PCLIPS FINAL REPORT

Patrick D. Krolak

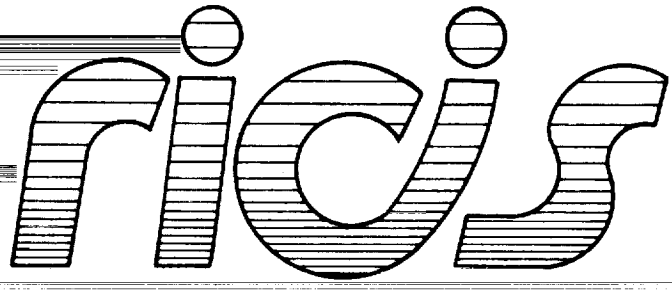
Center for Productivity Enhancement

University of Lowell

December 1990

Cooperative Agreement NCC 9-16
Research Activity AI.10

NASA Johnson Space Center
Mission Support Directorate
Mission Planning and Analysis Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

PCLIPS FINAL REPORT

Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by Dr. Patrick D. Krolak, Director of the Center for Productivity Enhancement, the University of Lowell, Lowell, Massachusetts. Dr. Terry Feagin served as RICIS research representative.

Funding has been provided by Mission Planning and Analysis Division, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Timothy Cleghorn, of the Mission Planning and Analysis Division, Mission Support Directorate, NASA/JSC.

PCLIPS FINAL REPORT

Center for Productivity Enhancement
University of Lowell

Subcontract #17

Introduction:

Expert system technology was developed to take over the role of decision making in areas where human experts are unavailable. One of the problems with a large expert system is that it requires a large, very high speed, expensive processor to attain a reasonable response time. This becomes even more critical in a real-time environment, such as a continous process plant, where slow reaction to a situation can have serious, even deadly consequences. It also creates a situation where a single point failure could have disastrous results. In a lot of cases, one large expert system can be broken down into multiple smaller experts, each responsible for a small piece of the problem. These, in turn, can be distributed over a network of smaller, cheaper processors. Not only can a significant savings be realized, but redundancy can be built into this system to minimize downtime.

CLIPS is an expert system shell developed by NASA, created specifically to allow rapid implementation of an expert system. Unlike most modern expert systems shells, CLIPS is written in C, not LISP, and thus needs a very small amount of memory to run. Parallel CLIPS (PCLIPS) is an extension to CLIPS. It is intended to be used in situations where a group of expert systems are expected to run simultaneously and occasionally communicate with each other on an integrated network. In keeping with the original concepts of CLIPS, PCLIPS also minimizes the amount of memory needed to run, thereby maintaining the idea of using micro-processors for handling PCLIPS experts.

Another goal of the PCLIPS development was to minimize the learning curve which always exists for a programmer learning distributed/parallel programming for the first time. Under most distributed programming environments, the programmer must become familiar with operating system specific details, such as process creation, inter-process communication, and network protocols. PCLIPS provides an operating system independent programming environment. It frees the programmer from having to learn about operating system specific functions, and allows the programmer to concentrate on the expert system control.

PCLIPS allows cooperating expert systems that are solving different problems to exchange data when appropriate. For example, if we have an automated factory and many co-operating expert systems, we want expert systems that control robots, do scheduling, planning, monitoring and prognosis, implement security and inventory, and perform many other roles in a factory environment to be able to talk to each other. They do not need to be completely dependent upon each other in the sense of fine-grained parallelism. In other words, we did not want an expert system which depends upon another expert system for its input, and will block and wait until it receives that input. We have very coarse-grained parallelism with this shell which allows cooperating systems to receive data knowledge that is pertinent to the functions they are performing. We did not want to take an expert system, divide it up into four parts, and run it on four processors, although it is possible to do this in certain cases.

PCLIPS is a coarse-grained data distribution system. Its main goal is to take infor-

mation in one knowledge base and distribute it to other knowledge bases so that all the executing expert systems are able to use that knowledge to solve their disparate problems. For example, in an automated manufacturing environment, there is normally a scheduling package responsible for handing out assignments to the various manufacturing entities. This scheduling package will occasionally need to get information regarding work orders, which would come from a forecast planning system, or from an order entry system. It will also need to get information regarding resource utilization in order to optimize the resources. Each manufacturing workcell can also be controlled by an expert system, to maximize throughput, and to handle errors that occur without requiring operator intervention.

Implementation:

In order to minimize the amount of training required to bring a CLIPS programmer up to speed with PCLIPS, the calling techniques of CLIPS were maintained. In a CLIPS program, when the expert system wants to place a new fact onto its fact base, it uses the function *assert* to perform that task. In PCLIPS, when the expert system wants to place a new fact onto the fact base of all the other expert systems on the network, it uses the function *rassert,* which provides the broadcast mechanism that sends the fact to every other expert system on the network. The expert system programmer does not have to worry about network protocols, operating system calls, or any other system specific programming.

Up to this point, we support four different protocols, and are researching several others.

The first protocol which PCLIPS has been implemented under is the Network Computing System (NCS). NCS is the Open Software Foundation (OSF) Remote Procedure Call (RPC) standard which was developed by Apollo (now HP/Apollo). NCS is a transport-independant, object oriented RPC protocol which allows the programmer to create and manipulate objects in remote server-modules without having to manipulate low-level communications. PCLIPS also takes advantage of NCS's "location brokering", which is achieved through the use of NCS's Global Location Broker (GLB). Using this resource PCLIPS need never know what machines other PCLIPS' are located on, as NCS maintains dynamic locationdatabases. Currently, PCLIPS uses NCS to transmit facts over TCP/IP or DDS networks, but can easily be modified to support many other network models. This implementation has become the standard, by which the functionality of any new ports will be copied. When an expert starts up, it firsts initializes by registering with

The second protocol that we implemented was VMS mailboxes. This implementation allows us to run multiple PCLIPS processes on a VAX running VMS, which was needed for a project that had been developed under VMS. Under this implementation, when a user starts a PCLIPS process, the first thing the process does is check to see if a PCLIPS server is running on the node. If a server is not running, one is started up. Then the process registers with the server. First, the process creates a link to the server's mailbox. It then sends a mail message to the server that states the logical name for the registering process. The server uses the logical name to create a link back to the registering process's mailbox. When another

PCLIPS process starts up, it also registers with the server. Then, whenever a fact is RASSERTED by one process, it goes to the server, which forwards the message to all the other registered PCLIPS processes. We have since received a copy of NCS for VMS, and are in the progress of porting PCLIPS over.

The third protocol we developed used the message ports on the Commodore Amiga. This approach allowed us to develop PCLIPS applications on a very inexpensive microprocessor, which was used for prototyping user interfaces. This implementation is similar to the VMS port, since the message ports perform tasks similar to VMS mailboxes. A server must be started on the system first, then any expert that starts up after it registers with the server.

The final protocol that we have ported PCLIPS to is the Intel Personal SuperComputer (IPSC) communication primitives on the Intel Hypercube. The Hypercube was designed by combining multiple low-cost microprocessors into a high-speed parallel processing machine. Since it has a fairly low external bandwidth, it is ideal for situations in which a highly CPU-intensive problem can be divided into smaller problems that can be solved in parallel. The Hypercube architecture is designed so that the message passing distance between the two furthest processors in the machine is minimized. Each processor in the cube runs an executive, which allows it to not only pass messages between processors, but to run multiple programs on each processor. Under PCLIPS, whenever one process remotely asserts a fact, it sends it to all its adjacent processors. They copy it and pass it along to their adjacent processors until all the processors have received the message.

Some of the other protocols that we are researching are:

GOSIP - Government OSI Protocol

FDDI - Fiber Distributed Data Protocol

TCP/IP - Tranmission Control Protocol/Internet Protocol

SNMP - Small Network Management Protocol

Information contained in a PCLIPS message is as follows; the fact is prepended with the processor name and process number or PID (process identifier) that the fact came from. This information will be useful if network security becomes an issue. There will also be two times stamps added to the PCLIPS message. One time stamp will be added when the expert system rasserts the fact. The second time stamp will be added when the fact is actually sent from the machine. The need for the two separate times exists because of the potential delay between the rasserting of the fact, and the actual transmission of the fact. The following scenario parameters define an example of how this delay could occur.

1) The network protocol that is being used requires handshaking between the sender and receiver everytime a message is sent.

2) There are a large number of expert systems on the network.

Under these conditions, when an expert system rasserts a fact, it must send out a message to one of the expert systems on the network, and wait for the corresponding handshake message that is returned before it can transmit the fact to any other expert system. Therefore, it could take several seconds, or longer for the transmitting expert system to send its fact to all the other expert systems on the network. If any form of information aging or network management is implemented, the time delay must be noted.

Finally, a zone identifier is added, which enables the implementation of multiple zones or groupings of expert systems. This information is not under the control of the user, it is inside the CLIPS kernel and cannot be modified. This allows us to write another very important part of a network of expert systems cooperating with each other, which is a security expert system that keeps track of everything and makes sure that no expert system is doing something that it should not be doing. Another use is that it allows us to do is to write a diagnositic expert system. The expert system is related to the security expert system, and its purpose is not to find expert systems creating bad knowledge or changing correct knowledge, but to diagnose expert systems that are failing in some way. It is likely that the security and diagnostic systems would be one and the same on smaller networks. Another purpose of broadcasting is that voting expert systems can be easily created, where three identical expert systems are running simultaneously, and a fourth expert system tabulates the voting results.

Applications:

One area of research that PCLIPS will be useful in is the area of network system management. The state of the art in network system management is exemplified by Digital Equipment Corporation's SysMan Utility on VMS. One system is dedicated to this problem electronically and uses a network it is attached to to go out and diagnose all of its systems. This is a stopgap measure at best because while it works fine for smaller networks (50-100 systems), it cannot handle a 500-1,000 system network unless there is a very large processor dedicated to running systems diagnostics.

In most scenarios that occur today, the system manager acts as a reactionary problem solver, as opposed to a proactive problem solver. To correct this situation, we are developing a system that uses PCLIPS processes for monitoring and solving network problems. Since PCLIPS is capable of talking to all the other expert systems on the network. It is best described as a "daemon" in Unix terminology. With roughly one minute cycle time, it activates and exectues a series of subroutines in C, Fortran, or any other language, requesting data or facts about a certain aspect of the system. For example, the expert system might activate and ask, "Have there been any disk errors?" The answer is "no", so it continues. It then asks, "Have there been any tape drive problems?", and the tape drive subroutine reports "No". Then, it continues, "Have there been any terminal errors, TGY line errors?", and the system returns and says, "Yes". Then PCLIPS asks, "Which one?" The system returns, "TGY-5". What PCLIPS does next depends on the complexity of the expert system one has built. The current implementation simply reports the problem.

