MASTER IN COMPUTER SCIENCE

Spool scheduling and expert systems

Scheen, Benoit

Award date:
1990

Awarding institution:
University of Namur

SPOOL SCHEDULING
and
EXPERT SYSTEMS

by **Benoit SCHEEN**

Advisor : Jean Ramaekers

A thesis submitted in
conformity with the requirements
for the degree of
"Licencié et Maître en Informatique"

Facultés Notre-Dame de la Paix
Institut d'Informatique
Namur - September 1990

First of all, I would like to express my greatfulness to Mr Philippe Dumont from Siemens Software S.A., who has devoted a lot of his precious time to the follow-up of this work. His practical experience as well as pedagogical skills were extremely helpful.

Secondly, I wish to thank all the people of the SWN11 team from Siemens Software S.A. and the VS2414 team from Siemens-Perlach (Munich - West Germany) for the support they have brought during the training period in Rhisnes and my stay in Munich.

Next, I would like to express my thankfulness to Mr Jean Ramaekers, professor at the "Institut d'Informatique" of the University of Namur, who has consented to guide this work and who has offered me such an interesting training.

Finally, I would love to thank Barbara for her precious english skills and for her constant support throughout the completion of this work.

ABSTRACT

For a long time, spool scheduling problems have been solved through the use of conventional programming techniques. However, in order to offer new functionalities and to improve the quality of service of the spools systems, other programming techniques may be considered. Among these techniques, we have chosen the expert system approach. As a matter of fact, this approach seems to be particularly adapted for the development of performant spool systems. This thesis thus draws a prototype of a spool scheduler expert system. It also studies the validity of such an approach for the development of a commercial product.

RESUME

Depuis longue date, les problèmes de *spool scheduling* ont été résolus grâce à des techniques de programmation traditionnelles. Cependant, dans le but d'améliorer la qualité de service des spools, d'autres techniques de programmation peuvent être envisagées. Parmi celles-ci, nous avons retenu l'approche des systèmes expert. Cette approche nous semble en effet particulièrement adaptée à une amélioration qualitative des spools. Dans cet ordre d'idée, ce mémoire présente un prototype de système expert dédié au *spool scheduling*. Il s'étend également sur la validité d'une telle approche pour le développement d'un produit commercial.

TABLE OF CONTENTS

SECTION 2 : EXISTING EXPERT SYSTEMS IN THE COMPUTER SYSTEM DOMAIN

SECTION 3 : A PROTOTYPE OF A SPOOL SCHEDULER EXPERT SYSTEM

TABLE OF FIGURES

INTRODUCTION

Over the past several decades, society has had an infatuation with trying to breathe life or intelligence into machines. We no longer want computers to just add, subtract, multiply or divide, but to act human, to think. Imagine the endless possibilities of intelligent machines : computer systems that recommend profitable financial and marketing strategies, create new designs in the automobile or semi conductors industries, or quickly monitor and diagnosis a patient's health. An entirely new research effort dedicated to the development of artificial intelligence evolved, growing in significance to a become virtual growth industry in today's world.

Expert systems - special-purpose computer programs using expert knowledge to attain high level performances in a narrow problem area - probably constitute the "hottest" topic in artificial intelligence today. The expert system technology, limited to academic laboratories in the 1970s, is now becoming cost-effective and is beginning to enter into commercial applications.

Several dozens of expert systems have already been developed in the fields of medicine, chemistry, geology, manufacturing, ... For a while, however, a new research effort focuses on the development of expert systems dedicated to the computer system domain. Our work is integrated in this new approach since its main goal was to develop a prototype of a spool scheduling expert system.

The present thesis is therefore divided in three sections.

The purpose of the first section is to give an introduction to the fundamental concepts and basic issues in expert system research. Since this thesis is entirely dedicated to the use and the development of expert systems, the first section will offer a methodological context to the reader.

The second section will give a short overview of the existing expert systems dedicated to the computer system domain. As our prototype meddles in the development of expert systems dedicated to computer systems, this second section will enable us to better understand the validity of this new approach.

In the third section, the detailed analyse of our four-months' work will illustrate the theoretical concepts described in the first section. The first chapter of this section focuses on a brief overview of the existing expert systems developed in the area of scheduling problems. The second chapter gives rudimentary knowledges about the structure of a spool. The final five chapter apply the general methodology for building expert systems to our problem.

SECTION 1

INTRODUCTION TO EXPERT SYSTEMS

Since our thesis is entirely dedicated to the use and the development of expert systems, it is necessary to offer a methodological context to the reader. Therefore, the purpose of this first section is to give an introduction to the fundamental concepts and basic issues in expert system research.

CHAPTER 1 : WHAT ARE EXPERT SYSTEMS ?

1.1 THE ARTIFICIAL INTELLIGENCE CONTEXT

1.1.1 Artificial intelligence domains

Artificial Intelligence is a scientific field concerned with the creation of computer systems which can achieve human levels of reasoning. More precisely, artificial intelligence is the branch of information science that focuses on developing computer programs able to perform tasks normally associated with intelligent human behaviour.

The science of artificial intelligence is rather broad and has, in effect, been explored from the beginning of time. The modern beginning can perhaps be dated from 1956 when Claude Shannon and Marvin Minsky met at Dartmouth College with other information science pioneers for the unveiling of the world's first expert system - Allen Newell's Logic Theorist.

Today, the science of artificial intelligence spans a growing list of emerging disciplines [TOW-86] : knowledge representation, problem solving, natural language interfaces, vision, robotics and expert systems.

