Using Rule-Based Software to Debug Factory Automation Systems

by

James R. Watson

B.S., Colorado State University, 1982

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing & Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

Major Professor

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The high cost of labor has forced American manufacturing to turn to complex computer systems to handle many of the operations previously handled by humans. These complex computer systems require highly trained personnel to keep them operating. A company becomes very dependent on these personnel and is reluctant to move them to other jobs for the fear that the computer system will not be supported. An even greater fear is that the trained person will leave the company and force the operations of the computer system into the hands of an untrained individual. What is needed is a way for the knowledge of a system's workings to be integrated with the system and not in the mind of the individual developer. A method is needed for the system to tell humans what has to be done to correct errors or modify execution. A system with characteristics such as these was developed for AT&T's Denver Works in Denver, Colorado.

The Integrated Pull Manufacturing (IPM) System consists of four 3B15 computers running UNIX 5.3.1. There is one 3B15 for each circuit pack production line, the red,

orange, and merlin, plus one 3B15 that coordinates the processing of the other three. On each of the circuit pack production machines is the Manufacturing Process Control System (MPCS) software. MPCS is responsible for tracking circuit pack progress through the manufacturing process. A barcode is affixed to each pack at the first production operation. The barcode is scanned at key operations and is stored in a database which is part of MPCS. At any moment MPCS can tell a user how many packs are in production, what type they are, where they are in the process, and how long they've been in process. All this information is critical in running an efficient manufacturing operation.

The barcode readers are wired in series through a RS232C connection to an AT&T PC6300. The PC has the capability of holding 10,000 scans. If, for some reason, the data is not able to be passed to the 3B15, the PC is able to buffer this data to prevent loss of information. The PCs are networked to the 3B15s through a System 85 PBX. Because all the wiring was already in place, using the PBX made adds and changes of PC concentrators very quick and easy. The 3B15 polls each PC to see if any information is to be sent for processing. Depending on the number of PCs connected to the 3B15 the polling takes

place approximately every five seconds, supplying MPCS with near real-time data.

The fourth 3B15 runs the Shop Floor Control (SFC) software. SFC is responsible for scheduling the circuit packs through manufacturing. A database in SFC contains a list of operations for processing a pack and the amount of time required for each operation. By knowing when the pack is needed for an order, SFC can schedule all the operations of a pack's production to meet its required due date.

The SFC 3B15 is linked to an IBM 3081 via a Remote Job Entry line. The IBM supplies order due dates, order quantities and current storeroom inventory levels so that SFC may determine what to schedule for production. There is also a 3270 emulation link between the IBM and the SFC 3B15. This link is used when a pack's production is completed. The SFC machine can update the inventory levels contained on the IBM. The IBM views the SFC 3B15 as a terminal and gives it on-line access to the database.

All four 3B15s are linked together using a TCP/IP network. The network allows the machines to pass

information back and forth. It also allows for the remote mounting of file systems to allow common software, located in one place, to be accessible to all machines. This simplifies updating because it is done in one place and all machines know about the change immediately. Because of heavy processor demand placed on the 3B15s by SFC and MPCS, it was decided that no users would be allowed on the machines. Because of this restriction a way was needed to get the information from the processing machines to users machines for viewing.

There already existed a user network in the factory. Consisting of PC6300s as delivery devices for the information to the user, 3B2/400s as file servers, and a 3BNET ethernet-based network. By using another 3B2/400 as a file server, information could be passed from a processing 3B15 to the 3B2/400 file server at regular intervals. Every fifty transactions or fifteen minutes, which ever is shorter, it was possible to deliver the information to the user with a nominal effect on the processing machine.

The IPM system is a complicated hardware system. When all the software that was necessary to make it work is included it becomes a highly complex system. Figure 1-1

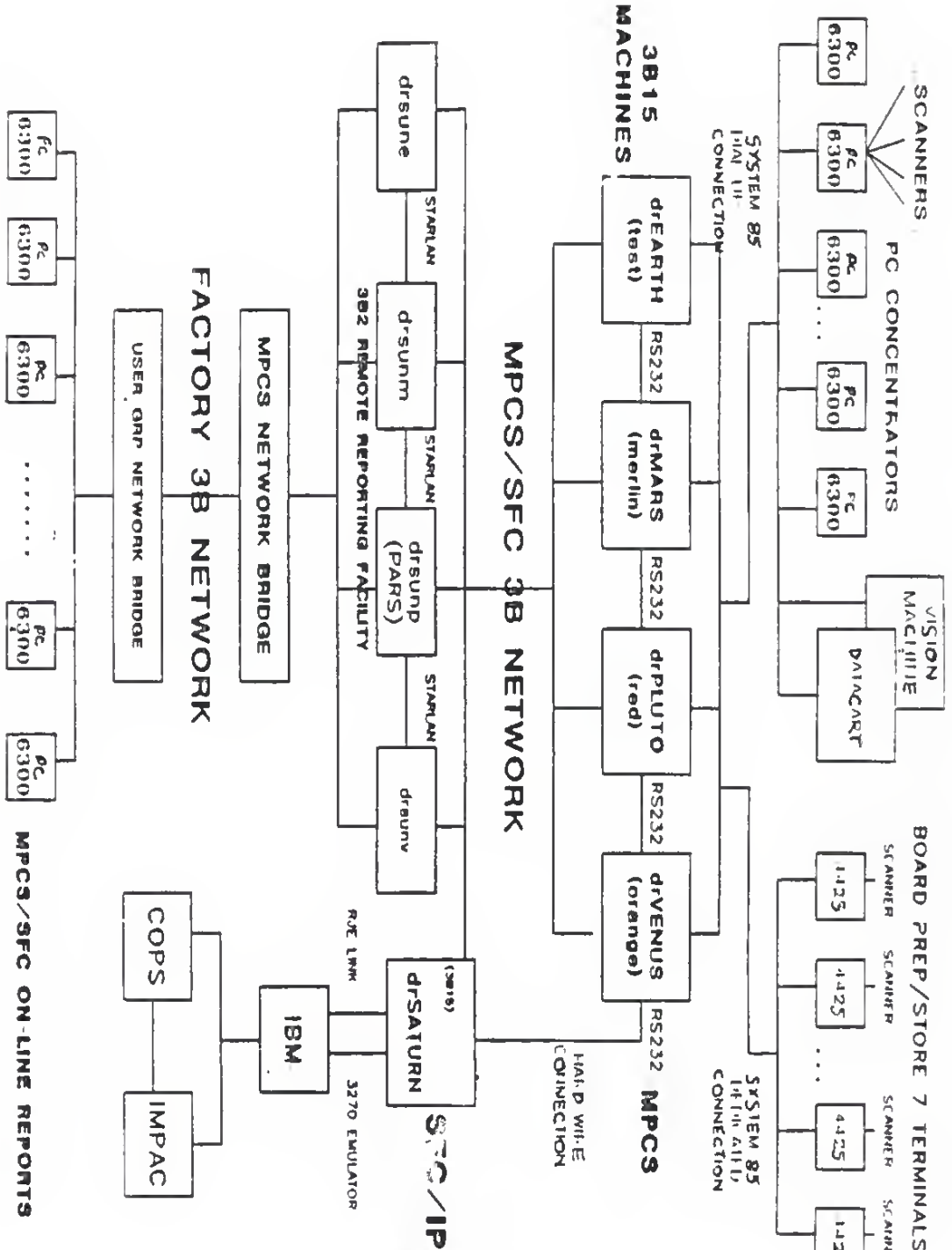
is a diagram of the system. All the knowledge of the design and workings of the system was contained in the minds of three people, two, in reality, since one person on the team was in charge of managing the project. As the system was cut to production on successive circuit pack lines, it became evident rather quickly that it would be impossible for the implementation team to continue to improve and expand IPM, and at the same time be responsible for day-to-day problems and questions.

An operations team was added to handle the daily production problems. This seemed like a good idea at the start, soon it became apparent it took more time for the implementation team to show the operations personnel the method to fix a problem than to fix the problem themselves. Another drawback of the operations team was the fact that they were hourly workers and not highly technical. Hourly workers tend to be reassigned quite often due to layoffs and job bumping, so there was a high percentage chance that the same personnel would not be there for a long period of time. What was needed was a way to transfer knowledge about the system from the implementation team to the operations team. It had to be easy to implement and maintain, usable in real-time problem solving, able to do a step-by-step procedure for

problem corrections, and need minimal training to use.

With this need in mind, it was necessary to research the feasibility of building such a system.

Figure 1-1


SCANNERS
PC CONCENTRATORS
PC 6300
VISION MACHINE
DAINCART
SYSTEM 85
3B15 MACHINES
drEARTH (test)
drMARS (merlin)
drPLUTO (red)
drVENUS (orange)
RS232
MPCS/SFC 3B NETWORK
drsune
drsunm
drsump (PARS)
drsunv
STARLAN
3B2 REMOTE REPORTING FACILITY
MPCS NETWORK BRIDGE
FACTORY 3B NETWORK
USER GRP NETWORK BRIDGE
BOARD PREP/STORE 7 TERMINALS
SCANNER
4425
HARD WIRE CONNECTION
drSATURN
IBM
3270 EMULATOR
COPS
IMPAC
MPCS/SFC ON-LINE REPORTS

2. LITERATURE SEARCH

"An expert system is a body of knowledge, embodied in a set of rules, that can be accessed and interrogated by a non-expert" [1]. This definition points out a key part of expert systems, that they be accessible and usable by non-experts. It also seems to be limiting in other areas. The definition doesn't explain why a non-expert would want to interrogate the system or what the purpose of the expert system is. By combining a second definition: "An expert system is a computer program that embodies the knowledge an expert uses when doing a task that requires reasoning and, using that knowledge, performs the task as well or nearly as well as the expert" [2]. We come up with a suitable description of an expert system: a system that is usable by non-experts and allows the non-expert to perform the task nearly as well as the expert. With this description of an expert system in mind, let's examine how they operate. Expert systems consist of three parts: the knowledge base, the inference mechanism, and the user interface.

The knowledge base is the most important part of an

expert system. "The power of an expert system derives from the knowledge it possesses, not from the particular formalisms and inference schemes it employs" [3]. Along with the importance comes a degree of difficulty that makes the knowledge base time consuming and challenging to complete. The steps needed to construct a knowledge base are information gathering, encoding, and confidence assigning.

The goal of the information gathering phase is to discover the heuristic or rule-of-thumb that the expert uses to solve a problem. Most experts don't use a step-by-step procedure to solve problems. Because of experience they use a "gut-feeling" type of logic when confronted with a situation. The knowledge engineer, the person building the expert system, has the responsibility of capturing that "gut-feeling" process from the expert and explaining what is happening.

One way to unearth the expert's process is to ask the expert to explain. This seems logical but past studies have shown, "The more competent domain experts become, the less able they are to describe the knowledge they use to solve problems" [4]. A better way to gather the information is to observe the expert at work and have

him/her talk through the process as it is being done. After several repetitions of the problem, the knowledge engineer should be able to explain what has to be done to correct the situation. It is not necessary, or even likely, for the knowledge engineer to know why something is done a particular way. Just knowing that this is what is done is sufficient. Once the knowledge is gathered, it is necessary to encode it in a form the system will understand.

The knowledge is encoded in what is generally called rules. A rule is a condition-action pair that is used to represent past, present, or future events in the process of solving a problem. A rule would be analogous to the if-then statement in classical programming. By coding the expert's knowledge into condition-action pairs, it is possible to "see" all the variables and logic that goes into the solving of a problem. When encoding the knowledge, it will become evident that all options are not yes or no, black or white, true of false. It is necessary to have a way to deal with uncertain logic. Confidence assigning is an attempt to handle this uncertain logic.