How is the problem reported? One system runs a control expert system as well as a

monitor expert system. The control expert system collects data from the entire network and stores it. This differs from the state of the art in that each system monitors itself. Also, because we are using C, a system manager can define his own error detection functions and errors on his system. PCLIPS then sends a message to the control expert system saying a TGY line has failed. The control expert system stores that information until the system manager retrieves it and analyzes the data. PCLIPS then continues because it cannot do anything about the broken TGY line. It asks, "Has the password file changes?" The system looks at a known copy of the password file, which is guaranteed to be correct by some secure method, and the system reports, "Yes, the password has been changed." Depending on the level of the expert system, we might simply report a security error at this point, or we might look at the user who authorized the change and potentially check that information against another data structure somewhere on the network that defines what users were in fact authorized by the system manager today. Since it is not always possible to guarantee security at one point, levels of security or multiple checks and balances exist to guarantee that one's security is correct. In this case, the fact that a new account was put on the system must appear, or an alarm to the system manager is made.

A final example is a pure software error. How can PCLIPS deal with this? With a large network, there will often be machines connected to other networks. Often the gateways fail, or are no longer recognized by the host system. Usually this causes losses of electronic mail, along with other problems. When this situation occurs, the first thing to do is to check if the gateway is working. Assuming that it is, very often the tables of available gateways and their current states have become corrupt. Depending upon the operating system, especially with most Unix operating systems, the gateway tables become corrupt quite easily. This is especially true if a user runs the routing deamon to determine the available gateways and their routes from Berkley. What a system manager can do with our current system is define a template of what the available gateways should look like, and compare that template against the actual existing gateways. If they do not match, PCLIPS detects the error when it queries whether the routing tables are correct or not. The C subroutine says, "No", and CLIPS reports the error to the control expert system. Then, it tells the control expert system, "I'm attempting the fix the problem", and then calls the C subroutine to fix the problem. After it calls the C subroutine, it asks once again, "Are the routing tables correct?" If the answer is "Yes", the expert system reports that it successfully fixed the problem. If the answer is "No", the expert system reports that it failed to correct the problem. Thus, in the case of a software error, we have allowed our expert system to use its knowledge base of rules to find and diagnose a problem and solve it without human intervention.

The first aspect of this system was that it had the capability of diagnosing hardware by asking the kernel on that system if there were any hardware problems. We have seen that it can also diagnose operating system software. The final possibility is to put a run-time parallel CLIPS into an application that must run constantly. When that application is experiencing problems, the parallel CLIPS can assert the information that it is having problems to the other parallel CLIPS in the network. The locations that should receive that information are the monitor expert system on the local node, and expert systems that depend on the application that is having problems, and the control expert system.

Another area in which PCLIPS could provide valuable assistance is in the area of the Space Station environmental control. Since the Space Station is targeted to consist of a number of different, complex systems, expert system technology would be utilized to monitor and maintain the individual systems. For instance, there would be a power grid expert that would keep track of power consumption, battery capacity, and solar panel input, among others. There would be an environmental expert that would keep track of air quality, to ensure that safety of the occupants. There would be experts in charge of monitoring experiments, and many others. In order to provide the astronauts with a safe environment, many different factors must be continously monitored and controlled. For example, one of the most important is the amount of oxygen in the atmosphere of the Space Station. Since the humans will be continously using oxygen, and generating carbon dioxide, the oxygen level must be monitored. A filtering and retrieval system will provide the normal equilibrium. But, in the event that an unnormally low level of oxygen were reached, due to a clogged filter, the air monitoring expert system would decide that an emergency oxygen pump should resupply the air. However, since the pump requires electricity to run, the power grid expert would have to be notified that a certain number of watts would be required to run the pump. Since the power grid expert has been monitoring consumption, it will be able to evaluate whether or not the power system can handle the new load. If it determines that it cannot, without exceeding certain thresholds, it would determine that it would have to cut back on consumption in other areas. One area it might cut back on could be an experiment area. The power grid expert would evaluate the consumption rates of the different experiments, looking to cut back the amount it needs to stay below its usage threshold. It would then double check with the experiment expert to verify that the experiment could be shutdown termporarily without any lasting side effects. Upon receiving verification of this, it would cut the power to the experiment in question, and energize the oxygen pump. Since each of these experts were running on their own microprocessors, they would be able to quickly come up with the necessary solutions, and combine to solve the emergency situation in a matter of seconds.

Conclusions:

The preliminary goals of PCLIPS has been achieved. The first goal was to maintain the ease of programming an expert system. Since the original CLIPS provided that capability, we wanted that capability to continue when developing parallel expert systems. Secondly, we wanted to isolate the expert system programmer from the complexities of network programming. This was accomplished by developing a standard library of functions that were callable form PCLIPS. These functions will exist under any implementation of PCLIPS, regardless of the networking or inter-process communication protocol. This will also make PCLIPS applications highly portable. An application that is written using the TCP/IP implementation of PCLIPS will also run, without any changes to the CLIPS code, on the NCS implementation.

With the advent of the low cost, high speed microprocessor, and high speed computer networks, it is now possible to develop a highly sophisticated set of expert systems using PCLIPS that can perform better than large expert systems running on a large, expensive mainframe.

Integrating PCLIPS into ULowell's Lincoln Logs Factory of the Future

The Center for Productivity Enhancement
University of Lowell

by
Brenda J. McGee
Mark D. Miller
Dr. Patrick Krolak
Stanley J. Barr

ABSTRACT

We are attempting to show how independent but cooperating expert systems, executing within a parallel production system (PCLIPS), can operate and control a completely automated, fault tolerant prototype of a factory of the future (The Lincoln Logs Factory of the Future). The factory consists of a CAD system for designing the Lincoln Log Houses, two workcells, and a materials handling system. A workcell consists of two robots, parts feeders, and a frame mounted vision system.

1. INTRODUCTION

The University of Lowell's Factory of the Future, consists of an intelligent Computer Aided Design (CAD) system, a graphical simulator, and a physical factory. Designed to be autonomous; needing minimal assistance from an operator, the factory is a state of the art prototype for automated manufacturing. This factory consists of two physical workcells, which are connected by a computer controlled material handling system. Each workcell has two robots, vertically mounted cameras which are controlled by a vision system, and parts feeders which have sensors to monitor workcell inventory. The CAD system provides the user interface for designing the houses. The design is sent to a CLIPS scheduling expert system. Thereafter other CLIPS expert systems, aided by the vision system, operate and synchronize the robots and other hardware to manufacture the design. For efficient execution of these parallel expert systems there is a need for a fast, reliable, user-transparent, hardware and operating system independent networking production system. PCLIPS (parallel CLIPS)[1], developed at the Center for Productivity Enhancement, has these qualities allowing concurrent independent CLIPS expert systems to exchange messages in the form of facts. The crucial feature of PCLIPS is a command called *rassert* or *remote assert*. *Rassert* allows a CLIPS process to assert facts into the fact databases of every other CLIPS process, thus communicating cooperatively with one another, ultimately resulting in an intelligent manufacturing workcell environment.

2. PCLIPS and Lincoln Logs: The Concept

Interprocess communication for Lincoln Logs was originally accomplished through a mailbox system, implemented on VMS[1]. Each process in the factory created its own mailbox, and a pointer to the mailbox of any other process that it needed to talk to. This reserved space in memory where messages were left and picked up, using QIOs. This method had two limitations. The first was that it was system dependent. It would only work on VAXEN[2]. The other limitation was the incompatibility between our interprocess messages and CLIPS, which we were implementing at the process level. PCLIPS was chosen, therefore, to replace this mailbox system.

PCLIPS has several advantages. The network operations and protocol requirements for the network are transparent to the user, thus eliminating that concern from the expert system developer. It also works on heterogeneous computer systems, enabling the expert system developer to design platform independent software. Finally, the inter-process messages are in the native format of CLIPS (facts), thus eliminating the earlier need for translating inter-process messages ino facts.

The first issue that we had to resolve was a standard format for interprocess messages since the use of the *rassert (remote assert)* command globally broadcasts each fact, or interprocess message, to every other process running PCLIPS. We used the following format:

(IPM receiver sender $?)

The atom *receiver* is the name of the process who the message is intended for. This is either the specific name of the process (ex. VISION), or the string ALL. An IPM fact with ALL in the receiver position is a message intended for all processes running.

The atom *sender* is the name of the process which broadcasted the fact. When an inter-process message is broadcast, each process picks up the fact and fires a rule in order to test whether or not that fact is meant for that process. Code from the Vision process will serve as an example, as the code in each process is simillar.

```
(defrule interprocess_message
        ?gnim <- (get_next_int_message)
        ?IPM <- (IPM VISION|ALL ?sender ?rm1 ?rm2 ?rm3 ?rm4 ?rm5 ?rm6 ?rm7)
=>
        (retract ?IPM ?gnim)
        (assert (rmessage ?sender ?rm1 ?rm2 ?rm3 ?rm4 ?rm5 ?rm6 ?rm7))
)
```

[1] VMS is a trademark of Digital Equipment Corporation

[2] VAXEN is a trademark of Digital Equipment Corporation

If the fact is not meant for that particular process, a rule is fired that retracts that fact from the list.

```
(defrule IPM_not_for_this_process
        ?IPM <- (IPM ~VISION&~ALL ?sender $?)
=>
        (retract ?IPM)
)
```

Since all the processes are event triggered, there are times when a single process will complete all its current tasks, and will have to wait until a new event occurs. In order to avoid a busy wait, we took advantage of the *salience* option in CLIPS and created a rule that suspends a process until a new event occurs. Since we used the lowest salience possible, this rule will only fire when there is nothing else on the agenda, thus eliminating the possibility of the process being suspended in the middle of a task. When all the rules have fired, whether or not the IPM was for that process, the process goes back into a wait state until the next global fact arrives.

```
(defrule wait
        (declare (salience -10000))
        ?w <- (wait for IPM)
=>
        (retract ?w)
        (call (suspend))
)
```

CLIPS has also been integrated into the factory of the future in the decision making process.