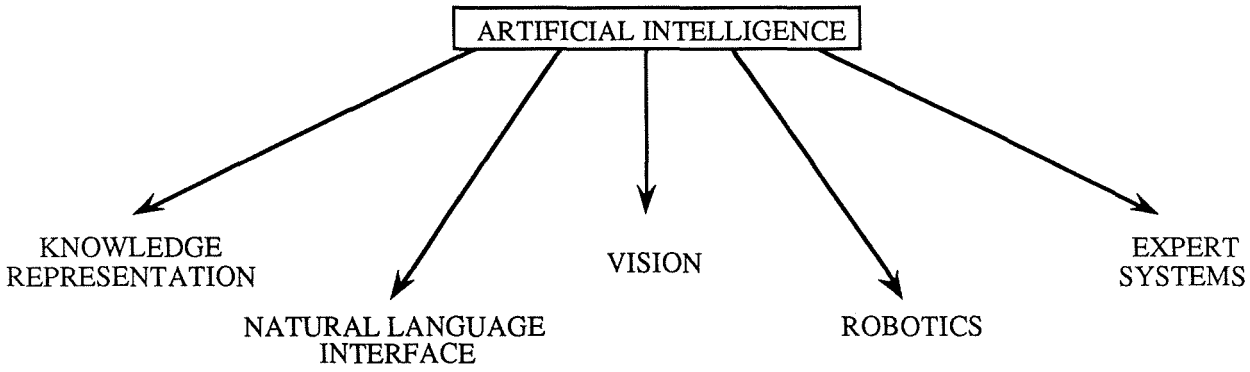

Figure 1.1 : artificial intelligence disciplines

1.1.2 Knowledge representation

Knowledge representation is perhaps the most important area of artificial intelligence research. It is the cornerstone on which all the other disciplines are built. Its long range quest is to find a general theory or method for representing any knowledge (such a theory would make the capture of commonsense knowledge possible and apply it to the solution of new problems).

1.1.3 Problem solving

Problem solving is finding a way to get from initial situation to a desired goal. Humans do it very effectively using deductive logic, procedural analysis, analogy, and induction. Computers, at least at the present time, generally solve problems only by deductive logic and procedural analysis. Efforts in that area are developed to solve problems by analogy and induction.

1.1.4 Natural language interfaces

The two primary goals of natural language research are to understand how humans communicate and to create machines with human-like communication skills.

1.1.5 Vision and robotics

Robotics is the branch of artificial intelligence research that is concerned with enabling computers to see and manipulate objects in their environment. Robotics research is primarily directed in three areas : the development of visual sensors, the development of manipulators and the development of heuristics for object- and space-oriented environmental problem solving.

1.1.6 Expert systems [GEV-84]

Experts systems probably constitute the "hottest" topic in artificial intelligence today. Prior to the last decade, in trying to find solutions to problems, artificial intelligence researchers tended to rely on non-knowledge-guided search techniques or computational logic. These techniques were successfully used to solve elementary problems or very well structured problems such as games. However, realistic problems exhibit the characteristics that their search space expands exponentially with the number of parameters involved. For such problems, these older techniques have generally proved to be inadequate and a new approach was needed. This new approach emphasized knowledge rather than search and has led to the field of knowledge engineering and expert systems. The resultant expert systems technology, limited to academic laboratories in the 1970s, is now becoming cost effective and is beginning to enter into commercial applications.

1.2 DEFINITION OF AN EXPERT SYSTEM

Edward Feigenbaum, a leading researcher in knowledge system at Stanford University, has defined an expert system as [GEV-84] :

> " ... an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant expertise for their solution. The knowledge necessary to perform at such a level,

> plus the inference procedures used, can be thought of as a model of the expertise of the best practitioners of the field.
> The knowledge of an expert system consists of facts and heuristics. The *facts* constitute a body of information that is widely shared, publicly available, and generally agreed upon by experts in a field. The *heuristics* are mostly private, little-discussed rules of good judgement (rules of plausible reasoning, rules of good guessing) that characterize expert-level decision making in the field. The performance level of an expert system is primarily a function of the size and quality of the knowledge base that it possesses. "

An expert system can be distinguish from other kinds of artificial intelligence program in that [JAC-86] :

- it deals with subject matter of realistic complexity that normally requires a large amount of human expertise;
- it must exhibit high performance in terms of speed and reliability in order to be a useful tool;
- it must be capable of explaining and justifying solutions and recommendations in order to convince the user that its reasoning is, in fact, correct.

But right from the outset, we should give an expectation warning : do not expect black magic ! Do not be surprised if some ideas are not totally new (many people can discover an idea, but few can recognise the significance of a key concept). And do not expect expert systems to make a complex problem simple. If a problem is complex, then it is a complex problem, and nothing can be done about it. The expert systems approach can only provide a mean of coping with complexity, not of eliminating it.

1.3 A LITTLE HISTORY [GOO-85]

In the very early days of artificial intelligence, there was a widespread belief that it would be possible to produce machines which were, to some degree, models of the human brain. Approaches to intelligence such as the cybernetic approach or the neural net approach were developed in the early 1950s. But the inventors of these approaches show an assumption often repeated in artificial intelligence - the assumption that if a small system works well with small problems, then scaling it up by a thousand times would enable it to work one thousand times as quickly, or solve problems a thousand times as big.