When an expert is deciding what steps to do next, there is a good likely-hood that several possible choices exist. The expert has a confidence, because of experience, of which choice is the best for the current situation. Expert systems can simulate the best choice by using certainty factors which indicate the degree of certainty with which the answer is believed [5]. The certainty factors may be assigned by calculating the frequency with which a fix actually solved a problem in relation to the attempts. As more experience with the system is gained the certainty factors can be modified manually. If you have a system that is capable of learning, "Machine learning is any automatic improvement in the performance of a computer system over time, as a result of experience" [6], this modification can be done automatically. After the information gathering, encoding, and confidence assigning, the knowledge base has reached an acceptable state for testing and use.

The second part of an expert system is the inference mechanism. The inference mechanism or engine is the part of the system that determines which rule to execute and when to execute it. Inference engines may be written in any language. It could be a structured, predicate logic type of language like PROLOG, a non-structured symbol

manipulating language like LISP, or a classical language like COBOL or PL1. The concerns about an inference engine are not what language is used but what it does and how it does it.

The what it does can be broken down into three components: match, select, and act [7]. When matching, the inference engine uses the current environment and collects all rules whose condition part of the condition-action pairs match the current state. The inference engine will then select one rule to execute. If there were several rules that matched the current state, then some type of scheduler would determine which rule to execute. Once a rule is selected, it is then acted upon. When a rule is acted upon or "fired", it causes the modification of the current state to a new state. After execution there is a new current state and the match-select-act process starts again. This loop continues until an end condition is reached or no condition of a condition-action pair matches the current state. This match-select-act process can either be done by a forward-chaining process, a backward-chaining process, or a combination process.

Forward-chaining is the process of starting with some data and chaining forward to come to a hypotheses or answer. In backward-chaining the answer or hypotheses is known and by chaining backward, data is found to support that answer. In combination-chaining, both backward-chaining and forward-chaining are used to try to arrive at the stopping point more efficiently than either method could on its own.

The knowledge base and the inference mechanism are in the background of the system. What people are interested in is "What am I going to see?" What the user sees is known as the user interface.

The amount of effort spent on the knowledge base or inference engine will be obscured by a poor user interface. A few key items can be the difference between a good user interface and a poor user interface. Three of the most important are usability, helpfulness and understandability.

Usability is the characteristic of a system with screens that are easy to read and to master. No one likes to spend the time to read manuals or instructions on how to make something work. If a novice can, after a few

repetitions, feel comfortable with the system and the screens are well designed and pleasant looking, then the system is usable.

For a system to be helpful, it has to have the capability of instructing the user when he/she gets stuck and can't figure out what to do next. This can be accomplished through help screens, menus, or anything else that will prompt the user on what to do next.

Understandability is a trait in which a system explains why a user is required to do something. If a user wants to know why a system needs a piece of information, the system explains its need for that data. The user will understand the "logic" of the system and be able to improve the system as the user becomes the new expert.

Usability, helpfulness and understandability are just three possible traits of a user interface. A system with these three interface items is in a good position to demonstrate its value.

With the description of expert systems that has been defined, "We would perhaps be better off using the term 'Computational Reasoning' to represent what we do" [8].

One other area of expert systems that should be mentioned is the testing or diagnosis of a system. "Diagnosis is the process of fault-finding in a system, . . . , based on interpretation of potentially noisy data" [9]. Unlike a typical program which can be debugged until it runs correctly, an expert system, just like the expert, will occasionally make mistakes. The important thing to remember is that if the system is correct in the same ratio as that of the human expert then the system is accurate.

3. SELECTING A SHELL

The first thing was to select the environment for the expert system. With little deliberation it was decided to use an expert system shell to develop the system. The reasons for using a shell were time, cost, and expertise.

The driving force behind the project was to demonstrate the usefulness of expert systems in the factory environment. It was also to show how quickly a system could be built for a particular situation. It was thought that it would take several person-years to develop an inference engine, not leaving a good impression on management. By using a shell the time was reduced to how long it would take to learn to use the shell.

The second reason, cost, was important because of the budget of the project, zero. The cost of developing an inference engine can grow rapidly when considering the people, the hardware, and the software that may be necessary to accomplish the goal.

The final item, expertise, was important because the inference engine would play a pivotal role in the success or failure of the project. Since no one on the team had ever developed an inference engine before or a knowledge base, it was thought that it would be too much unproven ground to attempt them both in the same project. By using a shell, it would allow the team to concentrate on the knowledge base, the ultimate power of any expert system. Because of time, cost, and expertise the shell was the only alternative. Before selecting a shell it was necessary to determine what the requirements of the shell would be.

Although it was decided to use a shell, it was important that the shell have certain characteristics. We needed a backward-chaining, PC based, commercially supported, and locally available package.

A backward-chaining inference engine was used because the system being developed seemed to be best suited for backward-chaining. The system could be implemented with any type of inference engine. From a user viewpoint there would be no differences in operation, but from a design standpoint it was thought that it would be more logical and easier to implement if backward-chaining was

selected.

Because of the set-up of the factory it would be ideal for the system to be PC based. Just about everyone in the factory, including the hourly workers, use PCs in their daily jobs. The people felt comfortable using their machines, making it easy for them to master the use of the expert system with little training or documentation. By putting the expert system on something that the ultimate user was already familiar with, the system could be sold as a new feature to an existing product as opposed to a new system that the user would have to learn.

A commercially supported product was overwhelmingly endorsed by the team. With AT&T Bell Laboratories access, consideration was given to using a home-grown shell. It was thought that a home-grown product would allow the possibility of customization to the shell for the particular project along with direct communication with the developer, in case of problems. After further thought, and remembering that the emphasis was to be placed on the knowledge base, a commercially sold system was determined to be the best. It was felt that the commercial system would be well tested, well documented,

and demonstrate that expert systems could be produced with off-the-shelf products as opposed to expensive local development. The requirements of backward-chaining, PC based, and commercially supported narrowed the shell selection down to tens of systems. The final addition to the equation was local availability (remember the budget). This drastically reduced the possible selections to a point where a decision on which shell to use could be made.

The final decision was that Teknowledge Inc.'s M.1 would be the expert system shell that was used for the project. The reasons for selecting M.1 were these; it met the requirements, additional features existed that would be very useful in designing and using the system, it was easy to understand the shell, and the shell was available.

M.1 had all the requirements that were necessary in an expert system shell. Produced by Teknowledge Inc. of Palo Alto, California, M.1 is a backward-chaining, PC based product that has been commercially available since 1984. The software package had been purchased locally when interest in expert systems was just beginning, but nothing had ever been done with the software. The

documentation was well written with an entire book of sample expert systems built with M.1. Along with the fact that M.1 met all the requirements that had been laid out, the additional features that came with the package were very attractive.

M.1 had many useful features built into the system that would enhance the system's use. Three of the key features were a trace option, design versus user environments, and the user interface.

The trace option is helpful in the design phase of building the system. It allows the designer to execute the expert system one step at a time. This allows the designer to watch the rules as they are firing, the order in which things are done, and the conclusions that are reached as the system tries to process its way to the goal. This is of extreme importance as the designer is trying to correct something that isn't working the way he/she thought it should.

The two environments, design and user, are important because both classes of people do not want to see the same thing when they work with the system. The designer is more interested in what is going on and why. The user

wants to see just questions and answers. On M.1 a function key allows the flip between the two environments. When in the design phase the designer is allowed to see what is presented to the user, along with what is running in the background processing of the system.

The feature that impresses a user is the user interface and M.1 has an excellent one. The screen is broken up into regions with questions appearing in one area, possible answers in another, and system messages in a third. The interface supports multiple colors which allows the high-lighting of important items by using a contrasting color for those items. Pop-up windows relieve the user of having to remember combination key strokes for commands. The trace option, design versus user environments, and the user interface made the choice of M.1 seem like a wise decision. The only thing remaining was how long would it take to figure out how to use the package.

The ease of learning M.1 is probably the most impressive feature about the software. In one afternoon a semi-literate computer user can be designing expert systems using M.1. In a week it is possible to learn enough

tricks and time saving short-cuts to feel very comfortable with M.1's capabilities. At the beginning of the search for an inference engine it was felt that it was an impossible mission. By the end a system was found that met the requirements, had additional useful features, and was easy to learn. The goal had been reached and M.1 was it.

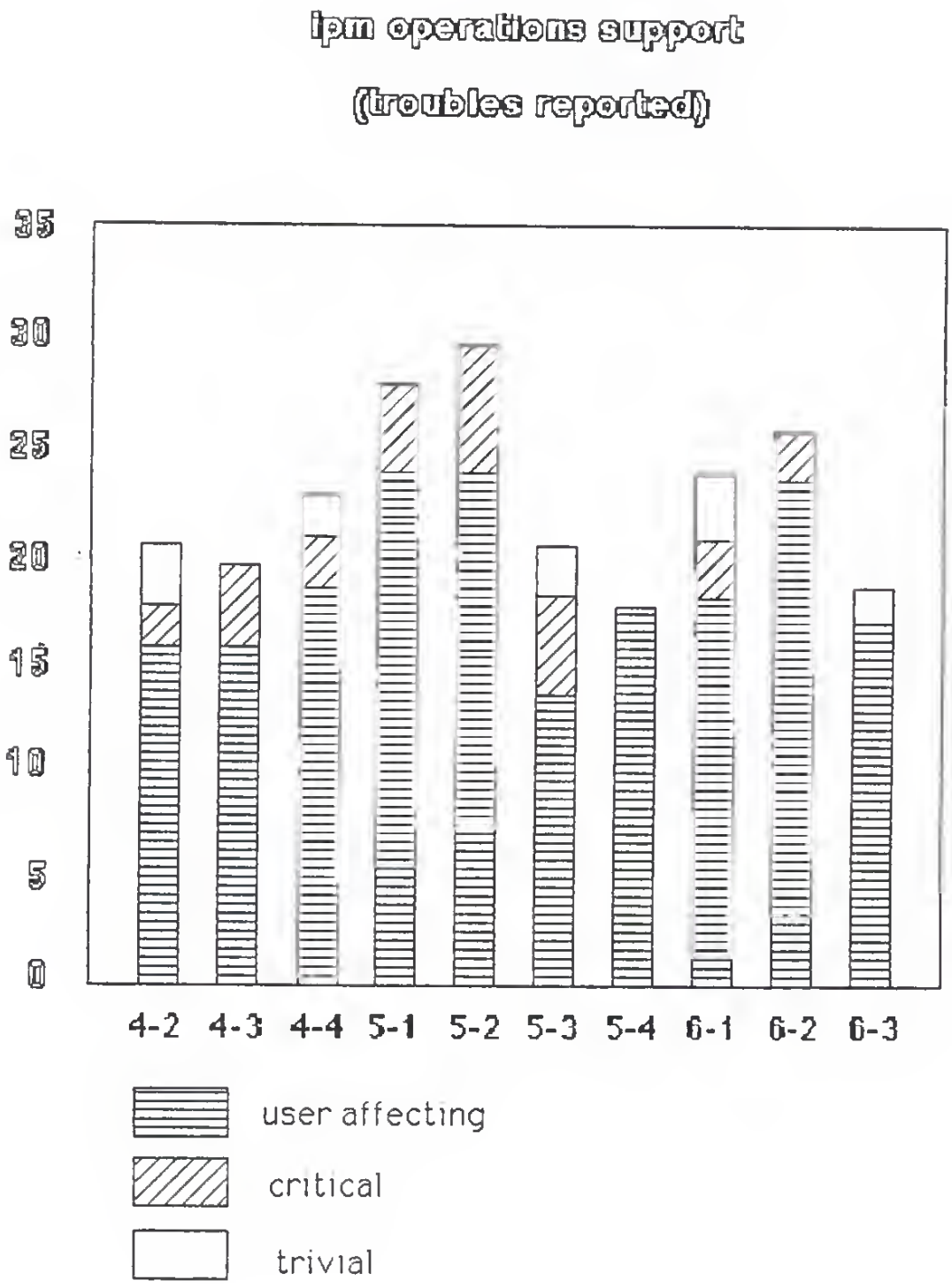
4. BUILDING THE KNOWLEDGE BASE

When building a knowledge base it is important to determine that the knowledge to be encoded into the system is that which is needed to do the job. In the case of a diagnosis system, is the fix that the system selects a fix to a problem that the user is having? The way to determine this is to collect data on the problems, analyze the data, and pick problems from the data that could be solved by the expert system.