3.1 Preventer (Collision Prevention)

At this time, our collision prevention algorithm allows us to use two robots in a workspace. The Preventer process performs collision prevention by calculating where each robot arm, gripper and part will be located during placement. A robot requests access to the workspace, through an *rasserted* fact. The Preventer then calculates the path the robot will follow to get to it's destination, and determines the potential for a collision or obstruction between any of the following: The two arms, the parts in the robot grippers, and the vision inspection system. The vision system needs a clear view of the part it is inspecting. Otherwise, it may report invalid information.

If the Preventer determines that a collision is possible, it will enforce mutual exclusion to the workspace by delaying *rasserting* the *access granted* fact to the Robot Process until the situation has changed, and the robot has a clear path to its destination.

3.2 Vision (Vision Inspection)

Vision Inspection, done with an overhead camera, occurs after a robot has successfully placed a piece on the work pallet. The Vision system waits for an *rasserted fact* from the Robot process. The fact contains information about the part that needs inspection, namely the part type, it's location, and orientation on the pallet. If the Vision System does not approve of the part's position, it alerts the robot with a fact that includes the calculated offset of the part. When alerted, the robot will re-enter the workspace and attempt to correct the problem. Once the Vision system approves a part, the robot moves on to its next tast.

3.3 Robot (Robotic Control)

We have created a Robot Planner using CLIPS. When the planner, or process starts up, it *rasserts* a task request to the workcell scheduler. When the scheduler returns the task message, the planner breaks the task down into a series of operations. The example we will follow is a Place Part task.

First, the planner must determine the part's location (in the parts feeder, on the jig, on the pallet, etc.). Based upon this information, it then determines its approach path to the object. Once it has the part in its grasp, and the gripper is clear of the part holder, a path to the workspace is calculated. At this point, the robot process must request access to the workspace, which it does by *rasserting* the request to the Preventer process. Once the robot has been given clearance, it calculates a path to the release point, follows the path, and releases the part. It then moves clear of the workspace, and *rasserts* a request for a vision inspection. If the vision system reports the part placement to be outside the tolerance limits, the robot will re-enter the workspace and attempt to correct the error. When the vision system approves the part, the robot sends a task completion fact to the scheduler. It then checks its agenda for any other work. If none exists, it sends another task request message to the scheduler.

The flow of the planner is controlled by two facts, *state* and *action*. When the planner enters a particular state, there are several actions which must be performed sequentially to assure a correct execution. There are several examples of built-in error handling. Whenever an error occurs, the planner will immediately move to the error handler. We force this to occur through the use of a high salience for the error handler initiator.

```
(defrule first-grasp-error-handler
        (declare (salience 100))
        (error occurred)
        (state get-part)
        ?action <- (action grasp-part first-attempt)
=>
        (retract ?action)
         (assert (action grasp-part second-attempt))
)
```

3.4 Sensors (Sensor Fusion)

The Sensor process allows the operator to be informed when there is a change of state in the parts feeder; as well as allowing the operator to shutdown a particular feeder. This control is accomplished by monitoring infra-red sensors near the base of each feeder. The Sensor process continuously monitors these sensors, and *rasserts* facts to the scheduler if a state change occurs. The Sensor process also has the ability to introduce errors into the system in order to test the system's ability to cope with malfunctions.

3.5 Scheduler (Task Scheduler)

The Scheduler Expert System is a dynamic task optimizer. The scheduler reads in a natural language description of the house. After parsing the description, the scheduler dynamically assigns tasks to the requesting Robot Processes. Due to the dynamic nature of the scheduler, it can change the schedule as workcell conditions change, enabling it to track workcell inventory, throughput, and resources. The Scheduler's main goal is to maximize the workcell yield. It achieves this goal by optimizing workcell events to allow parallel execution of robot operations. When mutual exclusion is enforced, one of the robots must wait for the other robot to exit the work space, cutting down on throughput.

3.6 IO-Process (Interprocess IO-controller)

The IO-Process is the parent of all workcell processes. It allows the operator to configure the workcell for the resources available (i.e. material handling system, vision, robots, simulator, etc.) It then starts up the workcell process and remotely asserts a startup fact in each. Afterwards, it monitors all the workcell processes and notifies each workcell process of changing resources. When the job is finished, the IO-Process terminates all workcell processes by *rasserting* a shutdown message.

3.7 Simulator (Workcell Simulator)

The Workcell Simulator provides a mechanism for testing control software without the need for workcell hardware. The Simulator graphically mimics the actions of both robots on a color workstation. While the Simulator is running, the Robot Processes simply redirect their output to the Simulator instead of the physical robots. The Simulator provides handshaking capabilities similar to the physical robots, which allows the operator to simulate a robot error, for testing the reliability of the workcell software.

3.8 Material Handling (Automated Materials Handling System)

The system loads and unloads work pallets into each workcell. It also has the ability to transport pallets from one workcell to another for completion of jobs, if the need arises. Error detecting and handling capabilities have been built into the expert system which controls the MHS. If there is an error, it can determine exactly what the problem is.

3.9 Pod (Pod Scheduler)

The Pod Scheduler is the middle man between the factory scheduler and the individual workcell processes. It not only gives assignments to individual workcells, but also controls the overall execution of workcells that are performing similar tasks. When, the Pod scheduler receives a

build request from the Factory Scheduler, it determines which workcell should take on the responsibility of carrying out the request. If the chosen workcell is unable to carry out this request for some reason, it will then choose another workcell to take over the job. There is also a materials handling system at the Pod level that is under the control of the Pod. This setup enables movement of the pallets among the workcells at the Pod Level.

4. Future Directions

The Lincoln Logs Factory of the Future will continue implementing improved versions of PCLIPS as they are developed. One limitation of the current version of PCLIPS is its lack of routing capabilities for remotely asserted facts. Every *rasserted* fact is broadcasted to every other process running PCLIPS. As our factory grows, and subsequently the number of processes running PCLIPS, routing mechanism will have to be implemented to avoid network and CPU saturation. We will also continue the development of our process level expert systems, with a focus on designing and implementing an advisory framework to provide operator, advisor and supervisor assistance at every level of the factory.

5. REFERENCES

[1] Miller, Ross, "PCLIPS: A Distributed Expert System Environment," First CLIPS Users Group Conference, Houston, Texas, August 1990.

[2] Alpha II Reference Guide. MICROBOT Inc. Mountain View, CA. January 1984.

[3] CLIPS Reference Manual. Mission Support Directorate, NASA/Johnson Space Center. Houston, Texas. Version 4.2, April 1988.

[4] RAIL Standard Vision Documentation Package. (AI Part #510-500610). AUTOMATIX Inc., Billerica, MA. March 1987

[5] Kosta, C.P., Wilkens, L., and Miller, M., "A Three-Dimensional C.A.D. system". Center for Productivity Enhancement, University of Lowell. Working Paper #FOF-87-101. Lowell, MA. February 1988.

[6] Miller, M., "Multiple Robot Scheduling". Center for Productivity Enhancement, University of Lowell. Working Paper #FOF-87-103. Lowell, MA. November 1987.

[7] Miller, M., Kosta, C. and Krolak, Dr. P., "Computer Assisted Robotic Assembly" 3rd International Conference on CAD/CAM, Robotics, & Factories of the Future.

[8] Dean, Thomas L., "Intractability and Time-Dependent Planning" 'Reasoning about Actions & Plans, Proceedings of the 1986 Workshop', Morgan Kaufmann, Los Altos, California.

[9] Dougherty, Edward R., Giardina, Charles R., 'Mathematical Methods for Artificial Intelligence and Autonomous Systems' Prentice Hall, Englewood Cliffs, New Jersey. 1988.

A Neural Network Simulation Package in CLIPS

Himanshu Bhatnagar, Patrick D. Krolak, Brenda J. McGee, John Coleman.

Center for Productivity Enhancement, University of Lowell, Lowell, Ma. 01853

ABSTRACT

The intrinsic similarity between the firing of a rule and the firing of a neuron has been captured, in this research, to provide a neural network development system within an existing production system (CLIPS). A very important by-product of this research has been the emergence of an integrated technique of using rule based systems in conjunction with the neural networks to solve complex problems. The system provides a tool kit for an integrated use of the two techniques and is also extendible to accommodate other AI techniques like the semantic networks, connectionist networks, and even the petri nets. This integrated technique can be very useful in solving complex AI problems.

1. INTRODUCTION

Direct hardware implementation of Neural Networks is not always easy and hence there is a need for simulating them through computer software. Early examples of software simulation models can be found in [1] and [2]. These and the other simulation models primarily simulate the neural states, neural architectures and connection strengths, and implement the tools to manipulate them. Several learning techniques (rules) have been proposed in the Neural Network literature, one of them being the generalized delta rule (or Back Propagation)[3]. Our first level goal is to provide a more efficient package, in CLIPS, for simulating neural networks employing back propagation, together with expert systems.

CLIPS is an expert system shell developed by NASA [4], which provides a LISP like interface and allows both forward and backward chaining. The production rules, under forward chaining, have facts on the lhs and action commands on the rhs. When facts, in the facts database, match the lhs of any rule that rule fires, possibly causing assertion of more facts and hence firing of other rules. In a binary neural network, a neuron fires when its activation has exceeded its threshold value. There is an inherent similarity in the way rules fire in an expert system and the way neurons fire in a Neural Network, suggesting the modeling of one in terms of the other, and hence CLIPS can prove to be a very effective simulation tool for Neural Network modeling. We, at the Center for Productivity Enhancement, University of Lowell, have developed a shell called Neural CLIPS, or N-CLIPS which allows Neural Network Simulations to be built, tested and implemented along with regular expert systems. N-CLIPS provides a common environment for development, implementation and operation of two competing and radically different artificial intelligence techniques : the C Language Integrated Production System (CLIPS) for writing expert systems and a Neural Network system. These systems can either operate independently to solve different classes of artificial intelligence problems or can cooperate to help solve much bigger AI problems [9]. In [6] Rabelo has shown the usefulness of combining the neural networks and the expert systems. Knowledge representation, acquisition and manipulation, decision making and decision support are the major characteristics of these techniques and hence when they are used together they can share knowledge and can share the decision making process itself.

To further emphasize the importance of such a common platform we are using it to model a traffic control system for mobile robots operating the Material Handling System of a Flexible Manufacturing System based factory [7]. The (simulated) mobile robots have on-board neural networks which work together with expert system modules to guide them through the factory floor without collisions and with minimum delays. Since CLIPS provides an excellent interface with C, these expert system rules can interact with other processes and also interact with different types of peripheral hardware [5].

