Figure 6).

```
(defrule ester
   (bin 15 is strong)
   (or (bin 24 is strong)
       (or (bin 25 is strong)
            (bin 26 is strong))
       (and (or (bin 22 is strong)
                 (bin 23 is strong)
                 (bin 24 is strong))
            (or (bin 25 is strong)
                 (bin 26 is strong)
                 (bin 27 is strong)))
        (and (bin 28 is medium)
             (or (bin 22 is strong)
                 (bin 23 is strong)
                 (bin 24 is strong))))
=>
  (RETURN 1 "IR_Classifier" "compound is an ester"))
```

Figure 6. Example of a Chemical Rule

The results of this pre-processing were *RETURN*ed to the neural net which then was able to choose a module specific to the chosen functional group. Later, the output from another pre-processor (a nearest neighbor classifier) was also input into CLIPS and rules were added to resolve differences between CLIPS' initial choice and the nearest-neighbor's choice.

CONCLUSIONS

CLIPS proved to be a very useful tool when used in this way. In ET and ATAFS, the Expert System Domain could easily be expanded to use a hybrid system similar to that of the Chemical Analysis problem, or use multiple copies of CLIPS operating independently or coordinating their efforts using some kind of Blackboard. In many applications, intelligence is not the main concern. User interface concerns in both ET and ATAFS were more important than the intelligence. ET has the added burden of operating in conjuction with a black box system (software written by a partner company). ATAFS' biggest concern was capturing every packet off of the network where the exercise was occurring and storing them in an easily accessible manner. The flexibility of C++ is better able to handle these tasks than CLIPS, but not nearly as well suited for representing an expert coaching and grading a trainee howitzer crewman or reviewing a tank battle. While recognizing individual chemical compounds would be tedious task for CLIPS (and the programmer), recognizing the functional group the compound belongs to and passing the information along to a trained neural net is almost trivial.

Expert systems like CLIPS have both strengths and weaknesses, as do virtually all other methods of developing software. Object Oriented is becoming the most popular way to develop software, but it still has its shortfalls. Neural nets are gaining in popularity as the way to do Artificial Intelligence, especially in the media, but they too have their limits. By mixing different methods and technologies, modifying and using existing software to interact with each other, software can be pushed to solving greater and greater problems.

REFERENCES

1. Gary Riley, et al, CLIPS Reference Manual, Version 5.1 of CLIPS, Volume III Utilities and Interfaces Guide, Lyndon B. Johnson Space Center (September 10, 1991).

2. Sally Shlaer and Stephen Mellor, Object-Oriented Systems Analysis, Modeling the World in Data. (P T R Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1988).

3. Sally Shlaer and Stephen Mellor, Object Lifecycles, Modeling the World in States. (Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1992).

4. Grady Booch, Object Oriented Design with Applications. (The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991).

5. James Rumbaugh, et al, Object-Oriented Modeling and Design. (Prentice-Hall, Inc., Englewood Cliffs, 1991).

6. T.A. Metzler and R.H. Gore, Application of Hybrid AI to Identification of Chemical Compounds, Proceedings of TECOM Conference on AI and the Environment, Aberdeen Maryland, September 13-16, 1994