To gather data on the problems for this project, a trouble form was designed to capture information. When a problem was discovered, an operator would write a trouble ticket with a brief description of the problem. The ticket would then be given to a design team member. The team member would classify the type of problem and write a brief description on what was done to correct the problem. Although this was a time consuming process which slowed down the correction, it was necessary to get a handle on what was happening in daily operations. Appendix A contains examples of the trouble forms that were used.

Once the data was collected it was necessary to analyze the information. By analyzing the data it was possible to determine the frequency of problems, the relation between problems, which circuit pack line had more problems than the other circuit pack lines, and other information that would be useful in understanding the current operational situation. Figure 4-1 is a graph with the problems broken down into three categories: critical, trivial, and user affecting.

Figure 4-1


ipm operations support
(troubles reported)
35
30
25
20
15
10
5
0
4-2
4-3
4-4
5-1
5-2
5-3
5-4
6-1
6-2
6-3
user affecting
critical
trivial

A critical problem is one in which the entire system is brought down. Examples of critical problems are a system fault, an electrical power hit to the processor, or a disk crash. It was decided that the critical problems would continue to be the responsibility of the design team. It was felt that the frequency of these problems was low and the technical knowledge to fix them was high. This combination made it desirable for the design team to handle them.

The second class, trivial, consists of problems such that something on the system wasn't exactly what the user wanted, but the system is operational. Examples of trivial problems are: need a longer cord for the scan gun, don't like the color of the print on the terminal, and want the barcode in a different place on the circuit pack. To handle these problems one day bimonthly was picked as "trivial day". The operations team and the design team would get together and go over the list of trivial problems to solve as many as possible.

The third category of problems, user affecting, is the category where the design team was spending the majority of their time. This is the area at which the expert system would be targeted. It was felt that if the

operations team could handle this class of problems, it would free the maximum amount of time for the designers. User affecting problems are ones in which one or more users are unable to use the system. Examples of user affecting problems are a broken scan gun, terminal not responding, and inability to login to the system. Figure 4-2 is a table showing the occurrences of different types of user affecting problems. By using the expert system to help with the majority of the problems that happen most frequently it was assumed that the system would achieve maximum benefit early in the process. Once the data was gathered, the problems analyzed, and the possible implementation problems selected, it was necessary to examine how the solutions to these problems were effected.

Figure 4-2

Slow Scan Gun	0	0	1	1	0	1	0	0	0	0
Can't Access Reports	0	1	2	2	2	3	1	1	4	5
Buffering Scans	2	5	1	2	2	3	3	0	1	0
Select Not Found	3	6	4	1	0	0	4	2	1	2
Faulty Scan Gun	3	2	3	4	3	4	5	5	6	2
Faulty Terminal	9	9	4	3	5	8	11	7	3	3
No Remote Records	0	1	0	0	0	1	0	0	0	0
Reports Not Updated	1	1	3	3	0	5	1	4	2	5
	6-3	6-2	6-1	5-4	5-3	5-2	5-1	4-4	4-3	4-2

The solutions were written on each trouble report along with the problem. In some cases solutions were two or three word phrases so it was necessary to go back to the designer and get more complete responses. Once each problem was solved, a correlation was done to see if the same problem consistently had the same fix. It didn't have to be 100% of the time, but it had to be often enough that the fix would solve the problem in most cases. After determining a set of problems and a set of consistent fixes, it was necessary to determine if the operations team would be able to effect the fix. In some cases "super-user" privileges or special tools were needed to correct the problem. Since these things were not available to the operations team, it would be impossible for them to fix the problem. These types of problems were removed from consideration for implementation. When the final set of problems was picked, a written step-by-step diagnosis was produced for each problem. The written descriptions appear in Appendix B. From these descriptions the knowledge base would be encoded.

The encoding of the knowledge base was a smooth operation, thanks, in-part, to all the up-front work that was done before encoding ever started. The M.1 syntax is

simple and the error messages are fairly helpful in finding problems. Clarifying the solutions, determining if the solutions are possible, and encoding the knowledge base brought the system to a point where it was now possible to begin testing.

5. TESTING THE EXPERT SYSTEM

The testing of the expert system was cause for deliberation and thought. Knowing that an expert system would make mistakes in its diagnosis, it seemed unproductive to just test the system. What was decided was to list criteria that the expert system had to meet to be considered complete. If the system was syntactically correct running a thorough set of test scenarios, agreed in most cases with the expert's diagnosis, and was usable by novices, it would be considered to have passed test.

The M.1 system was "helpful" in locating syntax errors in the knowledge base. After several iterations of the load-correct-reload cycle it was possible to get the knowledge base to load. Once loaded the knowledge base was ready for execution. By stepping through trial diagnosis it was possible to watch the execution to see if the rules were processed in the expected order. While running the trial diagnosis, it became evident that it would be difficult to test all possible combinations of answers to verify that every branch of the knowledge base

was tested. Trees were constructed to determine if all possible branches of the knowledge base were represented. Figure 5-1 thru figure 5-6 are the tree structures representing the knowledge base. From looking at the trees it was seen that several possible branches were not included. Most of this was caused by the built-in unknown answer that M.1 automatically assigns as a possible answer to every question. Modifications were made to the knowledge base to allow the unknown branches to behave like the no branches in a yes-no-unknown tree fork. Since the missing branches weren't designed into the system at the beginning, the way they are handled is not as smooth as it could be. The knowledge base had now completed the syntactically correct and running criteria.

Figure 5-1

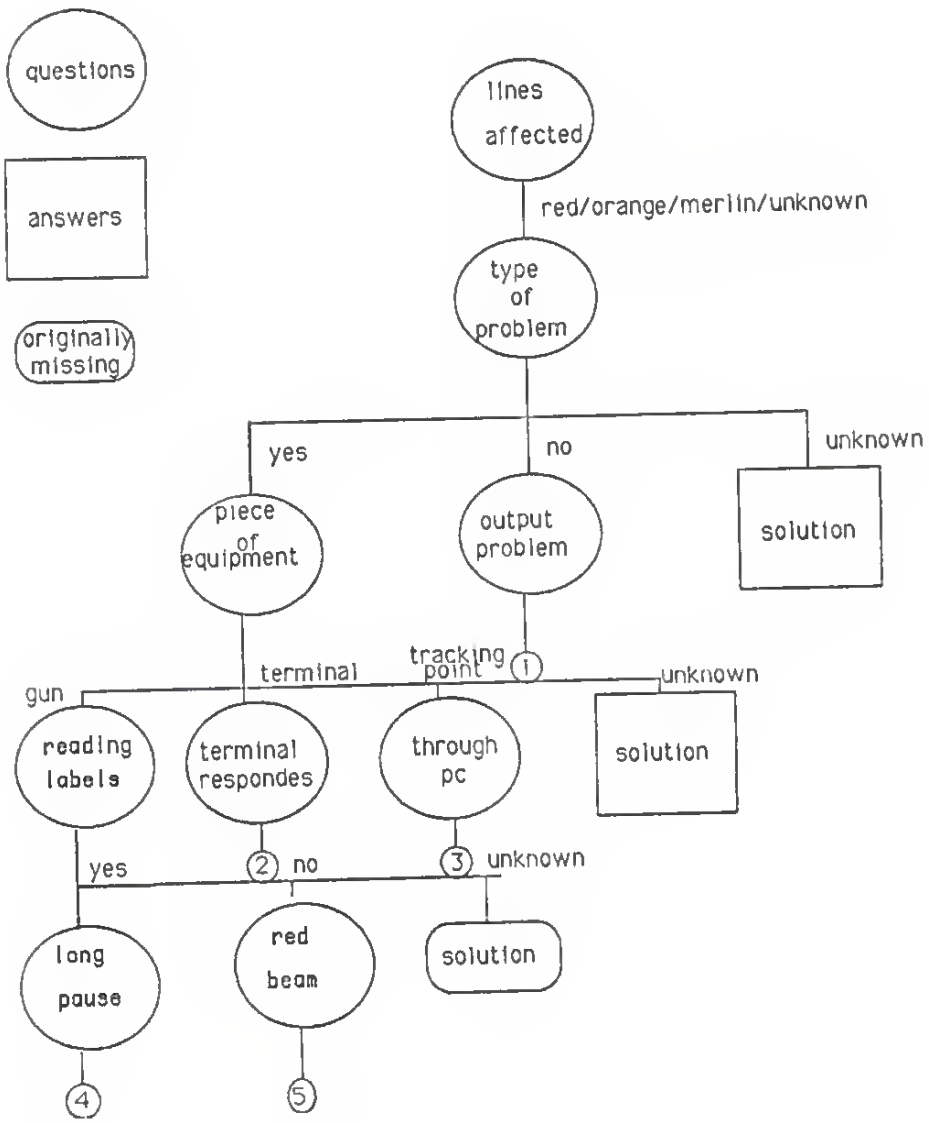

questions
answers
originally missing
lines affected
red/orange/merlin/unknown
type of problem
yes
no
unknown
piece of equipment
output problem
solution
terminal
tracking point
1
gun
unknown
reading labels
terminal responds
through pc
solution
yes
2
no
3
unknown
long pause
red beam
solution
4
5