The next section provides a brief description of the terms relevant to neural networks, followed by a survey of the features common to currently available simulation packages. The need for integrating AI techniques is discussed next followed by a description of N-CLIPS. The last section gives a detailed explanation of the system developed.

2. ARTIFICIAL NEURAL NETWORKS

2.1 Definitions

For our purposes a **neural network** is a densely connected, possibly layered, network of simple processing units (**neurons**). The connections, known as **synapses,** are weighted links between two such units where the **weight** of a link is modifiable, and determines what fraction of the signal, between the two units, is actually passed. A negative weight usually signifies an **inhibitory link**(synapse) which causes an inhibitory effect on the firing of a post-synaptic neuron. A positive weight usually signifies a **excitatory link** which excites the neuron to which it is connected.

Neurons, in the network may be classified into three types depending on the roles they play. They are either **input neurons** (input layer), **output neurons** (output layer) or **hidden neuron** (hidden layer) depending on whether they accept input from outside world, provide an output to the outside world or receive input from units within the system and generate output for the units within the system. Processing within a neuron may be divided into three stages : a) determination of net input to the neuron ; b) determination of neural state (an **activation function** associated with a neuron determines the state); and c) determination of the neural output (an **output function** determines the final output value).

2.2 Learning

The two major learning paradigms available currently are: generalized delta rule (GDR) or back propagation [3] and its variations for both feed forward and **recurrent networks**[16], and **hebbian learning**, with its sophisticated variants (by which we mean to include methods employed in Bi-directional Associative memories and other associative memory models) [10][17][18][19] [20].

2.3 Generalized Delta Rule

In the initial phase of our work we have focused on the GDR as applied to feed forward networks. In this approach a set of patterns is repeatedly presented at the input layer of a multilayered network. The output pattern generated is compared with a target pattern. The difference is propagated back and is reflected as a change in the weights of the links, all the while minimizing a global energy function (**mean squared error** function). The difference or the **delta** is

used to modify the weights of links between neurons. This process is repeated till the actual pattern is within a close range of the target pattern, for a particular input pattern. This is done for each input pattern.

3. EXISTING SIMULATION PACKAGES

A brief survey of most of the commercial neural network simulation and development packages reveals the following characteristics :

* A strong user-interface : Pop-up menus within a windowing environment, a file system and interface with major database systems for I/O.
* Types of Learning Paradigms supported : All major learning paradigms along with their variations.
* Capability for Customizing and designing user-specified Neural Nets : Ranges from just setting up of network parameters to script based design of neural networks.
* Debugging & Interaction tools : On-line graphical editing of a neural network; pausing, restarting and saving snap shots of neural nets during different states of their operation: displaying weight change, delta change, noise and a host of other features.

The different information processing paradigms are particularly well suited for the problem domain in which they evolved. However, when addressing classes of problems that span more than one domain an integrated approach seems attractive. This approach involves several different AI techniques. The inter-relationships of these techniques is still not well understood and there is a need to study their interaction with each other. None of the systems available today have the capability of providing a common platform to investigate these 'inter-relationships'. In N-CLIPS we provide a common playing ground for at least two of these, with the capability of extensions to accommodate others.

4. WHY CLIPS ?

By extending CLIPS to accommodate neural networks, semantic networks, connectionist networks and other knowledge representation techniques, we, will have a tool to understand their complex inter-relationships and the mapping of one technique into another. In real life systems we need the precision of expert systems, the localized representation of semantic networks and the flexibility of neural networks all encompassed into one. This is so because each of these techniques have strengths which compliment the weaknesses of the other. The brittleness of expert systems can be supplemented with the plasticity of neural networks on one hand and the lack of precision of neural networks can be substituted by precise rules and facts. Adding new knowledge to an expert system is quick (as a new rule) but its interaction with the existing rules can be of a conflicting nature. On the other hand adding a new pattern to a neural network takes a long time but can be made to interfere minimally with the old patterns. On a factory floor, new situations can be quickly learned by plugging in temporary rules. However, over a period of time, these rules get to be unmanageable and redundant and have to be trimmed. They can be collectively mapped into a neural network which could iron out the conflicting rules, and once trained it can be mapped back to a more parsimonious set of rules. To illustrate this further, assume a set of rules which do not trigger each other. The combinatorial arrangement of the

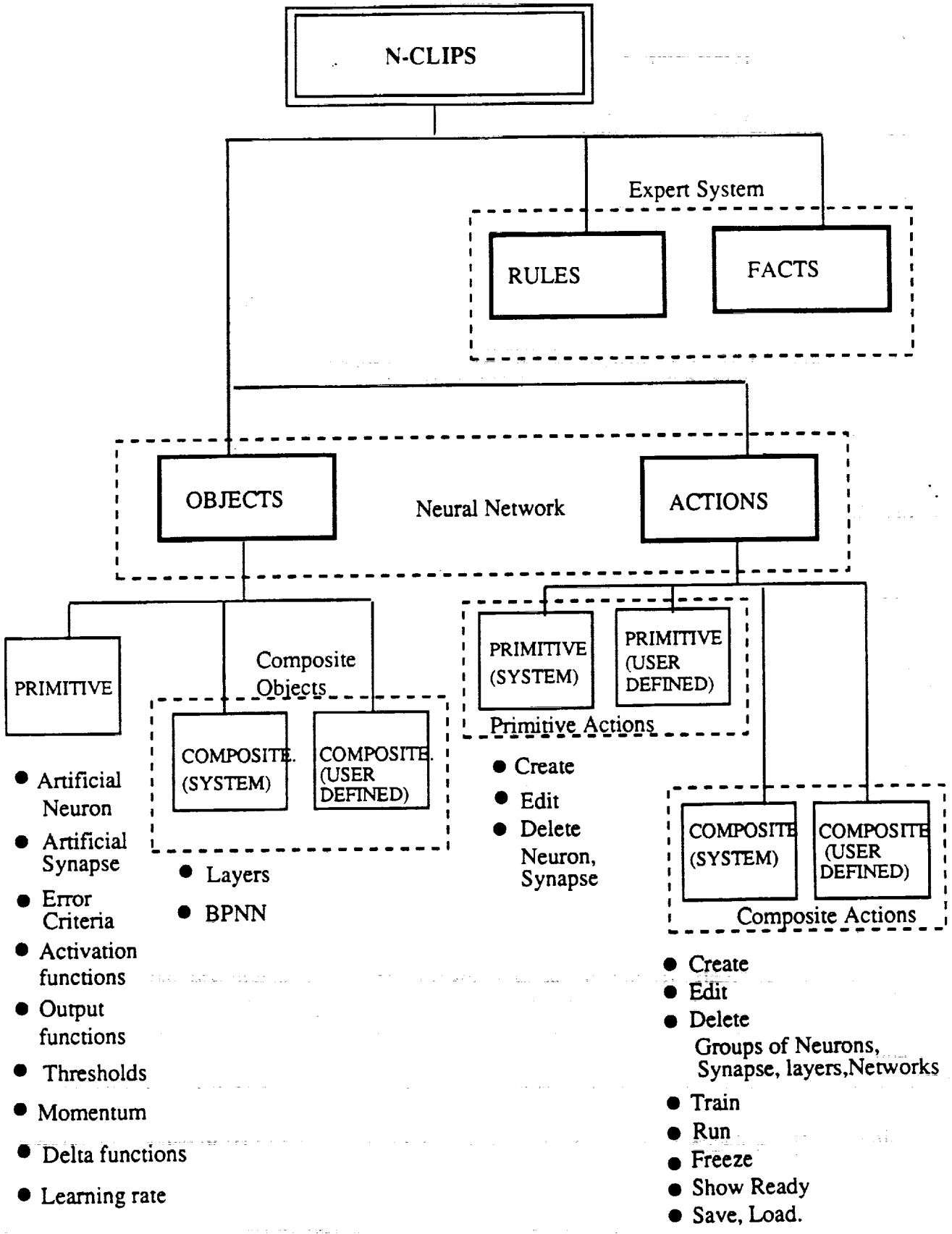


fig 1 : N-CLIPS : A hierarchical description

union of facts on the lhs of these rules and the actions on the rhs can be translated to the input and the output patterns of a back propagation neural network (BPNN). Out of the available output patterns the ones actually needed can be selected without difficulty. Then by applying the inverse mapping technique proposed by Williams [11] where the input values (at the input layer) instead of the weights are modified via back propagation of error, the neural network can be converted back into an expert systems. Of course, a major problem to be considered in this process is that of knowledge representation since patterns must be translated into facts. In addition there may be many-to-one mappings that are dependent upon initial states of the system.

Sometimes, at a higher level of design the localized representation of a problem can be done through semantic networks and the rest as expert systems and neural networks . For example the higher level path planning of mobile robots on a factory floor can be done using semantic networks, while the low level path planning and traffic control can be done by expert systems which in turn depend on neural networks for decision support. As can be seen all three models will need to communicate with each other. CLIPS allows that via rules and facts, moreso because all of these techniques shall have rules and facts as their building blocks.

Another example would be the cooperative use of multiple neural networks for mortgage underwriting and industrial parts Inspections [13][14][15]. In [13] the system is a collection of nine coupled sub-networks have three sub-networks acting as 'experts' and their cooperative effect helps in validating the confidence level of the decisions made by the whole system.

The major functions which were added to the existing CLIPS code have been briefly explained in Appendix A. The engine for neural networks manipulates its own data structure but eventually uses clips' agenda and fact lists to let the clips execute the neural network. The functions listed in the appendix are driver, nassert, add_nfact, ncompare, ndrive, nretract. PCLIPS [8], a distributed version of CLIPS has also been developed at the university.

5. N-CLIPS

This shell provides an object oriented approach to problem solving in the neural network and fuzzy logic domain and at the same time maintains the integrity of the CLIPS production system. The expert systems and sub-systems can be written as rules and facts while a neural network is represented as a collection of objects and a set of actions to be performed on them. It provides well known neural network learning paradigms as objects which the user can use to map their problems onto or use them as subsystems of more complex user-designed neural networks. Users can also build their own variations of the existing paradigms and can also create their own learning rules and models within the given environment. A library of functions for creating and editing neural network objects like neurons, synapses, activation functions and layers is made available to the user. The *ntrain* and *nrun* functions are a collection of rules linked with facts which can be invoked to train a neural network or execute it. The rules and facts making up the expert systems are written in the same way as in regular CLIPS. At the lowest level of expert system-neural network communication the two systems interact via rules and facts. However, at a higher level, complex but abstract interaction is possible. For example the neural network actions, composite and primitive, can be written as a set of rules linked with facts while an expert system can spawn off a neural network to extract useful information from available fuzzy or smudged knowledge. This system can also be used as a first level tutor for

understanding basic existing models. CLIPSs' capability to interface with other languages viz. C, Ada is exploited for a graphical (X-Windows and/or Motif) user-interface and a file-system interface for saving snap shots and networks themselves. In this system the following graphics user-interface is available :

* Neural Network interconnection diagram.
* CLIPS rules interconnection diagram for seeing which rules fire which other rules and on what basis.
* Mouse interface with the Neural Network diagram.
* 'Click-on-connection-for-weight-change' graphical facility.
* Change of color if a node fires.
* X-Windows link editor.
* X-Windows weight editor.