In the late 1950s, interest in symbolic digital computing eclipsed the cybernetic models. At the Dartmouth College

Conference in 1956, Newell and Simon gave a paper on "The Logic Theory Machine", and described how they had introduced the concept of list to help program the proof of problems. Within a few years, list processing had a language of its own - Lisp. An influential example of an early Lisp program was the GPS (General Problem Solver) which Newell and Simon developed from their Logic Theory Machine. But it turned out that programs like GPS were too general, and could not be applied to big problems. The behaviour of the best general problem solvers, the human problem solvers, was observed to be weak and shallow, except in the areas in which the human problem solver is a specialist. This announced the switch from general purpose reasoning to task-specific knowledge - the expert systems.

In 1965, Edward Feigenbaum had the insight which started work on DENDRAL within the Stanford University. DENDRAL, the first known expert system, was a departure from orthodoxy : a special-purpose program, conceived to solve problems only in the small topic of elucidating chemical structures. Concurrently to the realization of DENDRAL, other expert system prototypes were developed in the late 1960s in some research institutes. In 1969, CADAUCUS was implemented at the University of Pittsburgh to aid physicians in the diagnosis of human internal diseases. In 1970, MACSYMA (which represents approximately 100 work-years of development time) was employed at the MIT (Massachusset Institute of Technology) in sophisticated symbolic mathematical analysis. Then, in 1972, came MYCIN, one of the most publicized and famous expert systems, which assists the diagnosis and treatment of infections blood diseases. During the 1970s, MYCIN stimulated the development of other famous expert systems such as PUFF, AM, PROSPECTOR and XCON - the first expert system to be used successfully on a daily basis in a commercial environment [WOL-87].

Since the beginning of the 1980s, expert system development has become a key topic in artificial intelligence research, leading to the implementation of hundreds of new applications.

CHAPTER 2 : THE PROMISE OF EXPERT SYSTEMS

2.1 CONVENTIONAL PROGRAMS AND EXPERT SYSTEMS

To gain a better understanding about what expert systems really are, it is useful to compare them with the broad classes of conventional programs. Characteristics unique to expert systems are listed in Figure 1.2. [WOL-87]

	EXPERT SYSTEMS	CONVENTIONAL PROGRAMS
1.	quantitative and qualitative	quantitative
2.	symbolic processing	numeric processing
3.	heuristic computing	algorithmic computing
4.	data pattern driven	control driven
5.	uncertain and incomplete data	exact and factual data
6.	multiple solution path	single solution path
7.	explanation and justification	no explanation
8.	search intensive	computation intensive
9.	dynamic and static variables	static variables
10.	self-modification	no self-modification
11.	knowledgebase distinct from reasoning mechanisms	data and reasoning are mixed

Figure 1.2 : comparison of characteristics between expert systems and conventional programs

1. Expert systems are qualitative rather than quantitative in their results. Expert systems do not simply generate answers; they analyze and present the "best" possible answer with advice and recommendations. For example, an expert system can not only identify a mechanical failure in a piece of machinery, it can also identify potential sources of the

failure with their associated certainty factors, which indicate the strength of each recommendation.

2. Expert systems base their reasoning process on symbolic manipulation - the type of reasoning employed by human experts. Conventional programs recognize only numeric or alphabetic manipulations.

3. Conventional programs are based on algorithmic computing - they solve problems step-by-step. Expert systems computing process employs heuristics inference procedures. In other words, an expert system takes the place of a human expert in a problem-solving environment by interpreting knowledge, using heuristics, making inferences, and reasoning in ways similar to those of the human expert.

4. Expert systems are data pattern directed, versus control directed. Conventional programs execute according to a predefined data flow, no matter what type of data is inputted. Expert systems follow a flow determined by the relationship of the inputted data and the current contextual environment. There is no predefined execution path.

5. Expert systems can incorporate sets of values called certainty factors, confidence factors and probabilities which represent the definitude of a statement. These values represent the most common ways for the expert system to deal with uncertainty.

6. Problems addressed to expert systems can have multiple solution paths, one solution path, or no solution path. Expert systems have some knowledge of the basic principles in the problem domain; they know how they operate and how they arrive at the results. Conventional programs simply execute according to the predefined direction of the program. They have only one solution path.

7. Expert systems are capable of reconstructing inference paths for explanation and justification purposes. This provides what is called an "audit trail" for the end user. The end user can not ask a conventional program to explain how it derived its answer.

8. Expert systems are search intensive rather than computation intensive. Expert systems compare and analyse relationships among numeric and nonnumeric data, versus straightforward algebric computation of numeric data.

9. Expert systems are more flexibly designed than conventional programs. This allows modifications as problems become better defined and additional data becomes known. Since the environment of an expert system is never static, an expert system must be capable of manipulating dynamic variable inputs. Dynamic variables are those that are constantly changing, such as the stock market prices. An expert system employing stock market prices must be flexible enough to easily incorporate the continuous changes.

10. Some expert systems are self-modifying. If during, or after, a program run, the expert system determines that a piece of its data or knowledge base is incorrect, or no longer applicable because the environment has changed, it has the capacity to update the information. Consequently, if the same set of parameters were reentered, the expert system might deduce a different answer.

11. In expert systems, the knowledge base and the reasoning mechanism are distinct entities. In fact, it is often possible to use the reasoning mechanism with other knowledge bases to create a new expert system. In conventional programs, internal data and the reasoning mechanism are mixed entities.

2.2 THE ADVANTAGES OF EXPERT SYSTEMS

To help us understand what makes expert systems valuable, we shall discuss some of the advantages that expert systems have upon human experts and conventional programs. [GOO-85]

2.2.1 Increase of profitability

Expert systems can increase profitability because they :

- allow more turnover : XCON[1], an expert system which configures Vax computers, is a good example of how organisations can benefit from expert systems. XCON has enabled DEC (Digital Equipment Company) to increase fourfold the throughput of Vax orders. It has reduced the error rate on orders from 35% to about 2% .