Figure 5-2

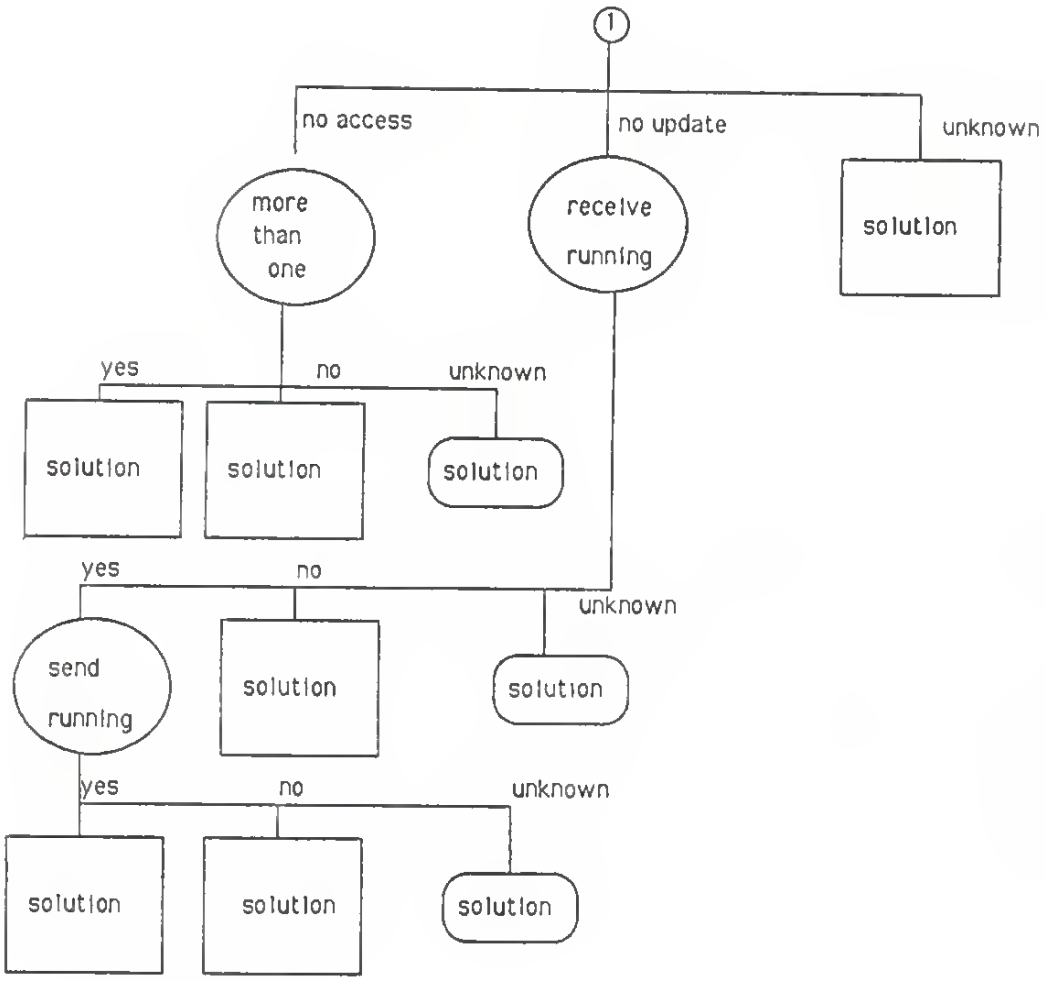

1
no access
no update
unknown
more than one
receive running
solution
yes
no
unknown
solution
solution
solution
yes
no
unknown
send running
solution
solution
yes
no
unknown
solution
solution
solution

Figure 5-3

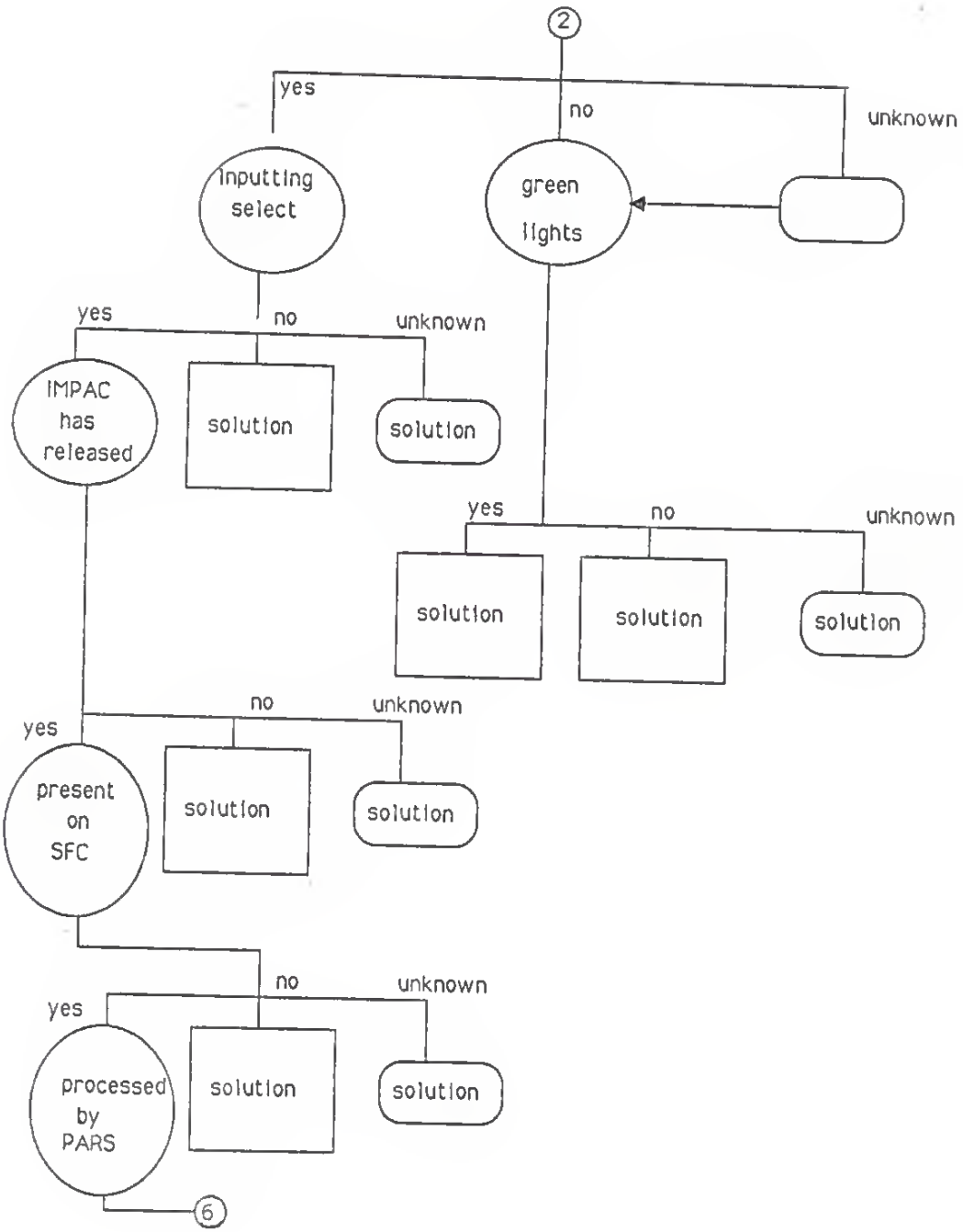

2
yes
no
unknown
inputting select
green lights
yes
no
unknown
IMPAC has released
solution
solution
yes
no
unknown
solution
solution
solution
yes
no
unknown
present on SFC
solution
solution
yes
no
unknown
processed by PARS
solution
solution
6

Figure 5-4

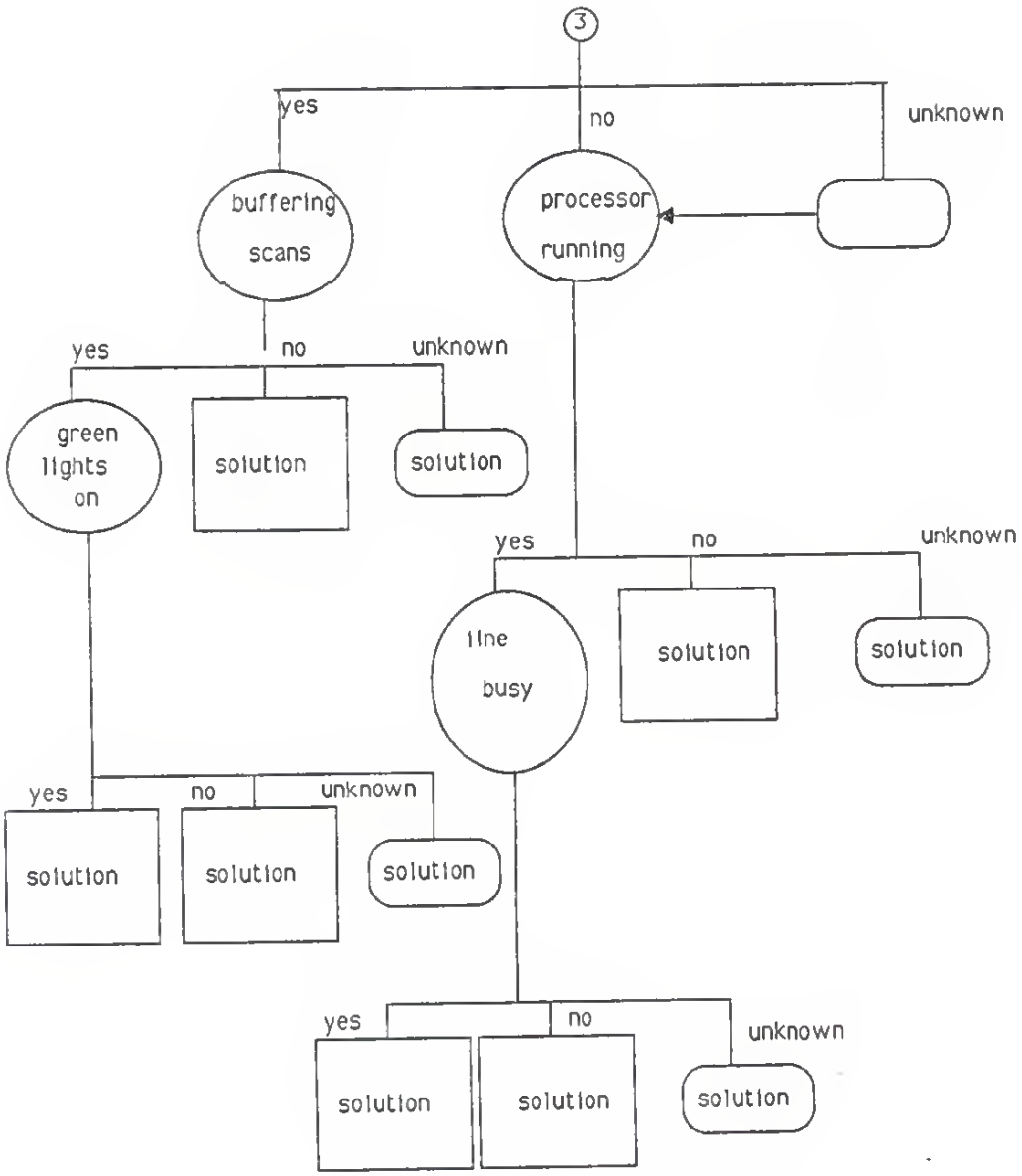

3
yes
no
unknown
buffering scans
processor running
yes
no
unknown
green lights on
solution
solution
yes
no
unknown
line busy
solution
solution
yes
no
unknown
solution
solution
solution
yes
no
unknown
solution
solution
solution