The file system interface allows saving and loading of neural networks via save_nn() and load_nn() functions, at any instance.

6. SYSTEM DESCRIPTION

6.1 OBJECTS (Primitive)

6.1.1 Artificial Neuron

An artificial neuron is basically of three types i.e. Input, Output and Hidden. Its major characteristics (for back propagation) are an identifying number, layer number, an activation and output function, threshold value and its type. These parameters could be either passed to a C function call or through a template invoked from the CLIPS interpreter. After the parameters of a neuron are accepted from the user they are encoded as a special rule in a string which is then compiled and loaded into the network. These parameters could be edited and a complete neuron deleted at any given instance. Internally in CLIPS the specifications of a neuron are also stored within a data structure (see fig. 2). Any modification of a neuron's specifications are automatically reflected in the data structures and the associated rule. A deleted neuron will also result in deletion of all the connected links.

The composition of the special rule (for back propagation only) is as follows :

```
(defrule artneu#
   ? neu <- (neuron # layer # ready to fire)
=>
      (nretract ? neu)
      (propagate layer #)
      (calculate_delta layer #)
      (change_weights from layer # to layer #)
)
```

On the rhs the function propagate(), propagates the output signal to the next layer neurons after duly multiplying it by the strength of the connection of the links. The next function calculate_delta(), calculates the deltas based on the error signal propagated by the succeeding layer and stores them in the data structures. Finally, the change_weights() function changes weights based on the calculated deltas. These functions manipulate the network data structure (fig. 2)

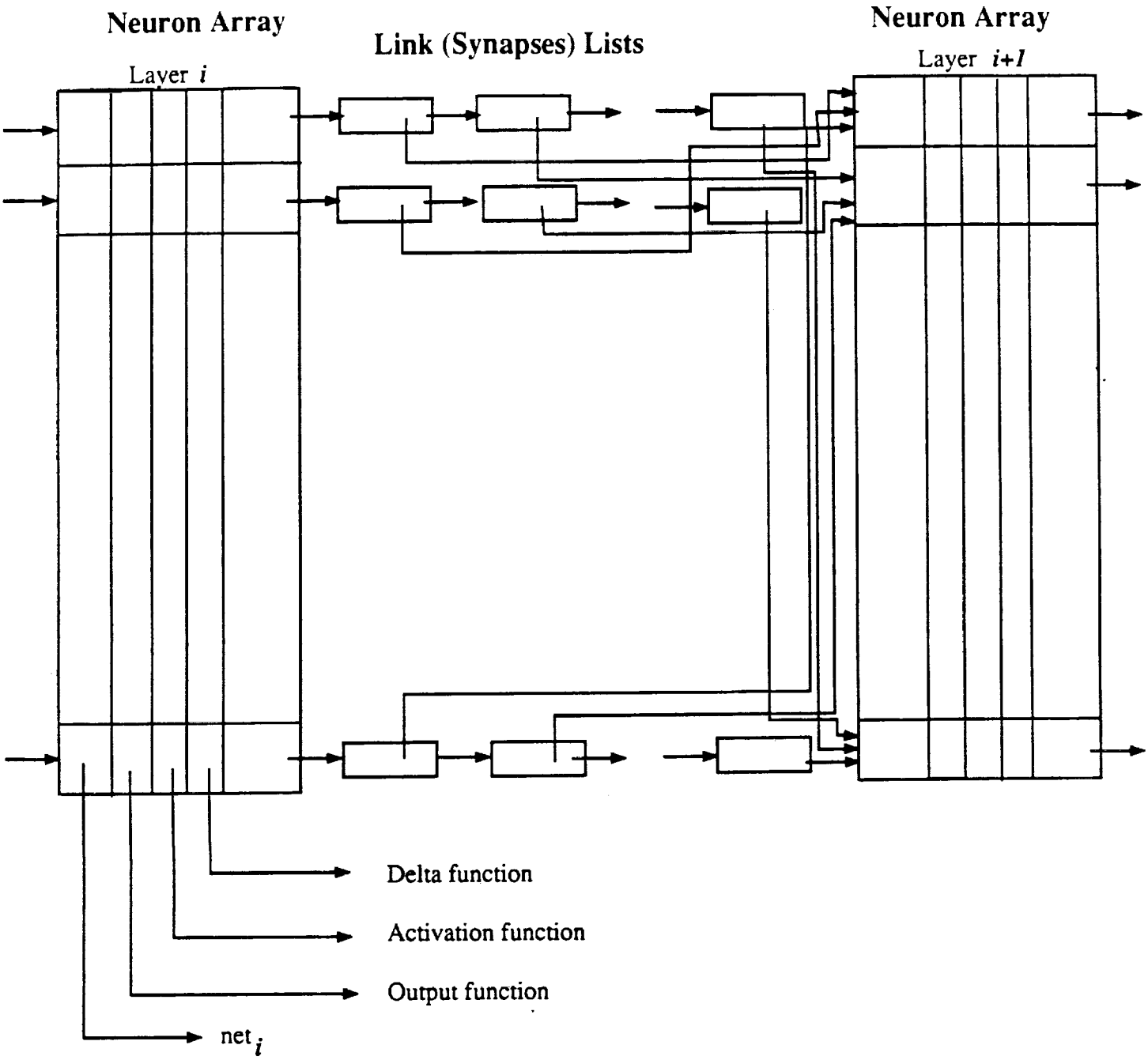


fig. 2 : A sample Data structure for storing a Neural Network in N-CLIPS

for performing the above mentioned functions. This neuron is specifically suited for representing the hidden layers of a feed forward neural network. The rules for input and output layer neurons are slightly different. These special rules can be modified via functions provided in the system to represent any other kind of neural network model. A more generalized model of a neuron is in design.

6.1.2 Artificial Synapse

These are the links between neurons, and are mainly characterized by the following parameters: 'from' and 'to' neuron # and layer #, the type (in or out link), weight. They are stored in a special data structure (see fig. 2) and can also be stored as facts; as in the case of the outgoing links from the output layer neurons. They can be created/edited and deleted as individual links or as a group (from one layer to another). Individual links can be created as C functions or from within CLIPS interpreter (a template possibly from within a windowing system) and group links can be created through a X windows graphics link map editor (explained later). This way fractional (percentage of total neurons) connectivity between layers can be represented very easily.

6.1.3 Activation functions

A library of different existing activation functions is provided to which a user can add a function or modify or delete a function. These functions can be selectively applied to individual neurons or to a group of neurons.

6.1.4 Input/Output functions

Different input/output functions, for neurons in the input/output layers, which are currently popular are provided in a library. The user can add, modify or delete a function from the library. The user can select a function from this library to apply to a single neuron or to a group of them. The input function is usually a linear function, nevertheless a different input function can also be provided. Also for single layer feature maps [10] the input functions could be much more complex. In N-CLIPS this complexity can as well be mapped directly in a neuron rule.

6.1.5 Threshold types

A high pass threshold is the most general type used, where if a neuron's activation is above a certain threshold it fires. A low pass threshold type is characterized by its ability to allow a neuron to fire only if its activation is below a certain threshold. The band pass (and the multiple band pass) threshold types [12] are applied when a neuron fires if its activation is within a single range of values or several ranges. These are available as choices when the user is describing a neuron and can be applied to a solitary neuron or a collection of them.

6.1.6 Constants of the Equations

The constants applied in the various equations can be changed during the network training sessions via the user interface provided by the system. Momentum factor, and Learning rates are two such constants which are applicable to the back propagation neural networks. Different momentum factors and learning rates can be applied to different parts of the network.

6.1.7 Delta functions

Delta functions, as prescribed in [3], are available in this system. Users can also add customized

delta functions to the library.

6.1.8 Error Criteria

While the mean squared error is the most generally used error function, and is the one currently supported, future extensions will provide for other error criteria (e.g. entropy).

6.2 OBJECTS (Composite)

6.2.1 Layers

This system provides both layered and non-layered neural networks. Neural layering allows for grouping of neurons wherein information is passed between a group of (layer) and its two 'nearest neighbours (layers)'. Information flow between neurons of the same layer (horizontal connectivity) is also permitted. The layers can be created, edited or deleted by the user through the system provided functions. The parameters are accepted via a template provided to the user, after which the parameters are encoded and saved in the network data structure (fig. 2).

6.2.2 BPNN

A multi-layer feed forward neural network which follows the generalized delta learning rule is provided with modifiable parameters. The user can specify in the BPNN template the number neurons/layer, the number of hidden layers, the bias (threshold) values, the input/output and activation function, layer specific learning rates and momentum factors and other parameters from a list default and optional parameters provided by the system. The user can also update the links between neurons by the link map editor.

6.3 ACTIONS (Primitive)

6.3.1 Create, Edit & Delete Neurons, Synapses

The user shall be given a library of functions for creating and modifying the above mentioned objects. The create_neuron function can be called from within a C program or from the CLIPS interpreter just like defrule. In CLIPS> the user can enter the parameters of a neuron from the template provided. The template will carry default parameters and also provide help on different options available for each parameter. The parameters have to be passed to the create_neuron function if called within a C program. The function will encode the parameters into a special rule and shall also update the network data structure (fig. 2). The function for creating a synapse is called create_synapse and it also is C and CLIPS callable. The synapse information though is only stored in the network data structure. Other functions like edit_neuron and edit_synapse, are basically invoked in the CLIPS interpreter. They let the user modify the values of the neuron/synapse parameters. The delete_neuron functions simply take the neuron and layer numbers and delete the neurons and the links from/to them. The delete_synapse requires the 'from' neuron and layer numbers and the 'to' neuron and layer numbers. The network data structures and clips data structures are updated accordingly.

