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.

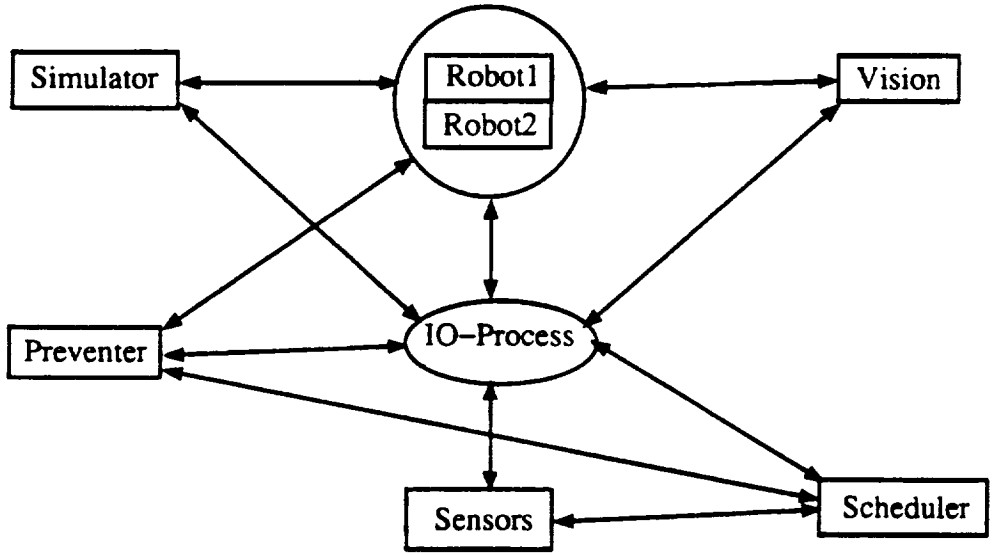

Figure 1. Workcell Processes

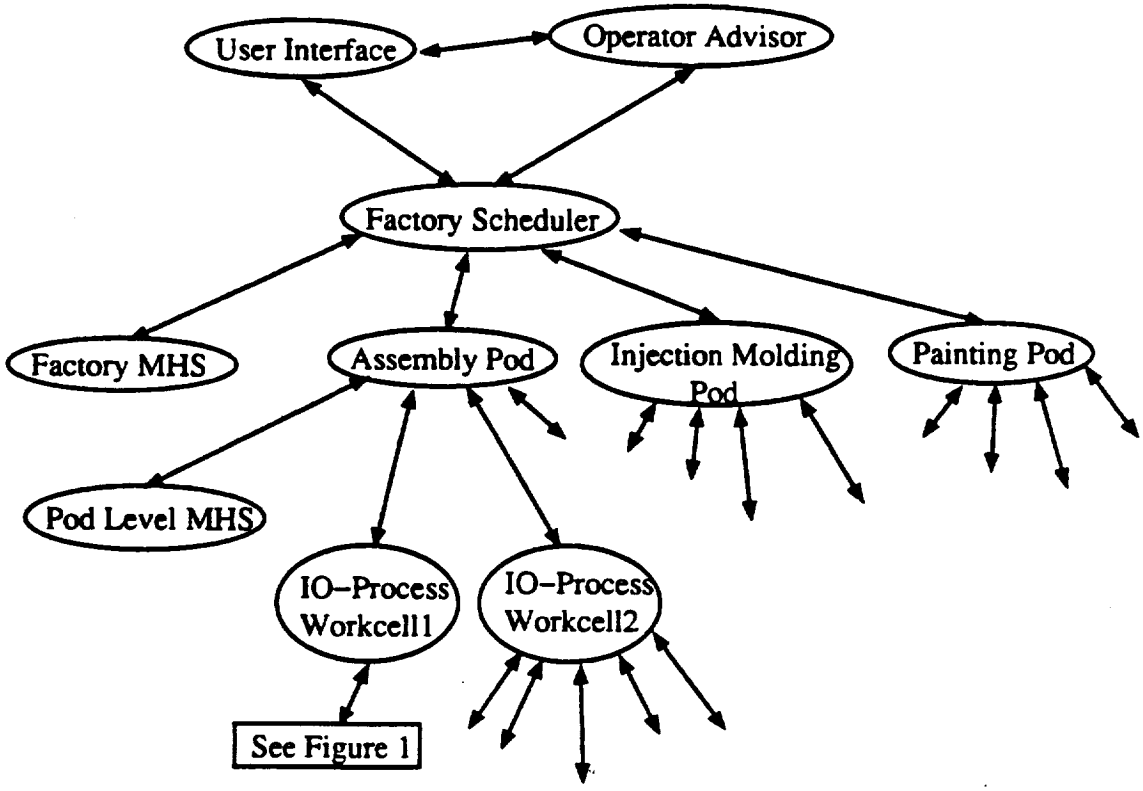

Figure 2. Factory Control

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.