Figure 5-5

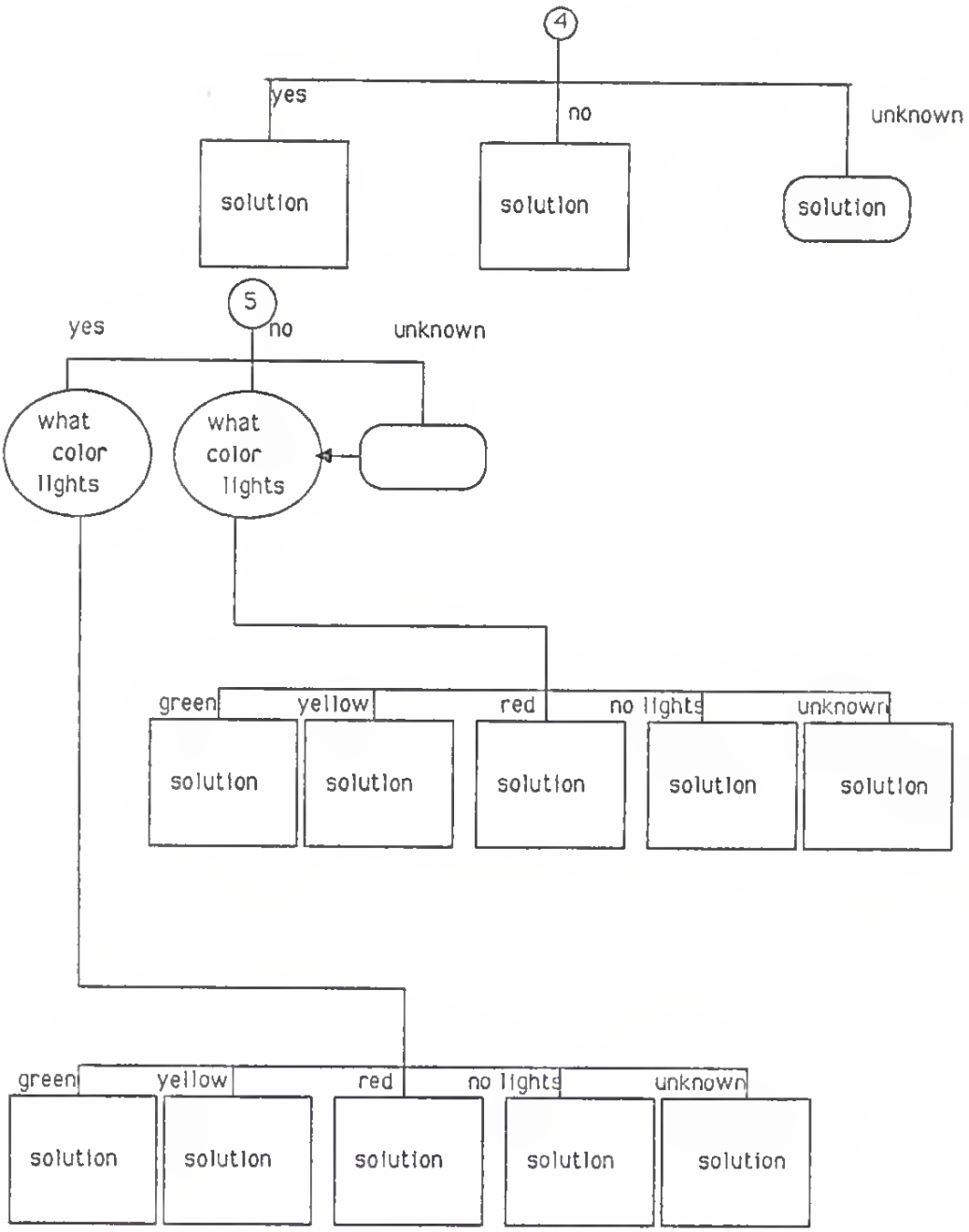
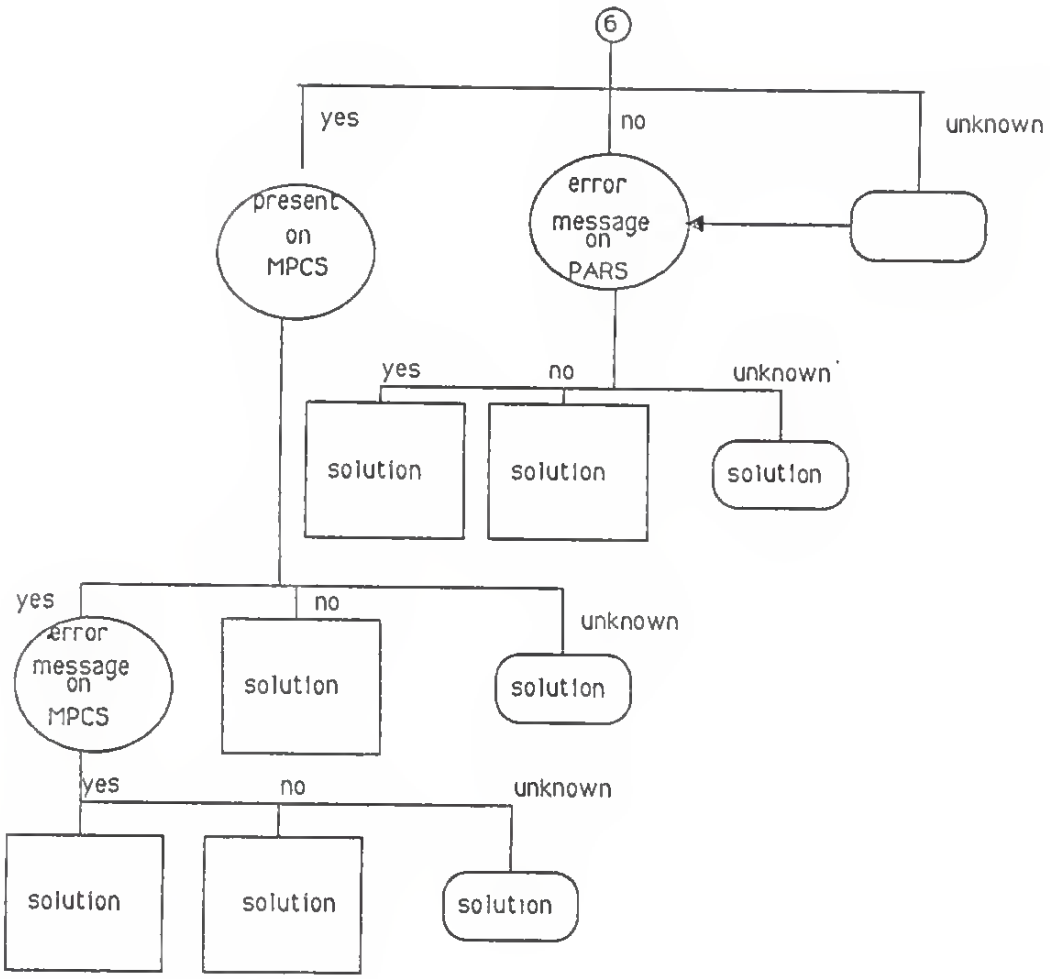

4
yes
no
unknown
solution
solution
solution
5
yes
no
unknown
what color lights
what color lights
green
yellow
red
no lights
unknown
solution
solution
solution
solution
solution
green
yellow
red
no lights
unknown
solution
solution
solution
solution
solution

Figure 5-6


6
yes
no
unknown
present on MPCS
error message on PARS
yes
no
unknown
solution
solution
solution
yes
no
unknown
error message on MPCS
solution
solution
yes
no
unknown
solution
solution
solution

The next test was to determine the accuracy of the expert system. By having the system and the expert solve the same problem, it was possible to tell in many of the situations, that the final answer was the same for both parties. It was discovered in most cases, that the expert tended to use other information to vary the order of search for the problem. The expert would, for example, know there was a modification to the PC concentrator software earlier that week so he/she would start looking for problems from that piece of equipment. The expert system did not have this information and would use the same search sequence repeatedly. This allowed the expert to find the problem in less time in most cases. Another instance of the expert being "better", is in the case of a repeated problem. The expert would "learn" over a period of time to go directly to the source of the problem. The expert system did not have this learning capability and would have to traverse through each piece of equipment in series to find the fault. Even with its shortcomings, the expert system was felt to be very good at a step-by-step methodology for solving problems. The expert system wasn't as fast as the expert, but it would eventually get to the source of the problem.

The final criteria for testing the expert system was for it be usable by novices. The initial idea was for the operations team to use the expert system and see if: A) They could use it and B) They could solve problems with it. Unfortunately, due to some unforeseen circumstances this was not possible. Instead random computer people were chosen to use the system, given a problem scenario, and expected to come up with the recommended answer to the problem. Almost immediately it became evident that some of the questions were ambiguous and confusing. These questions were rewritten to make them understandable. During the testing the subjects were able to get the system to perform some unexpected actions. Careful observation led to the discovery of several side-effect problems which were corrected. The test subjects were able to shed light on the fact that it would not be possible to just take untrained personnel, and with the help of the expert system, expect them to be able to solve problems. The questions as well as the solutions required a certain amount of UNIX administration knowledge of the user. In places slang, acronyms, and abbreviations were used to explain fixes. The target of the expert system was changed from a novice to an operations type personnel, familiar with UNIX and computer workings in general. The expert system does not

require much knowledge about the IPM system but it does require quite a bit of computer skills.

The expert system runs and is syntactically correct, agrees with the experts in most cases, and is helpful in allowing a computer literate person to solve problems. The final version of the knowledge base is contained in Appendix C.

6. CONCLUSIONS

With the completion of the project it was possible to take a step back and see the accomplishments of the project. The areas to examine were what was learned, what should have been done differently, and what could be done in the future?

Several things were learned from this project. It is possible to take an off-the-shelf expert system environment package, and develop a usable expert system in a short amount of time. It was learned that manufacturing can use expert systems to help with operational problems. The knowledge base is the source of power in an expert system, not the inference mechanism. The final and most important thing that was learned is that expert systems are not software of the future, but that expert systems are here and usable now.

Having gone through the experience of building an expert system, there were items in the building of the system that could have, and maybe should have, been done differently. In M.1 when the user asks "why?", the

system responds with the rule it is trying to prove. If the user isn't familiar with the rule structure and variable names, the information will not be of much value. A better way to handle this situation is to use the explain feature of M.1. The explain feature allows text to be associated with a rule. When the user asks "why?", the text is displayed instead of the rule being worked on. This may have been a way to provide the user with more information about problem solving procedure without getting into the details of the system. Another area that may have been done a little differently, is the handling of the unknown answer from the user. The system is dependent on answers from the user to accomplish the task. It would have been helpful if, when the user selected the unknown answer, a message was displayed stating that this was unacceptable. The system would then loop around to the same question again.

The future of this system holds many possibilities. The first possibility is to use this system in a production environment and measure the amount of time saved for the designers by the use of the system. Another possibility includes the creation of an operations document that would explain to the user how to find the answers to questions that the system is asking. This manual would

hopefully alleviate the UNIX knowledge requirement that a user must have and allow almost anyone to solve problems with the expert system. Since the recovery from problems is usually accomplished by executing UNIX commands, it would be useful to let the system automatically execute UNIX shells that would effect the solution as opposed to telling an operator to do it.

The lessons that were learned, the mistakes that were made, and the things that could be done in the future all point to the success of the project. The project team and those around the project team have learned valuable information and insight about expert systems. Expert systems are beginning to appear in data centers, production control, and factory floor operations. It is possible that expert systems will become standard operating procedure in the future.

7. REFERENCES

[1] Bryant, N., Managing Expert Systems, Wiley and Sons, New York, 1988, p. 31

[2] Vesonder, G. T., "Rule-Based Programming in the UNIX (R) System", AT&T Technical Journal, January/February 1988, Volume 67 . Issue 1, p. 69

[3] Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., Building Expert Systems, Addison-Wesley, Reading, 1983, p. 6

[4] Waterman, D. A., A Guide to Expert Systems, Addison-Wesley, Reading, 1986, p. 154

[5] Teknowledge, Inc., M.1 Reference Manual, Palo Alto, 1986, p. 3-6

[6] Forsyth, R., Expert Systems, Principles and Case Studies, Chapman and Hill Computing, London, 1984, p. 153

[7] Vesonder, G. T., "Rule-Based Programming in the UNIX (R) System", AT&T Technical Journal, January/February 1988, Volume 67 . Issue 1, pp. 71 - 73

[8] Brachman, R. J., and Henig, F. H., "The Emergence of Artificial Intelligence Technology", AT&T Technical Journal, January/February 1988, Volume 67 . Issue 1, p. 4

[9] Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., Building Expert Systems, Addison-Wesley, Reading, 1983, p. 83

8. BIBLIOGRAPHY

Waterman, D. A., A Guide to Expert Systems, Addison-Wesley, Reading, 1986

Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., Building Expert Systems, Addison-Wesley, Reading, 1983

Weiss, S. M., and Kulikowski, C. A., A Practical Guide to Designing Expert Systems, Rowman and Allanheld, Totwa, 1984

Forsyth, R., Expert Systems, Principles and Case Studies, Chapman and Hill Computing, London, 1984

Brachman, R. J., and Henig, F. H., "The Emergence of Artificial Intelligence Technology", AT&T Technical Journal, January/February 1988, Volume 67 . Issue 1

Vesonder, G. T., "Rule-Based Programming in the UNIX (R) System", AT&T Technical Journal, January/February 1988, Volume 67 . Issue 1

Bryant, N., Managing Expert Systems, Wiley and Sons, New York, 1988

Teknowledge, Inc., M.1 Reference Manual, Palo Alto, 1986

9. APPENDIX A

MPCS/IPM TROUBLE REPORT

Date 05/13/88	Trouble Number 3	Time 8:00 A.M.