6.4 ACTIONS (Composite)

6.4.1 Create, Edit & Delete Neurons, Synapses

When a group of neurons or synapses have similar characteristics they can be created, edited and deleted by a single function call. Functions to create, delete and edit a group of neurons and

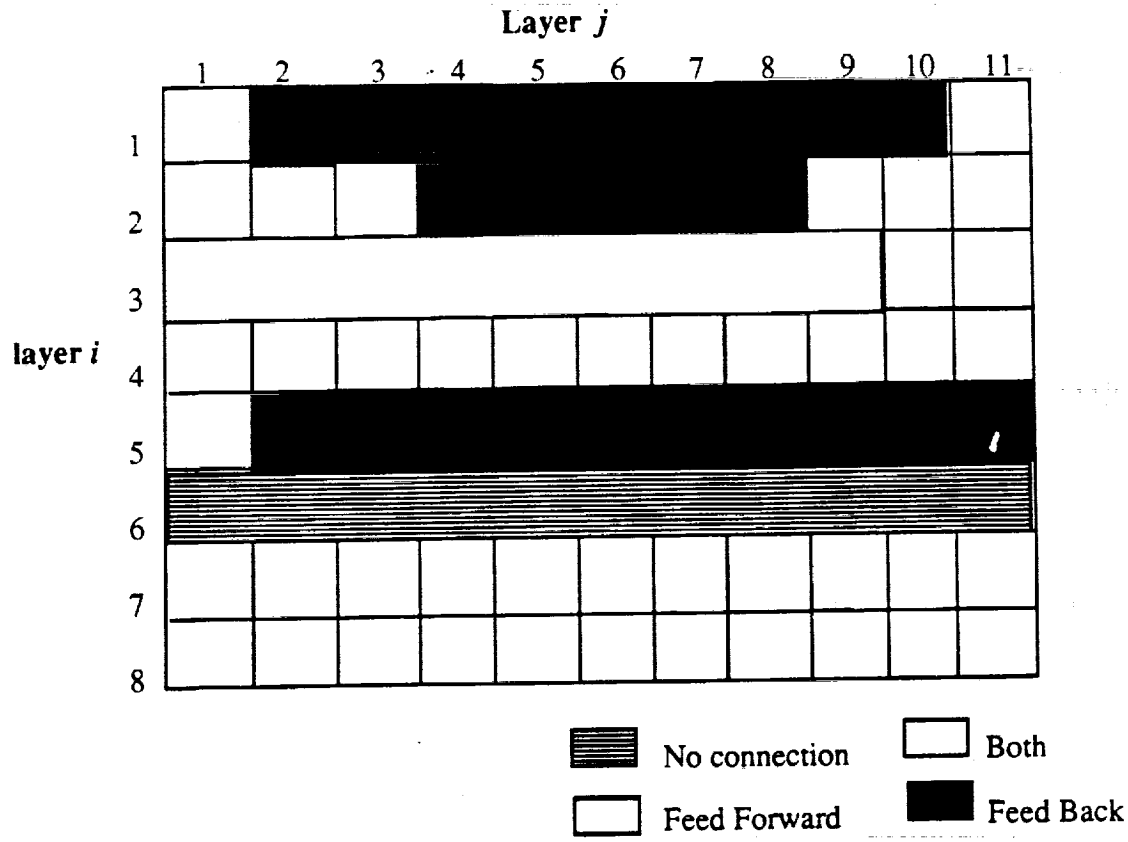


fig 3a. X Windows Link Map Editor : Modifying links, an example.

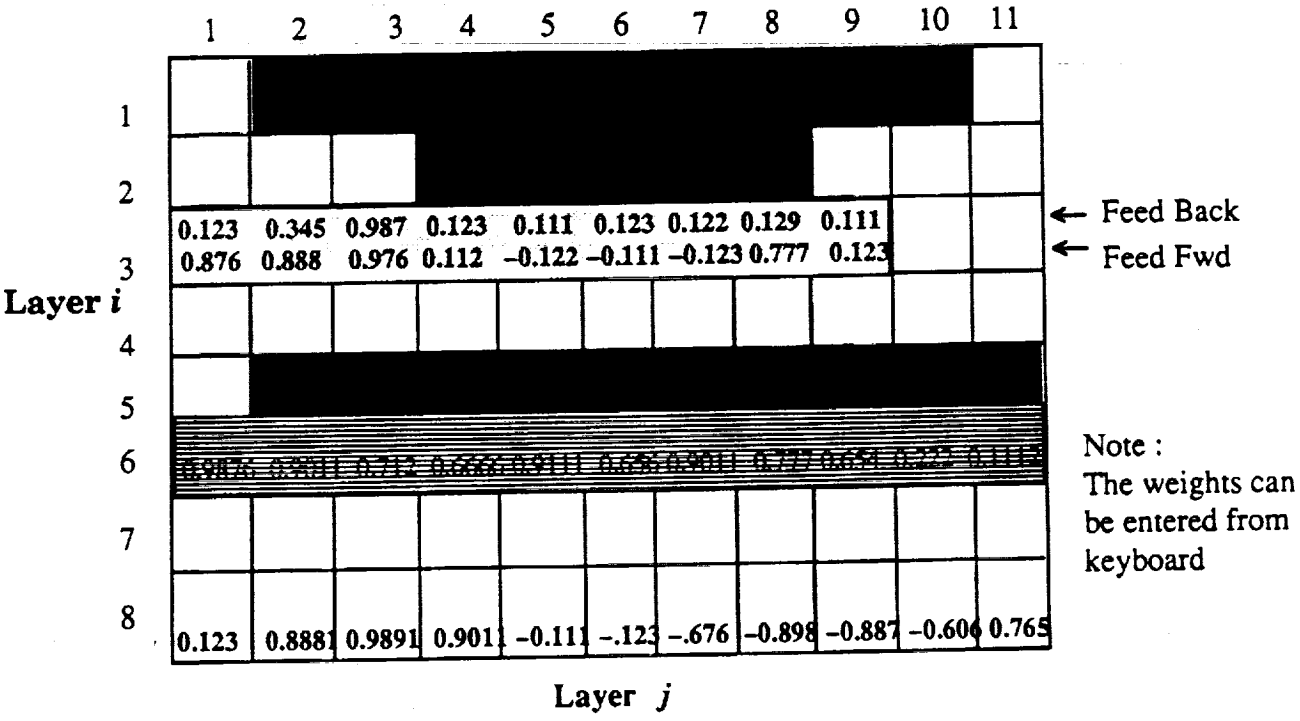


fig 3b. X Windows Weight Editor : Modifying weights, an example.

synapses are provided in the function library. As in the case of primitives, these functions (for the actions) can also be accessed both, from within a C program and from the CLIPS interpreter. The template invoked from the interpreter, however would request additional information from the user apropos the number of neurons, synapses or the layers under consideration, their topological relationship etc. The group is treated as a composite object in the system which stores it as a collection of possibly inter-connected primitive neurons and synapses. These groups can be connected to other groups, though it is a very difficult task to determine the actual neuron to neuron connection as it could be a one-to-one, one-to-many or a many-to-many from one group to another. Also, the connection from one group to another can be a higher level, logical (or abstract) connection. Besides these there can be a neighborhood effect [10] which can be programmed into the group as a rule. The creation and editing of groups of synapses is carried out with an X-Windows link map editor explained next (fig. 3a). The weights of the links can be changed through a similar graphical editor.

6.4. 2 X-Windows link map editor

It is a two dimensional link map where the rows represent the 'from' neurons on a layer and the columns represent the 'to' neurons in another (defaulted to next). It has a mouse interface to switch between four types of connections, namely the feed-forward (black color), feed-back(white color), none at all.(B&W pattern 1), both (B & W pattern 2). After the user has created or modified the links between two layers and has saved them, the map will return a matrix with the values (-1,0,1,2) for feed-back, 'none', feed-forward or 'both' connections between neurons. The user could then either use that matrix to create his/her own link specs in a C program or can let the library function create and modify the data structures. The map has default link connection specifications to create the links automatically.

6.4.3 X-Windows weight editor

It is the same as the link map editor in appearance and functionality with the exception that the user can enter the weights or modify them manually for each type of synapse at the time of creation or at any point during training, even during the execution (fig. 3b).

6.4.4 Create, Edit & Delete Layers

These can be created via direct function calls to create layers, or can be built incremently by first creating the other sub-components of the layers. The layers can be of basically three types input, output and hidden, though feature maps usually have only one layer. The system provides functions to create a standard layer or a group of them. These can be edited as individual layers or a group of (hidden) layers. Once all the neurons on a layer are deleted, the layer automatically collapses. Deleting a layer would result in all connecting synapses being purged too. If a hidden layer is deleted resulting in partition of the network the user shall be prompted with available options which would include destruction of the network and default connections.

6.4.5 Create, Edit & Delete Networks

A user can create, modify and even delete complete neural networks. In this system the user will have the capability of creating his/her own networks by either modifying the system defined neural networks (BPNN, currently, is the only available Neural network) or by customizing one of his/her own.

6.4.6 Ntrain

This function is a set of expert system rules (in CLIPS) which is system defined for feed forward type networks. But the user can write his own training function, if desired. The system defined training function first reads the input pattern and then systematically triggers each layer. To write ones own training function the user will have to write an expert sub system which will then override the previously defined training function. It could be possible to have different training functions if the network consists of different learning algorithms as sub networks. Since there can be more than one network active at any given time, the training functions should be classified by the network number to which they pertain.

6.4.7 Nrun

The neural networks or sub networks can be run from a CLIPS interpreter, a C program, or can be spawned off from CLIPS rules. Since there can be more than one network active at any given time, hence this function also needs to be passed a network identifying number.

6.4.8 Freeze

This function pauses the execution of the network after which the save function can be called to save the snap shot of the system for later analysis.

6.4.9 Show_ready

If the user wants to know, at any given instance, which set of neurons is ready to fire, he can invoke the show_ready function. This function provides a display, either in the form of a list of neurons or as a change of neuron color in a graphical representation of the neural network interconnections. The function can be invoked via a mouse.

6.4.10 Save, Load

A neural network can be saved at any given time in the disk files via the save_nn() and load_nn() functions. The save function saves all the rules in appropriate files and also the data structure associated with that network. The load function reads the same files and builds the neural network representation within the system.