- save money by saving time : DRILLING ADVISOR is an expert system which the company Tesknowledge created for Elf-Aquitaine, a French oil company. This system acts as a consultant on diagnosis and cure of drilling problems. When a well is drilled in the search for gas or oil, geologists collect large quantities of data called well-logs. These logs must be analysed to discover whether oil or gas are present. These analyses take a few days and a bad diagnosis is expensive; one day of lost time on an oil-rig can cost Elf $250.000. DRILLING ADVISOR reduces considerably the time spend on analyses.

- save money on equipment : where a human relies on the more detailed data from a more expensive instrument, an expert system can investigate more thoroughly the information from an instrument of lower quality.

- do a job with fewer staff : a company may be faced with the problem of too few experts on a given problem. There may be only one person in the organisation who has a skill that is needed widely : an expert system can be used to distribute knowledge throughout the organisation. Training a replacement

[1] XCON is described in section 2.

could take a long time, because skill at such tasks is usually gained through long experience, not exclusively by the study of formal rules and theoretical principles.

2.2.2 Better performances

Experts systems can perform better than a human because they:

- do not become tired or bored : this reminds us that a computer and its software can run 24 hours a day, 7 days a week.

- can handle large volumes of data : SUS is a naval application that monitors the many different kinds of information available to the commander of a naval vessel. SUS is used in the many situations where the flood of information to a ship's control centre is too great for a human.

- can respond more rapidly : foreign exchange and commodity markets are an area where immediate response to information is vital. An expert system which could spot a trading opportunity faster than a human would quickly recover its costs.

- can work in hostile environment : this is an obvious advantage of computer programs in general. We need only suggest the benefits of applications such as reactor control, or deep-sea drilling rigs.

2.2.3 Better knowledge manipulation

Expert systems are tools for manipulating knowledge, they can:

- discover new knowledge : AM is an expert system which tries to discover new mathematical concepts form old ones. After running for an hour in CPU time, AM had discovered about 300 new concepts; about 25 of these seemed interesting to professional mathematicians.

- help manage complex documentation : the ESP ADVISOR expert system was developed to replace complex manuals, procedures, rules or regulations.

2.2.4 Act better than conventional programs

Expert systems are better than conventional programs because they :

- can do tasks hard to program conventionally : XCON was such a case. DEC had tried and failed to write conventional programs to configure computers. ICL faced similar difficulties with DRAGON, their system for configuring 2900 series computers. The first full version of DRAGON took about 6 man-months to develop. It is estimated that with conventional methods, it would have taken 4 to 5 man-years to write the same system (that would have been much harder to keep up-to-date).

- can deal with uncertain and incomplete data : the knowledge base in an expert system does not need to be complete before the system can be useful. Similarly, the user can usually give a "don't know" or "not sure" answer to one of the system's questions during a session, and the expert system will still be able to produce an answer, although it may not be a certain one.

- can provide the users with explanation of why the computer is asking a particular question and how the computer obtained a particular result.

- can understand the goals of the user : a computer system that is based on expert system technology in its user interface includes knowledge about the user in the knowledge base. This knowledge could assist the computer system in understanding the user. With this understanding, the computer could provide the user with assistance in using the computer.

2.3 WHY KEEP A HUMAN IN THE LOOP ? [WAT-86]

If artificial expertise is so much better than human expertise, why not eliminate human experts, replacing them with expert systems ? The most highly skilled expert can perhaps be eliminated, but in many situations a moderately skilled expert should be kept in the loop. The expert system can then be used to augment and enhance this user's skills.

There are some very good reasons for not entirely eliminating the human from the loop. Although expert systems tend to perform well, there are important areas in which human expertise is clearly superior to the artificial kind.

One such area is creativity. People are much more creative and innovative than even the smartest programs. A human expert can reorganize information and use it to synthesize new knowledges. Human experts handle unexpected events by using imaginative and novel approaches to problem solving, including drawing analogies to situations in completely different problem domains. Programs have had little success in doing this.

Another area where human expertise excels is learning. Human experts adapt to changing conditions, they adjust their strategies to conform to new situations. Expert systems are not particularly good at learning new concepts or rules. Progress has been made in developing programs that learn, but these programs tend to work in extremely simple domains and don't do well when confronted with the complexity and detail of real-world problems.

Human experts can make direct use of complex sensory input, whether it be visual, auditory, tactile or olfactory. But expert systems manipulate symbols that represent ideas and concepts, so sensory data must be transformed into symbols that

can be understood by the system. Quite a bit of information may be lost in the translation.

Finally, human experts and nonexperts alike have what we might call *commonsense knowledge*. This is a very broad spectrum of general knowledges about the world and how it works, knowledges that virtually everyone has and uses. Because of the enormous quantity of commonsense knowledge, there is no easy way to build it into an intelligent program, particularly a specialist one like an expert system. Commonsense knowledge includes knowing what you know as well as what you don't know. For example, if you were asked to recall the phone number of your previous residence, you would search your memory trying to retrieve the information. If you were asked to give the phone number of Shakespeare, you would know at once that no answer exists, since telephones weren't around in Shakespeare's time. When an expert system is given questions it can't answer or for which no answer exists, it doesn't have the commonsense to give up. Instead, it may waste much time searching through its data and rules for the solution. Even worse, when the solution isn't found, it may think it is because its knowledge is incomplete and ask for additional informations to complete the knowledge base.