Priority A = Trivial
Priority B = User Affecting
Priority C = Critical

PRIORITY: B

Problem Initiated by: VICKIE BROWN, X82220, MULTI-LINE

Problem Assigned to: MARY GONZALES

Description of Problem:
AT TERMPREP 5 AND TERMPREP 8 THE CRT'S WERE "HUNG UP" DUE TO THE MODEMS BEING "OFF-LINE".

Date/Time of "Fix": 05/13, 8:20 A.M.

Duration of Problem: 20 MIN.

Description of Solution:
REPEATED MPCS START COMMANDS AT TERM PREP 5 AND 8 REINITIALIZED THE CRT.

Description of Permanent Corrective Efforts Required:

Basically, this problem was:

Hardware		Software (Local)		IPM Admin.	
Shop		Software (PRISM)		UNIX Admin.	
Network		System Engineering		Human Factors	

MPCS/IPM TROUBLE REPORT

Date 05/13/88	Trouble Number 4	Time 10:51

Priority A = Trivial
Priority B = User Affecting
Priority C = Critical

PRIORITY: B

Problem Initiated by: CHUCK WHEATLY - RED LINE, X3192 (END (S. OF L28)

Problem Assigned to:

Description of Problem:
FIXED HEAD GUN GREEN LIGHT KEEPS FLASHING. CANNOT GET RED SCAN LIGHT. GUNS 1 & 3 OF THE DAISY CHAIN WORK.

Date/Time of "Fix": 05/13/88, 1:00 P.M.

Duration of Problem: 2 HRS.

Description of Solution:
REPLACED LS6000 FIXED HEAD GUN.

Description of Permanent Corrective Efforts Required:

Basically, this problem was:

Hardware	X	Software (Local)		IPM Admin.	
Shop		Software (PRISM)		UNIX Admin.	
Network		System Engineering		Human Factors	

MPCS/IPM TROUBLE REPORT

Date 05/13/88	Trouble Number 6	Time 1:20 P.M.

Priority A = Trivial
Priority B = User Affecting
Priority C = Critical

PRIORITY: B

Problem Initiated by: LOIS & GARY

Problem Assigned to: MARY GONZALES

Description of Problem:
MODEM AT RED ENDLINE WOULD NOT COME UP.

Date/Time of "Fix": 05/13, 1:35

Duration of Problem: 15 MINUTES

Description of Solution:
CALLED MARY GONZALES AND SHE RESTARTED THE PROCESS

Description of Permanent Corrective Efforts Required:

Basically, this problem was:

Hardware		Software (Local)		IPM Admin.	
Shop		Software (PRISM)	X	UNIX Admin.	
Network		System Engineering		Human Factors	

10. APPENDIX B

10.1 Records Not Received From Tracking Point

If the tracking point is hooked to a PC concentrator, check and see if the buffer is filling up on the PC. If the buffer is not filling up, reboot the PC because the concentrator program has stopped. If the buffer is filling up on the PC, check and see if there are three green lights on the front of the modem connected to the PC. If there are not three green lights on the modem, depress the line-status button on the front of the modem until three green lights appear. If there are three green lights on the front of the modem, go to the 3B15 that the PC concentrator is wired to and stop and start the tracking process for that PC concentrator. If the tracking point is not hooked to a PC concentrator, check and see if the remote system that should be sending the records is running. If the remote system is not running, notify the remote system's system administrator of the problem. If the remote system is running, check and see if the communications line that the information should come across is busy. If the line is busy, kill any process that is running on that line to free it up. If it is not busy, make sure that the passwords and phone numbers match in the required files between the remote system and the 3B15 tracking machine.

10.2 Scan Gun Not Working

Pull the trigger of the scan gun and make sure the gun is producing a red beam. If the gun is not producing a red beam, make sure the Scanstar is plugged in and turned on. If the Scanstar is not plugged in and turned on, plug it in and turn it on. If the Scanstar is plugged in and turned on and the gun still produces no red beam, replace the scan gun with a known working gun from inventory. If the scan gun does produce a red beam but will not read the label, check and see if a red or yellow light is present on the front of the Scanstar. If the light on the front of the Scanstar is not yellow or red, replace the label on the board with a duplicate label. If the light on the font of the Scanstar is yellow or red, turn off the Scanstar and turn it back on. Scan another label and see if the yellow or red light reappear. If the yellow or red light does not reappears, problem is corrected. If the yellow or red light does reappear, turn the Scanstar off and on again, then make sure that all Scanstars that are wired in series with the problem Scanstar show green lights on the front panel. If all Scanstars show green lights and after scanning a label the problem Scanstar shows red or yellow, turn off and back on all the Scanstars that are wired in series. Reboot the PC concentrator causing the concentrator program to restart.

10.3 Terminal Not Working

Check the modem attached to the terminal and see if there are three green lights on the front. If there are not three green lights on the front of the modem, depress the line-status button until three green lights appear. If there are three green lights on the front of the modem, turn terminal off and back on. Go to the 3B15 that the terminal is wired to and stop and start the process for that terminal. If the problem exists on ALL terminals on ALL lines, stop and start ALL processes on ALL machines.

10.4 Slow Scan Gun Response

If the scan gun is working but the time between the reading of the label and the confirmation tone is long, turn off the monitor on the PC concentrator.

10.5 Select Not Found

If the terminal is working but the select is not found for processing, check and see if the select has been released from IMPAC. If the select has not been released from IMPAC, call the storeroom and get them to release it. If the select has been released from IMPAC, check and see if the select is present in the receive directory on the SFC machine. If the select is not present on the SFC machine, stop and start the RJE process between the SFC machine and the IBM. If the select is present on the SFC machine, check and see if the select has been processed by the PARS machine. If the select has not been processed by PARS, see if there are any error messages about that select on PARS. If there are no error messages on PARS about the select, stop and start the communications link between SFC and PARS. If there are error messages on PARS about the select, add the code/series combination of the select to the PARS database and manually release the select. If the select has been processed by PARS, check and see if the select is in the receive directory of MPCS. If it is not in the receive directory of MPCS, stop and start the communications link between PARS and the MPCS machine. If it is in the receive directory of MPCS, check and see if there are any error messages about the select on MPCS. If there are not any error messages about the select on MPCS, stop and start the auto-orig process. If there are error messages about the select on MPCS, add the code/series combination to the MPCS database and manually input the select.

10.6 Cannot Access Output Reports

Check and see if all lines are affected by the problem. If all lines have the problem, reboot the LAN bridge that connects the IPM network to the user network. If the problem is not on all lines find out how many users on a particular line are affected. If more than one user on a particular line is having the problem, reboot the 3B2/400 server machine for that line. If it is only one user that is having the problem, reboot the PC that the user is trying to access the reports on.

10.7 Output Reports Not Updated

If the user can access the reports but the user's screen has not updated in twenty minutes, check and see if the user's PC is still connected to the file server. If the PC is not connected to the file server, reboot the PC and login to the file server again. If the PC is still connected to the file server, check and see if the file server is running the receive transaction process. If the receive transaction process is not running on the server, start the process. If the receive transaction process is running on the server, check and see if the send transaction process is running on the MPCS tracking machine. If the send transaction process is not running, start the process. If the send transaction process is running, kill any process on the communications link between the tracking machine and the server.