7. CONCLUSIONS

N-CLIPS has turned out to be a very useful tool for solving real life technical problems for which a single knowledge representation or AI technique does not suffice. The building-in of a neural network simulator within CLIPS (the expert system shell) made it easy for the two to communicate with each other, share a common fact (data) base and utilize the other's strengths to overcome its weaknesses (e.g. expert systems brittleness versus the neural networks associative capabilities). The problem of mapping one system into another is a very difficult research topic to be addressed in future extensions of N-CLIPS. As far as the neural network paradigms are concerned, we plan to add all known learning paradigms as stand alone objects. The user-interface, can be enhanced to a complete windowing environment (e.g pop-up menus, mouse selectable options list, graphic templates, etc). The most important enhancement to the system would be the incorporating of semantic networks, searching algorithms, more general connectionist networks, frame based systems, and even petri nets.

8. REFERENCES

[1] Hoskins J. and Jones W. "Back Propagation", BYTE , Oct 87.

[2] D'Autrechy C.L., Reggia, J.A. "MIRRORS/II, Connectionist Simulation", First Annual INNS Meeting, Boston, 1988.

[3] Rumelhart et al, "Parallel Distributed Processing", vol I., 1987, MIT Press,Cambridge,Mass.

[4] CLIPS User's Guide, Artficial Intelligence Section, Johnson Space Center,June 1989.

[5] McGee B., Miller M., Krolak P., and Barr S., "Interfacing PCLIPS into the Factory of the Future", "The First CLIPS Users Group Conference", NASA J.S.C., Houston, Texas, Aug. 1990.

[6] Rabelo L.C. and Alpteking S., "Synergy of Neural Networks and Expert Systems", Proceedings of the Third TIMS/ORSA Conference on FMS, MIT, Cambridge, MA., Aug. 1989.

[7] Bhatnagar H., Krolak P., and McGee B. "A Traffic Controller for Material Handling Systems", submitted to SOAR Conference, Albuquerque, New Mexico. June 26-28, 1990.

[8] Miller R., Korlak P. "PCLIPS : A Distributed Expert System". "The First CLIPS Users Group Conference",NASA J.S.C., Houston, Texas, Aug. 1990.

[9] Coleman J. "Evolutionary Telerobotics: An Approach to the Designing of Telerobotics System", #CPE-NERV-90-5, Center for Productivity Enhancement, University of Lowell, Lowell, Ma. 01854.

[10] Kohenen T. "Self Organizing and Associative Memory. " Springer-Verlag, New York. 1989.

[11] WIlliams R. J. "Inverting Connectionist network mapping by back prop error." Proc. 8th Ann. Conf. Cog. Sci. Soc. 1986.

[12] Gelband P. "Neural Selective Processing and Learning, " Proc. of the First Ann. INNS Meeting, Boston, 1988.

[13] Collins E., Ghosh S. and Scofield C. "An application of a Multiple Neural Network Learning System to Emulation of Mortgage Underwriting Judgements, " Nestor Inc., 1 Richmond Sq, Providence RI 02906.

[14] Reilly D., Scofield C., Elbaum C., and Cooper L.N. "Learning System Architectures composed of Multiple Learning Modules".

[15] Reily D. et al., "An application of a Multiple Neural Network Learning System to Industrial Part Inspection," ISA, 1988, Houston, Texas.

[16] Pineda F.J., "Generalization of Back-Propagation to Recurrent Neural Networks, " Physical Review Letters, Nov. 1987, pp 2229-2232.

[17] Hopfield J., and Tank D.W. "Computing with Neural Circuits," Science 233, 625-633 (1986).

8. REFERENCES *Contd.*

[18] Kirpatrick S., Gelatt C.D., and Vecchi M.P., "Optimization by Simulated Annealing," Science 220, 671-680 (1983).

[19] Grossberg S., Carpenter G.A. "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," Chapter 5., Neural Networks and Natural Intelligence, MIT Press, Cambridghe, Massachusetts,1988.

[20] Kosko B. "Bi-Directional Associative Memories, ". IEEE Trans. on systems, Man & Cybernetics, vol 18, pp 49-60, 1988.

APPENDIX

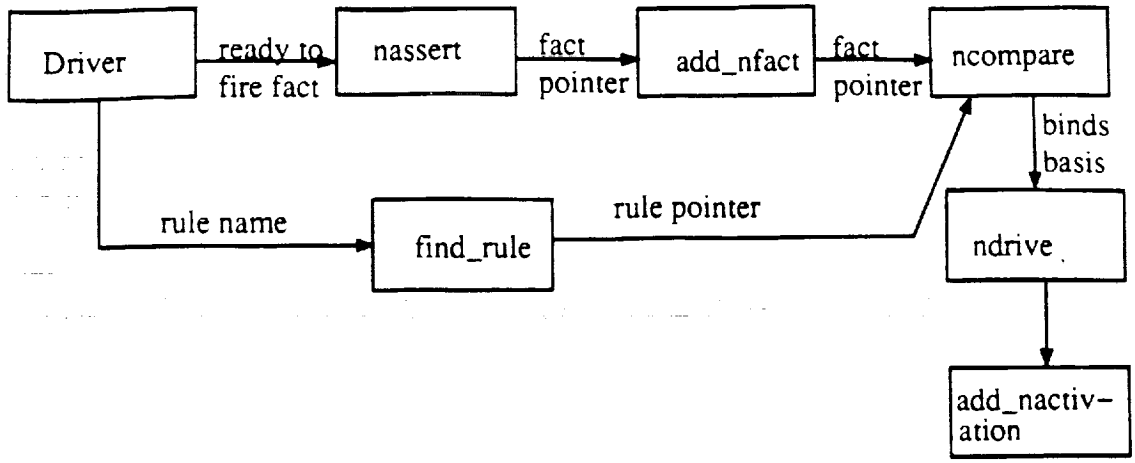


fig. 4 : **A data flow diagram of the changes made to CLIPS for N-CLIPS**

Driver

This function goes through an array of neurons (a layer) and for each neuron that is ready to fire it calls find_rule to set up a global variable pointer which points to the current neuron rule. This is followed by a call to nassert to assert the following fact : (neuron # layer # ready to fire).

Nassert

It calls add_nfact() with the above fact after making sure it has not been asserted already.

Add_nfact

It adds the above fact to the fact list and calls ncompare to filter through the special neuron rule.

Ncompare

It make s the var list (binds), the joins and gets the rule pointer from the global variable and then calls ndrive to drive the fact through the network patterns for that rule .

Ndrive

its task is to put the input parameters in proper data structures and calls add_nactivation to add the rule to the agenda.

An important feature of the above functions has been that only one rule and one fact is in picture. this is done since we know both the fact and the rule which its assertion will trigger. However in case of output neurons other facts are asserted which could trigger an expert system.

Nretract

It retracts the ready to fire fact from the fact list after the neuron has fired.

Abstract

CLIPS is being used as an integral module of a Rapid Prototyping System. The Prototyping System consists of a display manager for object browsing, a graph program for displaying line and bar charts, and a communications server for routing messages between modules. A CLIPS simulation of physical model provides dynamic control of the user's display. Currently, a project is well underway to prototype the Advanced Automation System (ASS) for the Federal aviation administration.

A prototype, as defined by *The American Heritage Dictionary*, is an original type, form, or instance that serves as a model on which later stages are based or judged.

LEVELS OF FUNCTIONALITY

The prototyping of user interfaces has evolved into four distinguishable levels. The first level is the "straw man" stage, when a basic screen design is developed that approximates how the interface should look. The purpose of this phase is to work out aesthetics issues only; it does not give any indication of the usability of the display. Using C or another script -like language, the second level prototypes static responses using limited scenarios. At this phase the objects can react to user input, but the responses do not deviate from an internal script. The third level incorporates a dynamic response from the system. During this phase the dynamic system attempts to mimic the real system as closely as possible in such areas as responding to user events and simulating (or generating) user scenarios. While using this level prototyping users should not be able to tell that they are using a prototype and not the real system. The highest level of prototyping contains everything in the previous three levels plus the ability to capture and report on usage metrics.

The function of prototyping is to demonstrate whether or not a model serves a useful purpose. At the first level, we are trying to find out if the screens are discernible; do they portray right meaning. The second level asks whether or not the prototype can respond in an intuitive manner. The third level utilizes scenarios that in turn simulate events to which the user must react. The highest level uses metrics to modify the behavior of the running system. It is important to note that the first three levels also have metrics, but they are not integrated into the prototype; they are external: surveys, video taped sessions, subjective comments of the user community.

USER DISPLAYS

Typically, static mock-up displays are the first prototypes created for most applications. They help determine spatial and size constraints for various data models. Dynamic displays are later generated to allow users to interact with the prototype.

Today's prototypes not only deal with data models, but with user models as well. For example, icons must somehow depict a similar meaning for all users. Supporting this trend is the rapidly increasing role that windowing systems are playing in today's computing environments. Specifically, the method in which information is distributed into windows and icons is important for users who are trying to understand the state of an active system.

New techniques are being developed daily that strive to go beyond the borders of windows of information into what have been termed widgets. Widgets are typically some graphical representation, in the form of an icon or window, that provide movements and actuators upon some object. An example of this type would be a sliding bar widget. In a similar manner to the sliding bars used on stereo equipment, the user can select the slide bar with the mouse and move it along the axis to set or adjust some scalar value. Widget complexity is limited only by the creator's imagination, and they can be as simple as a small radio knob dial or as complicated as the entire front panel of a virtual computer. In general, prototyping systems are becoming increasingly object oriented with data items taking on object properties. These

properties can be linked to widget functionality on the display and when an object value changes the corresponding widget can be updated.

This paper will attempt to explain one particular system that was designed to elicit user requirements through the use of prototyping user interactions. The project is called User Requirements Prototyping System (URPS). URPS is positioned at the prototyping interactions (third) level on functionality. This does not mean that the two lower levels (static and responsive) are excluded — they are also available. What we have not included as yet is a method to obtain metrics from the running prototype.

OBJECT RENDERING

Information can be represented (rendered) in different manners. A temperature can be rendered as a number; a picture of a mercury thermometer that has more pixels filled as the temperature increases; or as a square block that changes from blue to red. Any one of these methods may be appropriate in a given situation. Any object can be rendered in some manner, although the method is usually based on object functionality as far as the user interface is concerned.