For these reasons and others relating to the public acceptance of artificial expertise, expert systems are often used in an advisory capacity - as a consultant or aid to either an expert or novice user in some problem area.

2.4 LIMITATIONS IN THE USE OF EXPERT SYSTEMS

So far, we have seen that expert systems have many more advantages than traditional programming techniques. Nevertheless, it does not mean that expert systems have no failings. Expert systems, like any other techniques, have their own limits.

Companies encounter several types of difficulties when trying to apply expert system technology to their problems [WAT-86] :

2.4.1 Lack of resources

Personnel competent to design and develop the systems are scarce, and few of the high-level support tools and languages are fully developed or reliable.

There are two reasons for this lack of resources. First, expert systems, like its parent field artificial intelligence, is still quite new and unfamiliar to many computer specialists, and therefore somewhat difficult for them to understand and apply. Second, the crush of companies entering the artificial intelligence arena has created many more openings for experienced artificial intelligence and expert system people than can be filled with existing personnel.

2.4.2 Inherent limitations of expert systems

Current expert systems and expert system tools have limitations, many of which will gradually disappear as artificial intelligence researchers advance the state of the art. But for development efforts in the near future, they are a fact of life.

Expert systems are not very good at representing temporal or spatial knowledge. Representations of this type can require huge amounts of memory to keep track of the state of things at various points in time or to record the spatial relations between different groups of objects.

We have already discussed the problems that expert systems lack using commonsense or general knowledge about the world.

Expert systems have a very narrow domain of expertise and hence, their operation is not as robust as the users might want. Because of this, expert systems have difficulties recognizing the limits of their ability. When pushed beyond their limits or given problems different from those for which they were designed, expert systems can fail in surprising ways (to put it another way, expert systems exhibit rather fragile behaviour at their boundaries).

The most serious limitations of expert system building tools is their inability to perform knowledge acquisition. Knowledge acquisition is the major bottle-neck in expert system development; it is tedious and time-consuming to extract knowledge from an expert and incorporate it into a large knowledge base. At present, a knowledge engineer extracts knowledge from human experts and laboriously builds it into the knowledge base. Despite research aimed at designing tools for automatically acquiring the knowledge, the bottle-neck still exists and results in project development times that seem unnecessarily long.

Languages for building expert systems are not as flexible and general as the knowledge engineer might want. Particular types of knowledge (e.g. temporal or spatial) often can not be represented easily, or different representational schemes can not be represented naturally and efficiently in the same language. Also, many languages do not provide mechanisms for building adequate user interface. The explanations are still primitive and a human expert may need to explain again what the expert system has explained.

2.4.3 Expert systems take a long time to build

Expert systems can not be built quickly. With the currently available technology, it takes from 5 to 10 man-years to build an expert system that solves a moderately complex problem. XCON, the system that configures computers, took about 8 man-years to reach reasonable performance. Earlier systems such as DENDRAL took over 30 man-years of effort to be built.

CHAPTER 3 : OVERVIEW OF EXPERT SYSTEM APPLICATIONS

We can broaden our understanding of expert systems by reviewing some of their most characteristic uses. We will describe these uses from two perspectives : the basic activities of expert systems and the areas in which they solve problems.

3.1 PROBLEM-SOLVING ACTIVITIES

Expert systems have been built to model the problem-solving strategies of human experts. Because different human experts use different problem-solving techniques, the expert systems modelled after the strategies of the human use a variety of problem-solving approaches. Depending on an expert's intentions in problem-solving strategies, the expert will perform one or more of the activities listed in Figure 1.3 [MAR-88]. The list may not be all-inclusive, but it does feature activities that experts have been known to use and can be supported by expert systems.

ACTIVITY	PROBLEM ADDRESSED
designing	configuring objects under constraints
planning	designing actions
forecasting	inferring likely consequences of given situations
controlling	governing overall system behavior
monitoring	comparing observations to expected outcomes
diagnosing	inferring system malfunctions from observables
prescribing	finding remedies for malfunctions
interpreting	inferring situation descriptions from sensor data
training	diagnosing, debugging and repairing student behavior

Figure 1.3 : problem-solving activities

Designing is the task of developing configurations of objects based on a set of problem constraints. Examples are designing integrated circuit layouts or creating complex organic molecules. Because design is so closely coupled with planning, many design systems provide mechanisms for developing and refining plans to achieve the desired design.

Planning is the activity of establishing goals, plans, policies, and procedures to achieve some end. Planning involves organizing actions as well as determining the modes of actions. Examples are creating an air strike plan or organizing the construction of a new high-rise office building.

Forecasting involves the examination of information of a given situation and predicting future situations. The forecasting activity could also be looked at as inferring future consequences of current decisions. Forecasting is often the act of presenting predictions after study and analysis of all available pertinent information. Examples are predicting the damage to crops from some type of insect or estimating global oil demand from the current geo-political world situation.

Controlling activities include direction, regulation and coordination of systems or machines. The directing activities include planning activities such as establishing goals and procedures, forecasting such as observing the present environment and correlating this with past data. Putting things into the same or required order is an example of coordinating activity.

Monitoring involves observing and checking for some specific purpose. To monitor a system means to keep track of the system, to observe the behaviour of the system and compare the observations with planned or designed behaviours. Monitoring systems usually have set limits within which the behaviour of the system being monitored can vary without any action on the part of the monitoring system. However, if observed system behaviours exceed these limits, the monitoring system is designed to notify some system or operator. Examples are monitoring instrument readings in a nuclear reactor to detect accident conditions or assisting patients in an intensive care unit.