11. APPENDIX C

```
/*
This knowledge base is used to try and solve IPM
operational problems at the Denver Works.  The
system will not solve all problems but it is hoped
that it will be able to handle the
daily recurring problems.  The system uses M.1 by
Teknowledge, Inc. as the inference engine and was
designed by James R. Watson
*/


/*
Try to find out as much information as possible before
starting the actual consultation.
*/

initialdata = [start-text-displayed, line-affected,
               problem-type,problem-answered,next-step].


/*
Determine what circuit pack lines operations are being
affected by the problem.
*/

multivalued(line-affected).
question(line-affected) =
         "What circuit pack lines are affected by the
                                    problem?".
legalvals(line-affected) = [red, orange, merlin, all].
automaticmenu(line-affected).
enumeratedanswers(line-affected).


/*
Determine type of problem.
*/

multivalued(problem-type).
question(problem-type) =
        "What type of problem is it? ".
legalvals(problem-type) = [input, output].
```

```
/*
Determine if the user can see the problem.
*/

multivalued(user-affecting).
question(user-affecting) =
        "Has the user noticed the problem? ".
legalvals(user-affecting) = [yes, no].


/*
This is the question to find out if three green lights
are visible on the front of the modem.
*/

multivalued(three-lights-on).
question(three-lights-on) =
        "Are there three green lights lit on the front
                                of the modem?".
legalvals(three-lights-on) = [yes, no].


/*
Find out if red beam comes on when trigger is pulled.
*/

question(red-beam) =
        "When you pull the trigger on the gun do you get
                                        a red beam? ".
legalvals(red-beam) = [yes, no].



/*
Find out what lights are glowing on the scanstar.
*/

multivalued(light-color).
question(light-color) =
     "What color is the light on front of the scanstar?".
legalvals(light-color) = [green,yellow,red,no-light-on].
automaticmenu(light-color).
enumeratedanswers(light-color).
```

```
/*
Find out what particular piece of hardware is having
a problem.
*/

multivalued(input-part).
question(input-part) =
        "What piece of equipment on the input stream
                                is not working?".
legalvals(input-part) = [gun, terminal, tracking-point].
automaticmenu(input-part).
enumeratedanswers(input-part).


/*
This question determines if a concentrator is used for
this tracking point.
*/

question(through-pc) =
        "Does this tracking point go through a PC
                                        concentrator?".
legalvals(through-pc) = [yes,no].


/*
Determine if the remote processor that sends transactions
to the traking 3B15 is up and running.
*/

question(remote-down) =
        "Is the processor that the input records come
                                 from running?".
legalvals(remote-down) = [yes,no].


/*
Find out if scanning transactions are being buffered up
at the PC and not sent to the tracking 3B15.
*/

question(buffer-filling) =
        "Is the PC concentrator buffering scan
                                transactions?".
legalvals(buffer-filling) = [yes,no].
```

```
/*
Determine if the line used to pass information between
the remote processor and the tracking 3B15 is busy.
*/

question(line-busy) =
        "When you dial the phone number of the line used
for remote transactions, do you get a tone or a
busy signal?".
legalvals(line-busy) = [tone, busy-signal].
automaticmenu(line-busy).
enumeratedanswers(line-busy).



/*
This question determines if the scan gun is still able
to read barcode labels.
*/

question(gun-working) =
        "Is the scan gun reading labels?".
legalvals(gun-working) = [yes, no].


/*
This question determines if there is an abnormally long
pause between label scan and confirmation.
*/

question(slow-gun-response) =
        "Is there a long pause between reading a label
         and the confirmation tone?".
legalvals(slow-gun-response) = [yes, no].



/*
This question determines if the terminal is currently
operational.
*/

question(terminal-working) =
        "Does the terminal responsed when the return
         key is depressed?".
legalvals(terminal-working) = [yes, no].
```

```
/*
This question determines if the usert is trying to input
a select for preparation of scanning.
*/

question(select-not-found) =
        "Is the user trying to input a select and getting
         a "Select not Found" error message?".
legalvals(select-not-found) = [yes, no].


/*
This question determines if the select has been released
from IMPAC yet.
*/

question(impac-release) =
        "Has the select been released from IMPAC yet?".
legalvals(impac-release) = [yes, no].


/*
This question determines if the select made it to the SFC
machine.
*/

question(in-sfc) =
        "Is a copy of the select file present on the SFC
          machine?".
legalvals(in-sfc) = [yes, no].


/*
This question determines if the select made it to the
MPCS machine.
*/

question(in-mpcs) =
        "Is a copy of the select file present on the
          MPCS machine?".
legalvals(in-mpcs) = [yes, no].


/*
This question determines if the select has been processed
```

```
by the PARS machine yet.
*/

question(processed-by-pars) =
  "Has the select been processed by the PARS machine?".
legalvals(processed-by-pars) = [yes, no].


/*
This question determines if PARS produced any error
messages on why the select was not processed.
*/

question(error-pars) =
        "Is there an error message on PARS stating why
         the select was not processed?".
legalvals(error-pars) = [yes, no].


/*
This question determines if PARS produced any error
messages on why the select was not processed.
*/

question(error-mpcs) =
        "Is there an error message on MPCS stating why
         the select was not loaded onto the database?".
legalvals(error-mpcs) = [yes, no].


/*
This question determines the exact nature of the ouput
problem.
*/

question(output-source) =
  "Is the problem with the ouput that the user cannot
  access reports or that the reports are not updating?".
legalvals(output-source) = [no-access, no-update].
automaticmenu(output-source).
enumeratedanswers(output-source).


/*
This question determines the number of users that are
```

```
affected by the output problem.
*/

question(more-than-one) =
    "Is there MORE THAN ONE user affected by the output
     problem?".
legalvals(more-than-one) = [yes,no].


/*
This question determines if the transaction receive
process is still running on the server machine.
*/

question(trans-rec-running) =
        "Is the transaction receiving process running on
         the 3B2 server?".
legalvals(trans-rec-running) = [yes, no].


/*
This question determines if the transaction send process
is still running on the processing 3B15.
*/

question(trans-send-running) =
        "Is the transaction send process running on the
         3B15 machine?".
legalvals(trans-send-running) = [yes, no].



/*
************************************************
*/



/*
Display the openning banner.
*/

nocache(start-text).

rule-0 : if start-text = TEXT and
```

```
            display(TEXT)
            then start-text-displayed.


/*
Set up condition for end of consultation.
*/

rule-1 : if problem-answered
            then do(abort).



/*
If fix to problem was displayed set exiting condition.
*/

rule-2 : if input-problem and
            input-part-known and
            specific-problem-known and
            message(input-part, specific-problem,
                                    line-affected)
            then problem-answered.



/*
Set type of trouble indicator.
*/

rule-3 : if problem-type = input
            then input-problem.

rule-4 : if problem-type = output
            then output-problem.

rule-5 : if problem-type is unknown
            then unknown-problem.



/*
If it is a gun problem check and see if red beam works.
*/
```

```
rule-6 : if input-problem and
            input-part = gun and
            not(gun-working) and
            not(red-beam) and
            not(power-on)
            then specific-problem = power.

rule-7 : if input-problem and
            input-part = gun and
            not(gun-working) and
            not(red-beam) and
            power-on
            then specific-problem = badgun.

rule-8 : if input-problem and
            input-part = gun and
            not(gun-working) and
            red-beam
            then check-lights.



/*
Determine if power is on.
*/

rule-9 : if light-color = green or
            light-color = yellow or
            light-color = red
            then power-on.

rule-10 : if light-color = no-light-on or
            light-color = unknown
            then power-on = no.



/*
If red beam is working look for other cause of gun
problem.
*/

rule-11 : if check-lights and
             light-color = green
             then specific-problem = label.

rule-12 : if check-lights and
```

```
                not(light-color = green)
                then specific-problem = busy.


/*
This is a general message routine for display
different answers to problems.
*/

nocache(message(input-part, specific-problem,
                                    line-affected)).

rule-13 : if input-problem and
                input-part = I and
                specific-problem = S and
                line-affected = A and
                display(") and
                message-text(I, S, A) = LIST and
                display(LIST)
                then message(input-part, specific-problem,
                                        line-affected).


/*
Determine if the piece of equipment with a
problem has been found.
*/

rule-14 : if input-part = ANYPART
                then input-part-known.


/*
Determine if the exact problem is known.
*/

rule-15 : if specific-problem = ANYPROBLEM
                then specific-problem-known.


/*
The following rules determine what the specific problem
for a terminal is.
```

```
*/

rule-16 : if input-problem and
             input-part = terminal and
             not(terminal-working) and
             not(line-affected = all) and
             not(three-lights-on)
             then specific-problem = dropped-line.

rule-17 : if input-problem and
             input-part = terminal and
             not(terminal-working) and
             line-affected = all and
             not(three-lights-on)
             then specific-problem = switch-hit.

rule-18 : if input-problem and
             input-part = terminal and
             not(terminal-working) and
             not(line-affected = all) and
             three-lights-on
             then specific-problem = hung-process.

rule-19 : if input-problem and
             input-part = terminal and
             not(terminal-working) and
             line-affected = all and
             three-lights-on
             then specific-problem = switch-hit.


/*
These rules handle determining the problem if it is with
a tracking station.
*/

rule-20 : if input-problem and
             input-part = tracking-point and
             not(line-affected = all) and
             through-pc and
             buffer-filling and
             not(three-lights-on)
             then specific-problem = pc-line-drop.

rule-21 : if input-problem and
             input-part = tracking-point and
             not(line-affected = all) and
             through-pc and
```

```
                buffer-filling and
                three-lights-on
                then specific-problem =
                          hung-tracking-process.

rule-22 : if input-problem and
                input-part = tracking-point and
                not(line-affected = all) and
                through-pc and
                not(buffer-filling)
                then specific-problem = concentrator-stop.

rule-23 : if input-problem and
                input-part = tracking-point and
                line-affected = all
                then specific-problem = switch-hit.

rule-24 : if input-problem and
                input-part = tracking-point and
                not(line-affected = all) and
                not(through-pc) and
                not(remote-down)
                then specific-problem = remote-failure.

rule-25 : if input-problem and
                input-part = tracking-point and
                not(line-affected = all) and
                not(through-pc) and
                remote-down and
                line-busy = tone
                then specific-problem = bad-password.

rule-26 : if input-problem and
                input-part = tracking-point and
                not(line-affected = all) and
                not (through-pc) and
                remote-down and
                not(line-busy = tone)
                then specific-problem = busy-line.


/*
These rules determine if the problem is a slow scan
gun.
*/

rule-27 : if input-problem and
              input-part = gun and
```

```
            gun-working and
            slow-gun-response
            then specific-problem = slow-gun.

rule-28 : if input-problem and
            input-part = gun and
            gun-working and
            not(slow-gun-response)
            then specific-problem = trivial-gun.


/*
These rules determine where the problem has occurred
causing a select not to be down-loading into the
MPCS system.
*/

rule-29 : if input-problem and
            input-part = terminal and
            terminal-working and
            select-not-found and
            not(impac-release)
            then specific-problem = no-release.

rule-30 : if input-problem and
            input-part = terminal and
            terminal-working and
            select-not-found and
            impac-release and
            not(in-sfc)
            then specific-problem = rje-link.

rule-31 : if input-problem and
            input-part = terminal and
            terminal-working and
            select-not-found and
            impac-release and
            in-sfc and
            not(processed-by-pars) and
            not(error-pars)
            then specific-problem = sfc-pars-link.

rule-32 : if input-problem and
            input-part = terminal and
            terminal-working and
            select-not-found and
```

```
                impac-release and
                in-sfc and
                not(processed-by-pars) and
                error-pars
                then specific-problem = pars-stop.

rule-33 : if input-problem and
                input-part = terminal and
                terminal-working and
                select-not-found and
                impac-release and
                in-sfc and
                processed-by-pars and
                not(in-mpcs)
                then specific-problem = pars-mpcs-link.

rule-34 : if input-problem and
                input-part = terminal and
                terminal-working and
                select-not-found and
                impac-release and
                in-sfc and
                processed-by-pars and
                in-mpcs and
                not(error-mpcs)
                then specific-problem = auto-orig-hung.

rule-35 : if input-problem and
                input-part = terminal and
                terminal-working and
                select-not-found and
                impac-release and
                in-sfc and
                processed-by-pars and
                in-mpcs and
                error-mpcs
                then specific-problem = mpcs-stop.

rule-36 : if input-problem and
                input-part = terminal and
                terminal-working and
                not(select-not-found)
                then specific-problem = terminal-trivial.


/*
```

```
This rule handles finding a solution for output type
of problems.
*/

rule-37 : if output-problem and
             output-source-known and
             specific-problem-known and
             message(output-source, specific-problem,
                                        line-affected)
             then problem-answered.


/*
This rule sets the problem known indicator when the
source of the problem is discovered.
*/

rule-38 : if output-source = ANYSOURCE
             then output-source-known.


/*
These rules handle the situation when the output problem
is that the user cannot access the reports.
*/

rule-39 : if output-problem and
             output-source = no-access and
             line-affected = all
             then specific-problem = bridge-down.

rule-40 : if output-problem and
             output-source = no-access and
             not(line-affected = all) and
             one-user-affected
             then specific-problem = pc-hung.

rule-41 : if output-problem and
             output-source = no-access and
             not(line-affected = all) and
             multiple-users-affected
             then specific-problem = server-hung.


/*
These rules determine how many users are affected by the