WINDOWING

It is important to consider the user model as a guide to object rendering. Current windowing systems allow the designer to choose different techniques for window (or object) management. The three main types are tiled, overlapping, and pop-up windows. Tiled windows are those that split up the screen into smaller tiles — no window ever covering up another — and is based upon the user's ability to deal strictly with base spatial concepts. Overlapping windows allow for the possibility of data being covered up and are usually equipped with the ability to resize, move, and place one window over another. In user models terms, overlapping windows represent the "desktop" paradigm. Pop-up windows are interesting in that they can represent a user model that goes beyond the "desktop" into models that are based on a virtual technical assistant working with the user's "desktop." In particular, current pop-up windows are used for displaying a message about the system that you must deal with immediately (like a high priority memo on your desktop); displaying a menu that represents either local or global choices about the window below it; and displaying pop-up windows that act like post-up notes from the system.

DYNAMICS

Allowing dynamic changes to happen on the display is useful. Most user design prototypes find it necessary to know if the user can use and interact with the data that is presented. Current techniques make use of C language (object-code linkability), specially designed scripting languages, or message passing constructs to facilitate dynamics. URPS takes a combined approach in the form of an expert system shell call CLIPS (C Language Integrated Production System). Event messages travel between objects via a FACT construct. Programmability is available at both runtime via CLIPS rules and link time via C code though CLIPS.

CURRENT SYSTEMS

There are many systems currently available for prototyping user displays. Two will be discussed briefly.

The first is a low cost solution available through COSMIC called TAE+ (Transportable Applications Environment Plus). TAE was developed by NASA Goddard as a tool for building consistent, portable user interfaces in an interactive alphanumeric terminal environment. TAE also supports rapid prototyping of user interface screens and interactions, and allows the direct reuse of those screens in the final applications. TAE+ now supports X Window and MOTIF widgets.

VAPS (Virtual Application Prototyping System) is a much more elaborate, commercially available package that runs on silicon graphic workstations. The user can build prototypes by interactively laying out the display and then attaching scripts to each object. The scripts are C functions that are modifiable by the user. VAPS supports a wide range of input devices, and a designer can first prototype a control panel using just graphics and a mouse. Later, a touch sensitive screen can be added. VAPS, a sophisticated product that can prototype very realistic screens, is a product of Virtual Prototypes.

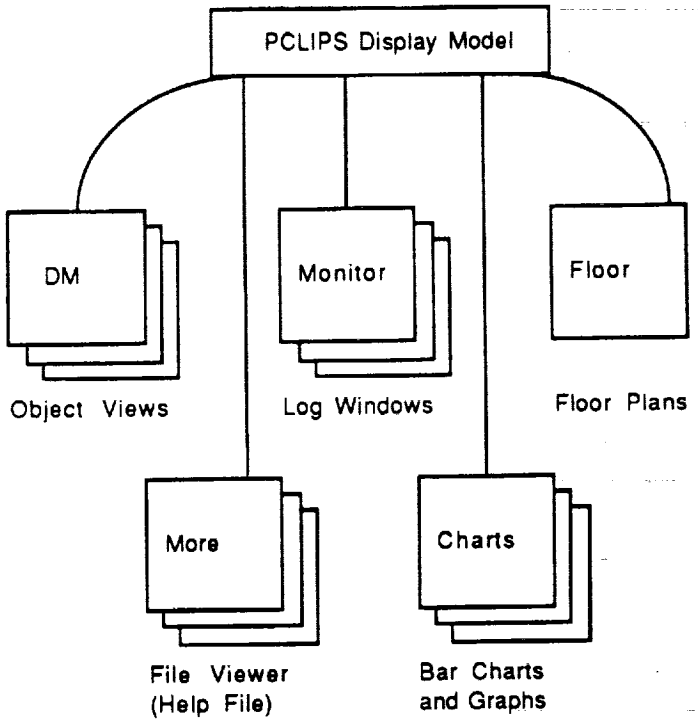


Figure 1. PCLIPS Display Model

DISPLAY MODELS

Rendering Models are based on the display devices available. These devices range from very low capability displays and very high level displays. To examine a few of these differences, three examples will be discussed here: the ANSI terminal, the IBM PC and the X Windowing System.

Using inverted text and special symbols whenever possible, the standard ANSI terminal can provide many rendering possibilities, although tiled windows seem to be the favorite on these systems. It is, however, possible to write, or use, a package can provide both overlapping and pop-up styles. Pictorially, widgets tend to be square and numbers are usually depicted with numerals. Artistically speaking it is possible to have icons that are intuitive.

The next step up from the ANSI terminal the IBM PC. The extended ANSI capabilities of the PC, along with the speed of the system, can support interesting pictorial effects. But one can always choose to tackle the graphic modes (using or buying a package). The biggest problem here is in choosing what level of graphics to support. Bit image graphics on the PC can provide a good medium for widgets; however, screen management is usually still up the programmer.

Lastly, the X Windowing System (and other windowing systems) provide window management features and widget management as well. A detailed explanation of the X Windowing System can be found in other places - it is referenced here to show that display models can vary greatly with device availability.

PROTOTYPING THE ISSS

The original work in this area was done to support the rapid prototyping of the maintenance and control consoles for the Federal Aviation Administration's (FAA) new air traffic control system, the Advanced Automation System (AAS). The purpose of the project is to develop a rapid prototyping system for a man-machine subteam to use in identifying user requirements in terms of the graphical interface. This information could then become the basis for a requirements document for the user interface.

The user displays were separated into functional groups where corresponding object structures and icons were created to represent the various objects. Functionally, the objects represented hardware and software objects that were in some state of usability. Widgets were built using the "traffic light" concept. Green means the object is functioning fine;

yellow means there is a degradation of the object; and red means that the object is dead. Blue is used to represent available but nonallocated resources.

CLIPS is being used as an event-based system. CLIPS is well qualified for this role due in part to the features of the production system model. It addition to events, CLIPS facts are being used to recreate the display model in the form of a fact base (knowledge base). These facts hold the object oriented system data about the actual objects and all the corresponding widget functionality. CLIPS rules function as receptacles for events that occur both by the simulation system and user's (display-based) events. See Figure 1.

PCLIPS is a parallel version of CLIPS that allows multiple CLIPS experts to communicate via a broadcasting function called *remote assert* (*rassert*). By using this method any number of CLIPS experts can be initiated. URPS presently has two: one that serves as a simulation of the prototyped system and another that maps simulation events to the user's screen. A display manager controls usage of the user's screen. Widgets communicate with the display manager in order to gain access to the display space and to update the data.

EVENT-BASED FUNCTIONALITY

There are two major types of widgets: an icon class made up of bit-image graphics and the other, an icon which is surrounded by a colored box; both represent the state of the object. The box type is our GENERIC class. For this demonstration we have only one icon class; it is called TERMINAL.

```
(deffacts DisplayManager "Base Object Classes for Display Manger"
;
; template: (map-dm-icon <widget-class> <widget-state> <icon-filename>)
; template: (map-dm-state <widget-class> <widget-state> <box-color.)
;
   (map_dm_icon termina up "ik:i_terminal_ok")
   (map_dm_icon terminal down "ik:i_terminal_err")
   (map_dm_icon terminal degraded "ik:i_terminal_warn")
   (map_dm_icon terminal standby "ik:i_terminal_standby")
   (map_dm_icon terminal spare "ik:i_terminal_spare")
   (map_dm_state generic up GREEN)
   (map_dm_state generic down RED)
   (map_dm_state generic spare WHITE)
   (map_dm_state generic standby BLUE)
   (map_dm_state generic degraded YELLOW)
)
```

NOTE: The *generic_display_update* and *icon_display_update* use facts sent from CLIPS to the Display Manager to control widgets. *ask_for_something* receives events from the Display Manager.

```
(defrule generic_display_update "Catch all Generic Status Changes"
   (status ?type ?object ?state)
   (dm_object ?object ?)
   (map_dm_state generic ?state ?signal)
=>
   (rasser dm turncolor ?object ?signal)
)
;
(defrule icon_display_update "Catch only TERMINAL Status Changes"
   (status CC ?object ?state)
   (dm_object ?object icon)
   (map_dm_icon terminal ?state ?fname)
=>
   (assert dm chg-icon ?object ?fname)
)
;
; (Select . . .) facts are remotely asserted by the
; Display Manager when the user does something These
; are much like user events.

; Currently, the default action is to  open up a
; subview.  If the object SELECTed does not have a
; subview, then it does not have a "map_dm_windows"
; fact either.  Another rule with a lower salience
; catches lost User Events in case there is no sub
; view.
;
(defrule ask_for_something "Catch User Events"
 ?r1<-(select ?obj)
 (dm_window ?obj ?w ?h $?Window_Stuff)
  (map-dm-window ?obj ?x ?y)
 =>
```

```
(rassert am open-window ?obj ?x ?y ?w ?h $?Window_Stuff
(retract ?rl)
)
```

DISPLAY MODEL FUNCTIONALITY

Functionally, the display is separated into views. These views consist of collections of such widgets as object views, monitor logs, bar charts, and pop-up menus. Object views are windows controlled via remote asserts (*rasserts*) to the Display Manager Screen control, and pop-up windows are also controlled by Display Manager requests. Log windows, bar charts, and the floor plan are separately running programs that join the PCLIPS session upon start-up.

IMPLEMENTATION ISSUES

The Commodore Amiga was chosen as a platform for the following reasons: Low cost, useful resolution (640 X 400), choice of bitplanes, dynamically loadable icons, commercially available image-based tools, and multiprocessing capabilities. The first challenge was porting Clips 4.3 over to the Amiga. The next challenge was in designing the actual display functions. Following this came the PCLIPS functionality; being able to allow multiple CLIPS experts to join together to form a PCLIPS Environment. This was accomplished via the recoding of a PCLIPS server which runs in the background. The server manages incoming requests to join a PCLIPS session and distributes remote *asserts* to all currently listed CLIPS processes. Once we had tools working we were then able to attack the problem of rapid prototyping the ISSS.

CONCLUSIONS

After considerable designs and redesigns, we have found widgets, object oriented programming and image-based icons to be important concepts in the development of new user interfaces. Widget technology is important for encapsulation of data and needs further study. Object Oriented approaches were definitely the way to go in our prototyping system. These approaches were used to determine the level of granularity for the prototype and also to specify functionality of object classes -- no one object was coded better or worse than another in the same class. Image-based view facilitated the involvement of art types who felt they had much more feedom with paint programs than when they were asked to layout displays based on geometrical (graphical) shapes.

Additionally, an interactive configuration tool was created to help in the layout of widgets within views, allowing objects to be positioned over bit-images (pictures). This is part of a far more interesting problem: whether to deal with image based objects or grahpical based (lines, cubes, geometry . . .) objects. One interesting group discussion led to the idea of rendering graphical objects on top of bit image backdrops.