Diagnosing activities include investigating and analysing the cause or nature of a problem. Diagnosing often involves the act of identifying a cause from observable signs or symptoms. Expert diagnostic systems infer what the problem is from observable signs and symptoms that are presented to them. Examples are locating faults in electrical circuits or finding defective components in the coolant systems of nuclear reactor.

Prescribing is the activity of recommending or designating the use of some agent as a remedy for a problem. Prescribing activities are very often related to diagnosing activities. When a doctor diagnosis a patient as having some illness, he will often prescribe some medicine as a remedy to correct the problem. Doctors can describe patient symptoms to the MYCIN

expert system, and the system will diagnose bacterial infections and prescribe treatment for the infection.

Interpreting systems use sensor data to infer situation descriptions. Interpreting systems deal directly with real data rather than with clean symbolic representations of the problem situation. They face difficulties that many other types of systems avoid because they may have to handle data that are noisy, incomplete, unreliable or erroneous.

The goal of *training* is to increase the knowledge or skill level of a student in some subject domain. Training often starts with some investigation of what the student already knows about the subject of interest. Analysis of what the student already knows versus what the student should know determines what needs to be taught. The GUIDON expert system is an intelligent system that trains a student by asking questions to technical subjects, receiving answers to its questions, and then either telling the student that the response is correct or correcting the student's response.

3.2 SUPPORTED DOMAINS

Although the basic activities of expert systems are easy to describe, it is misleading to use them to categorize existing expert systems because many expert systems perform more than just one activity. For example, diagnosis often occurs with debugging, monitoring with control, and planning with design. Consequently, it is useful to categorize expert systems by the type of problems they solve. Figure 1.4 shows some of the problem domains in which expert systems are now working [WAT-86]. Of these areas, the medical domain seems the most popular one; more expert systems have been developed for medicine than for any other problem area.

Expert system work in *chemistry* started with DENDRAL, an innovative research project at Stanford University in the mid-1960s. Current expert system work includes inferring molecular structure, synthesizing organic molecules, and planning experiments in molecular biology.

Expert system work in *computer systems* is typified by XCON, one of the first and most successful systems of this type. Current work in computer systems includes fault diagnosis, computer configuration and manufacturing control.

Applications in *electronics* is dominated by research and development efforts involving equipment fault diagnosis and integrated circuit design. ACE, developed by Bell Laboratories in the early 1980s, typifies fault diagnosis systems in this area. It is being used by AT&T to locate and identify trouble spots in telephone networks. Current expert system work in this area also includes the development of instructional systems for electrical troubleshooting and digital circuit design.

Agriculture	Manufacturing
Chemistry	Mathematics
Computer systems	Medicine
Electronics	Meteorology
Engineering	Military science
Geology	Physics
Information management	Process control
Law	Space technology

Figure 1.4 : supported domains

Expert system in *engineering* is typified by DELTA, a fault diagnosis system developed by General Electric in the mid-1980s. General Electric uses DELTA on a commercial basic to help maintenance personnel to find malfunctions in electric locomotives. Current work in that domain includes other fault diagnosis efforts and instruction in the operation of complex process control systems.

Work in *medicine* began with MYCIN, one of the earliest and best known expert systems. Developed at Stanford University in the mid-1970s, MYCIN helps a physician diagnose and treat infection blood diseases and is now being used for research and medical teaching. Current expert system work in medicine includes interpretation of medical test data, disease diagnosis, disease treatment, and instruction in medical diagnosis and management techniques.

Expert system work in the *military domain* has focused on interpretation, prediction and planning. One of the first military expert systems was HASP/SIAP. It identifies ship types by interpreting data from systems that monitor regions of the ocean. Current work in the military domain includes interpretation of sensor data, prediction of combat results and tactical planning.

CHAPTER 4 : ARCHITECTURE OF EXPERT SYSTEMS

As shown in Figure 1.5, the four basic components of an expert system are the knowledge base, the inference engine, the interface module and the development engine [WOL-87].

4.1 THE KNOWLEDGE BASE

The *knowledge base* of an expert system contains the information used to solve a problem. A knowledge base is a step above conventional databases in that a knowledge base not only contains static data as in a database, but also relational informations. A third area of the knowledge base is the *working memory*. The working memory is used only during processing and is the resident space for information manipulation. In that area, the knowledge base is modified by the inference engine as situations and data change (a much more interactive area of the expert system than the database). The working memory takes the knowledge from the knowledge base and combines it with the information supplied from the user to be massaged then by the inference engine in pursuit of a solution.