```

```
output problem.
*/

rule-42 : if not(more-than-one)
             then one-user-affected.

rule-43 : if more-than-one
             then multiple-users-affected.


/*
This rule handles the outputting of the solution for
output type of problems.
*/

nocache(message(output-source, specific-problem,
                                    line-affected)).

rule-44 : if output-problem and
             output-source = I and
             specific-problem = S and
             line-affected = A and
             display(") and
             message-text(I, S, A) = LIST and
             display(LIST)
             then message(output-source,
                   specific-problem,line-affected).


/*
These rules handle the situation when the output
problem is that the reports are not being update.
*/

rule-45 : if output-problem and
             output-source = no-update and
             line-affected = all
             then specific-problem = bridge-down.

rule-46 : if output-problem and
             output-source = no-update and
                 not(line-affected = all) and
                 not(trans-rec-running)
                 then specific-problem = server-stop.

rule-47 : if output-problem and
             output-source = no-update and
```

```
                not(line-affected = all) and
                trans-rec-running and
                not(trans-send-running)
                then specific-problem = send-process-stop.

rule-48 : if output-problem and
                output-source = no-update and
                not(line-affected = all) and
                trans-rec-running and
                trans-send-running
                then specific-problem = process-output-link.


/*
These rules handle the situation when the operator
does not have enough information for the system.
*/

rule-95 : if output-problem and
                output-source is unknown and
                    no-output = TEXT and
                    display(") and
                    display(TEXT)
                    then problem-answered.

rule-96 : if unknown-problem and
                no-problem = TEXT and
                display(") and
                display(TEXT)
                then problem-answered.

rule-97 : if input-problem and
                input-part is unknown and
                no-input-part = TEXT and
                display(") and
                display(TEXT)
                then problem-answered.

rule-98 : if line-affected is unknown and
                no-line-affected = TEXT and
                display(") and
                display(TEXT)
                then problem-answered.
```

```
/*
Decide if there is another problem to solve.
*/

rule-99 : if next-objective = quit-now or
             next-objective is unknown and
             display(") and
             display("Thank you for using the IPM
                              SOLUTION HELPER.") and
             do(abort)
             then next-step.

rule-100 : if next-objective = start-again and
              display(") and
              do(reset) and do(restart)
              then next-step.

question(next-objective) =
          "Do you have another problem to fix?

     quit-now, I am finished.
     start-again, I have another problem.".
legalvals(next-objective) = [quit-now, start-again].
enumeratedanswers(next-objective).


/*
**********************************************
*/


/*
The following are the messages that are displayed
for the fix to a problem.
*/

noautomaticquestion(message-text(I, S, A)).

nocache(mesage-text(I, S, A)).

message-text(gun, power, ANY) = [nl,
"The gun appears to not be getting power from the",
" scanstar. Check and make sure the scanstar is",
" plugged in and turned on. If it is plugged in",
" and turned on and there are still no ",
"lights visible on the front of the scanstar there is",
" a chance that it is a faulty scanstar.  Replace the",
```

```
" scanstar with a known working unit and see if the",
" problem clears.",nl,nl,
"If the problem still exists after these actions ",
"contact a design team member.",nl].

message-text(gun, badgun, ANY) = [nl,
"If there is no red beam being produced by the gun",
" chances are that this is a faulty gun.  Replace gun",
" with a known working gun and see if problem clears.",
nl,nl,
"If the problem still exists after these actions ",
"contact a design team member.",nl].

message-text(gun, label, ANY) = [nl,
"It seems that the equipment is in working condition ",
"the problem could be with the label on the board.",
"Generate a duplicate label and replace the current",
"label on the board with the new label and see if the",
"problem clears.",nl,"If the problem exists after",
"these actions contact ",
"a design team member.",nl].

message-text(gun, busy, ANY) = [nl,
"The gun has been set to an inactive state. There is a",
"three step process to correcting this problem.",nl,
"1. Turn off and back on the scanstar with the problem",
"gun.2. Turn off and back on all the scanstars that ",
"are wired in series with the  problem gun.",nl,
"3. Reboot the PC that the gun is wired into causing ",
"the concentrator program    to restart.",nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(gun, slow-gun, ANY) = [nl,
"It appears the monitor on the PC concentrator is on.Go",
"to the PC concentrator that this gun is wired into and",
"turn it off.  This should speed up the response time of",
"the gun.", nl,nl,
"If the problem still exists after these actions contact",
"a design team member.",nl].

message-text(gun, trivial-gun, ANY) = [nl,
"Everything seems to be working correctly, so this",
"problem may be considered a trivial problem. ",
"Inform the user that the problem will be added to",
"the trivial problem list and",
"looked at at a later date.",nl,nl,
"If the user is unhappy with this action contact ",
"a design team member.",nl].
```

```
message-text(terminal, dropped-line, A) = [nl,
"It seems the connection between the terminal and"
"the",A," line has been dropped. Depress and release",
"the line-status button on the front of the modem",
" until three green lights",
"appear.  The terminal should begin within 60",
" seconds.",nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(terminal, hung-process, A) = [nl,
"It seems that the process on this terminal has stopped",
"running.  Turn the terminal off and on.  Go to the",
"3B15 running the",A,"line and stop and start the",
" terminal process.",nl,nl,
"If the problem exists after these actions contact ",
"a design team member.",nl].

message-text(terminal, switch-hit, ANY) = [nl,
"It seems that the System 85 has gone down.  Go to all",
"3B15s and stop and restart ALL processes.",
nl,nl,
"If the problem exists after these actions contact ",
"a design team member.",nl].

message-text(terminal, no-release, ANY) = [nl,
"It seems that the select has not been released from",
"IMPAC yet. Call the storeroom and ask them to release",
"the select because the material has been delivered.",
nl,nl, "If the problem still exists after these actions",
" contact a design team member.",nl].

message-text(terminal, rje-link, ANY) = [nl,
"It seems that the rje link between the SFC machine and",
" the IBM is not functioning.  Stop and start the rje",
" process on the SFC machine, then call the data center",
" and have them stop and start the rje process on the",
" IBM.",nl,nl,
"If the problem exists after these actions contact ",
"a design team member.",nl].

message-text(terminal, sfc-pars-link, ANY) = [nl,
"It seems that the link between the SFC machine and",
"the PARS machine is not working. KIll any process that",
" is running on that link to clear the problem.",
nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].
```

```
message-text(terminal, pars-stop, ANY) = [nl,
"It seems that the PARS machine has flagged and error",
"on this select.  Add the code/series combination",
"from this select to the PARS database and manually",
"release the select.", nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(terminal, mpcs-stop, A) = [nl,
"It seems that the ", A," MPCS machine has flagged and",
" error on this select.  Add the code/series/vintage",
"combination from this select to the",A,"MPCS database.",
" The select should process within the next",
" 5 minutes.", nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(terminal, pars-mpcs-link, A) = [nl,
"It seems that the link between the PARS machine and",
" the ", A," MPCS machine is not working.  KIll any",
" process that is running on that link to clear the",
" problem.", nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(terminal, auto-orig-hung, A) = [nl,
"It seems that the auto orig process on the ", A ,
 " MPCS machine is not running.  Stop and start the",
" auto orig process on ", A, " machine to clear the",
" problem.", nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(terminal, terminal-trivial, ANY) = [nl,
"Everything seems to be working correctly, so this",
" problem may be considered a trivial problem.  Inform",
" the user that the problem will be added to the trivial",
" problem list and looked at at a later date.",nl,nl,
"If the user is unhappy with this action contact ",
"a design team member.",nl].

message-text(tracking-point, switch-hit, ANY) = [nl,
"It seems that the System 85 has gone down.  Go to all",
"3B15s running MPCS and stop and restart ALL processes.",
nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(tracking-point, bad-password, A) = [nl,
```

```
"It seems that the password file between the",A,"3B15",
"and the remote processor do not match.  Make sure ",
" the remote processor has the current passwords and ",
" phone numbers for the ", A, " MPCS tracking machine.",
nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(tracking-point, busy-line, A) = [nl,
"It appears the remote transaction line between the",
" remote processor and the",A," machine is busy.",
"Issue a kill command on any process running on the",
" TTY of the remote processor transaction line. ",
nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(tracking-point, pc-line-drop, A) = [nl,
"It seems that the connection between the concentrator",
" and the ", A, " line has been dropped.  Depress and",
" release the line-status button on the front of the ",
" modem until three green lights appear.  The ",
"concentrator should begin operating in 60 seconds."
,nl,nl, "If the problem exists after these actions",
" contact a design team member.",nl].

message-text(tracking-point, hung-tracking-process, A)
= [nl, "It seems that the process between the",
" concentrator and the",A," 3B15 has stopped running.",
"  Go to the ",A,"3B15 tracking machine and stop and",
" start the tracking process connected to the problem",
" concentrator.",nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(tracking-point, concentrator-stop, ANY)=[nl,
"It appears that the concentrator program has stopped",
" running on the PC.  Reboot the PC causing the",
" concentrator to restart. ",nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(tracking-point, remote-failure , A) = [nl,
"It appears that the remote processor that is sending",
" transactions to the ", A, " tracking machine has gone",
" down.  Notify the remote system administrator",
" of the possible problem.",nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].
```

```
message-text(no-access, bridge-down, ANY) = [nl,
"It appears that the bridge that connects IPM to the",
" user network has gone down.  Reboot the bridge device",
" by turning it off and back on. ",
nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(no-access, pc-hung , ANY) = [nl,
"It appears that the particular PC that the user is",
" working on has been disconnected from the network.",
"  Reboot the PC by turning it off and back on or by",
" depressing the reset button.",nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(no-access, server-hung , A) = [nl,
"It appears that the",A,"3B2 server has malfunctioned.",
" Notify all users on the",A,"machine that the server",
"will be out of operation for approximately 30 minutes.",
"Perform a "shutdown -y -g0 -i6" on the",A,"server.",
nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(no-update, bridge-down, ANY) = [nl,
"It appears that the bridge that connects IPM to the",
"user network has gone down.  Reboot the bridge device",
" by turning it off and back on. ",
nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(no-update, server-stop, A) = [nl,
"It appears that the transaction receive process has",
" stopped running on the ", A, " server causing the",
" reports not to be updated.  Start the transaction",
" receive process on the ", A, " 3B2 server.",nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(no-update, send-process-stop, A) = [nl,
"It appears that the transaction send process has",
"stopped running on the",A,"3B15 processing machine",
" causing the reports not to be updated.  Start the",
" transaction send process on the",A,"3B15 machine.",
nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].
```

```
message-text(no-update, process-output-link, A) = [nl,
"It appears that the link between the",A,"3B2 server"
" and the",A," 3B15 processing machine has gotten",
" hung up. Kill any processes running on that link",
" freeing the hang.", nl,nl,
"If the problem exists after these actions contact",
"a design team member.",nl].

message-text(I, S, ANY) = [nl,
"You are missing an output message for ",I,
" and ",S,nl].

no-input-part = [nl,
"You have not given the sytem an input part, it needs",
"this information to figure out what the problem is.",
"Please find the answer to the asked question and try",
"again.",nl,nl, "If you are unable to find the answer",
"to the question please contact a design team member."
,nl].

no-output = [nl,
"You have not given the sytem the type of output",
"problem. The system needs this information to figure",
"out what the problem is. Please find the answer to ",
"the asked question and try again.",nl,nl,
"If you are unable to find the answer to the question ",
"please contact a design team member.",nl].

no-line-affected = [nl,
"You have not given the sytem a line that was affected",
"by the problem. This information is needed to figure",
"out what the problem is. Please find the",
"answer to the asked question and try again.",nl,nl,
"If you are unable to answer the question ",
"please contact a design team member.",nl].

no-problem = [nl,
"You have not given the sytem type of problem, it ",
"needs this information to figure what to do next.",
"Please find the answer to the asked question and try",
" again.",nl,nl,
"If you are unable to find the answer to the question ",
"please contact a design team member.",nl].


start-text  = [nl,
"       ***Welcome to the IPM SOLUTION HELPER***",
nl,nl,"This system is designed to aid in the process",
"of solving IPM operational problems. By supplying the",
```

```
"system with answers to questions the system will",
"give you recommended courses of action to solve the",
"problem.  Please pick your answer to the question from",
"the menu options provided with each question.",
nl,"Good Luck and Happy Problem Fixing!"].
```

Using Rule-Based Software to Debug Factory Automation Systems

by

James R. Watson

B.S., Colorado State University, 1982

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing & Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

The Integrated Pull Manufacturing (IPM) System is made up of over twenty processors, hundreds of programs, fifty bar-code readers, several networks and many users. The system was designed by a three member team over a two year span. The short time frame left no time to develop documentation for operational procedures on how the system works or how to recover from a problem. What was needed was a way to transfer knowledge from the designers to a less technical operations team.

A rule-based software package was used to develop a repository for the designers knowledge that would give the operations personnel a step-by-step process for correcting problems in IPM. As a problem became a recurring operations issue it would be entered into the system along with a detailed description of how to correct it. This would allow the operations personnel to handle the majority of problems and the designers would only be involved if it was a unique situation that occurred. The rule-based software would become part of the IPM system which would allow the knowledge of the system to remain with the system and not with the people who designed it.

Although it is on a small scale and what may be considered a prototype, the rule based software demonstrated that manufacturing can begin to use expert systems now to solve operational problems. As manufacturing becomes more and more dependent on computer systems and expert systems advance in ease of use and implementation there is a window of possibility for the use of knowledge based systems to become the norm as opposed to the leading edge.