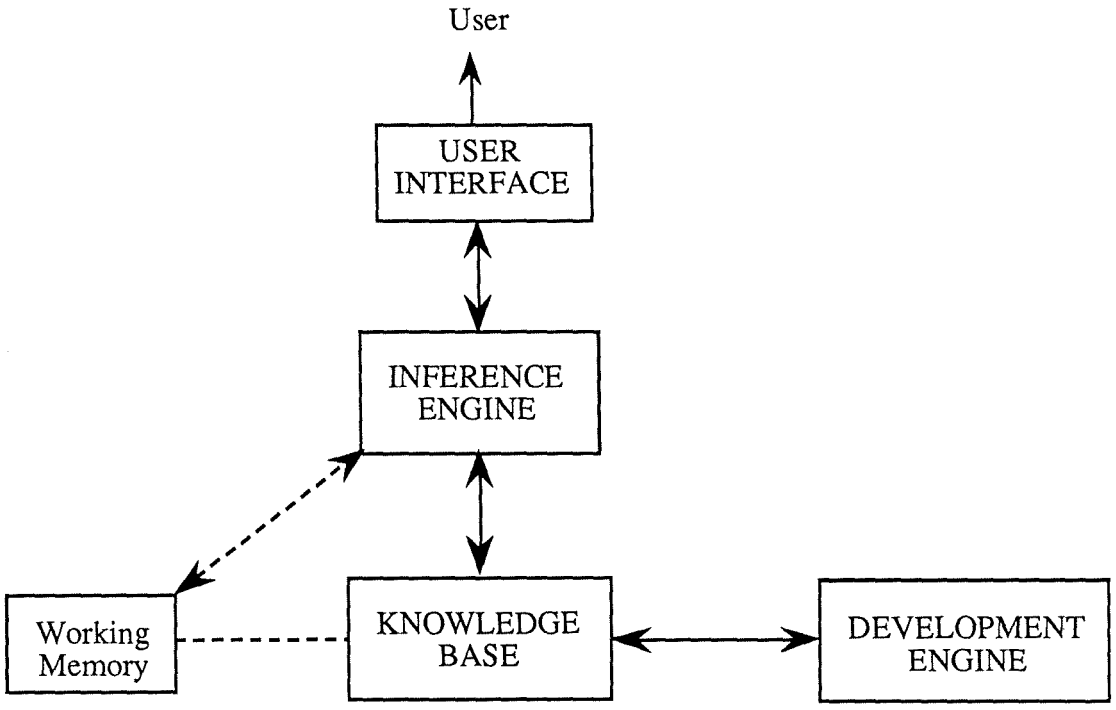

Figure 1.5 : architecture of expert systems

4.2 THE INFERENCE ENGINE

The inference engine is the workhorse of the expert system. It consists of the processes that work the knowledge base, do analyses, form hypotheses, and audit the processes according to some strategy that emulates the expert's reasoning. The inference engine massages new information, combines it with the knowledge base, considers the relationships in the knowledge base and proceeds to solve the problems in working memory using its established reasoning and search strategies. In other words, the inference engine is "the thinker" of the system.

4.3 THE INTERFACE MODULE

Numerous interfaces are used in the creation and operation of an expert system. Interfaces include a terminal, graphical representations, multiple character windows and multiple graphic windows. These interfaces operate in three situations. The first situation is where the user acts as a client. Here the user wants answers to problems. The second situation is where the user acts as a tutor to enhance the expert system. Here the user, primarily the knowledge engineer, wants to improve or increase the knowledge of the expert system. The third situation is where the user acts as a pupil of the expert system. In this situation the user wants to harvest the knowledge base.

4.4 THE DEVELOPMENT ENGINE

A development engine, also called knowledge acquisition subsystem, is vital in the creation of the expert system in that it allows the knowledge engineer to create, modify, add and delete information from the knowledge base. The development engine is not always resident within the expert system software[2].

[2] The knowledge acquisition process will be dicussed in more details in chapter 6.

CHAPTER 5 : INSIDE EXPERT SYSTEMS

5.1 KNOWLEDGE REPRESENTATION

The manner in which knowledge is represented crucially affects the ease and speed with which a program can use it. Consider the example of Roman versus Arabic numerals. Both of them can represent any positive whole number. There are unambiguous rules for operations like addition and division. These rules can be programmed into a computer, or embodied in hardware. Despite this, no one would ever use a Roman numerical like system for computer arithmetic. Therefore, knowledge representation is critical subject. For this reason, the choice of suitable formalisms and techniques for representing and reasoning with knowledge is a matter of continuing debate and active research in artificial intelligence circles.

In this chapter, we will briefly describe the four most common ways to represent knowledge : rules, semantic nets, frames and predicate calculus.

5.1.1 Rules

The most popular format for representing knowledge in a way that maintains its procedural character is the *production rule* which is simply a statement program of the form [ALT-84]:

" IF premise(s) THEN action(s) "

The IF clause describes an object, situation or position. If the IF clause is true, the THEN clause of the production rule is activated . The IF clause is called the premise; the THEN clause is called the action. A rule is said to be activated when all the conditions in the premise of the rule are satisfied by the current situation. The production is said to be fired or executed when the action is performed.

An example of a rule from a monitoring system is given below. The system, called REACTOR, monitors instrument readings in a nuclear reactor, looking for signs of accident [WAT-86].

IF		The heat transfer from the primary coolant system to the secondary coolant system is inadequate
	and	The feedwater flow is low
THEN		The accident is low of feedwater

Figure 1.6 : example of rule

The simple idea behind a set of production rules is that they define a set of allowed transformations which moves a problem from its initial statement to its solution. The current state of a solution is represented by the set of facts asserted in a database. The solution progresses by the matching of one side of a rule resulting in changes to the database.

One of the original attractions of production rule formalisms is their simplicity. Production rules are structured similarly to the way people think about solving a problem. Given a situation, production rules best express "what to do next" statements. Despite this ideal, it has been pointed out that in order to design usable systems, significant deviations are necessary. These may involve introducing some structure into the database, having various levels of rule, and allowing rules to access standard programming procedures [ALT-84].

5.1.2 Semantic nets

Semantic network notation is based on the ancient and very simple idea that memory is composed of associations between concepts. The basic functional unit of a semantic network is a structure consisting of two points, or *nodes*, linked by an arc. Each node represents some concept and the arc represents a relation between pairs of concepts. Such pairs of related concepts may be thought of as representing a simple fact. The judicious choice of relational labels permits the expression of very complex groups of facts. Figure 1.7 illustrates one possible representation of facts about an employee "smith" [ALT-84].

Flexibility is the major advantage of semantic nets through the ability to add, modify or delete new nodes and arcs where appropriate. Another benefit is the ability to inherit relationships from other nodes - more specifically, the ability to reason and make assertions about one node and its relationship with another node where no direct arc exists between the two nodes [WOL-87]. This inheritance is illustrated in Figure 1.7 by the statement "Mr Smith works-in building-1" .

The best known disadvantage of semantic nets is their inability to represent complex situation. If the problem under review has many exceptions, the number of nodes and arcs needed to describe the various exceptions becomes very large, making the structure of the semantic net inelegant and complex to handle.

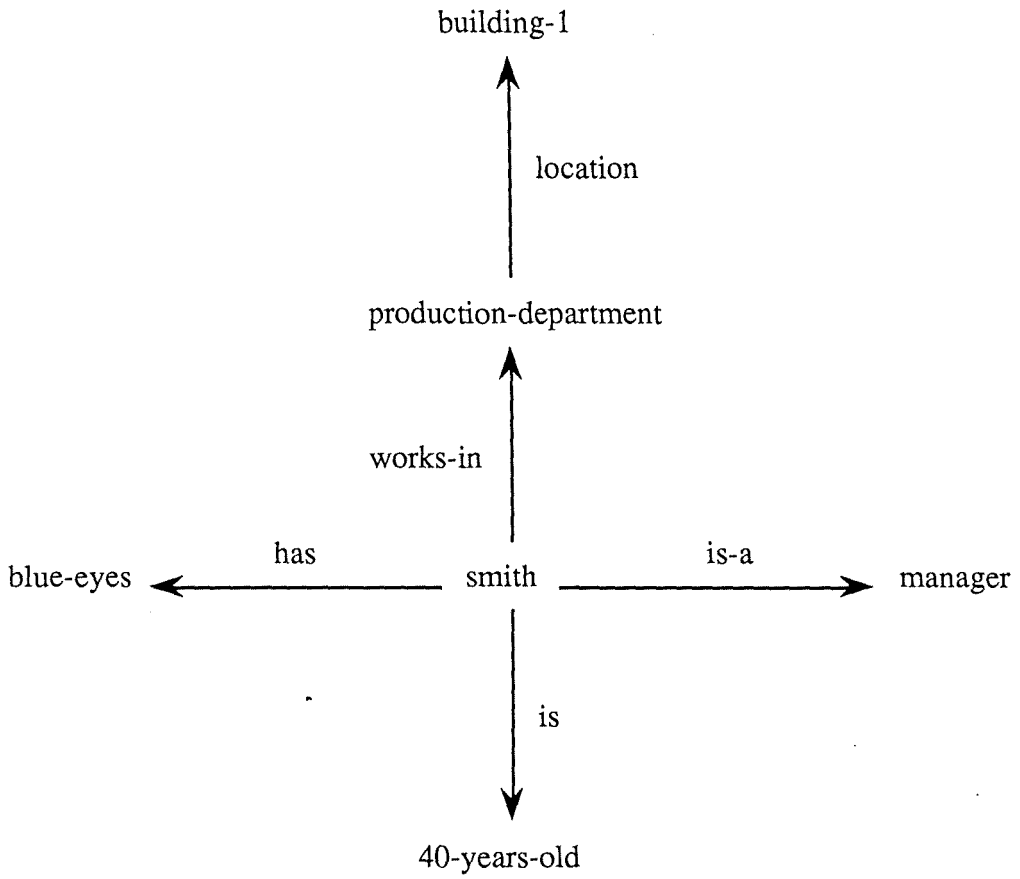

Figure 1.7 : example of semantic net

5.1.3 Frames

A *frame* is organized much like a semantic net [WAT-86]. A frame is a network of nodes and relations organized in a hierarchy, where the topmost nodes represent general concepts and the lower nodes represent more specific instances of those concepts. So far, this looks just like a semantic net. But in a frame, the concept at each node is defined by a collection of attributes, called *slots*, and values of those attributes. A frame is a partition of knowledge; a slot describes its individual proprieties. The slot representation is a declarative statement for asserting attributes of a frame. The slot can have default values or subslots which represent a further breakdown of attributes. Slots also allow for procedural attachments. Procedures can be attached to slots to drive the reasoning or problem-solving inference engine for that slot. The procedure is a set of instructions that, when executed, produces consistent results with the facts for a particular slot.

Figure 1.8 is an example of a frame instantiated to describe an employee "smith" named "sm-1" [ALT-84].

name : sm-1

specialisation-of : EMPLOYEE

name : smith, john, henri

age : 40

address : adr-1

department : production

salary : sal-1

start-date : april-1972

to : now

Figure 1.8 : example of frame

5.1.4 Predicate calculus [WOL-87]

Predicate calculus, or logic programming, relies on the truth and rules of inferences to represent symbols and their relationships to each other. For example, If A Then B and A exists, allow us to conclude B. This provides a simple way of determining the truth or falsity of a statement. Predicate calculus is actually an extension of propositional logic. The form of logic most often used is first-order predicate logic, which is an extension of predicate calculus.

First-order predicate logic works with variables, predicates, sentential connectives, qualifiers and functions. Variables are place holders to represent objects, things and statement in question. Predicates describe a relationships or make a statement about the variable under consideration. Predicates can be thought of as verbs and can be applied to any numbers of variables. Sentential connectives are used to make complex sentences. The five most commonly used sentential connectives are

and	^ or &
or	v
not	¬
implies	->
equivalent	=

The notion of a function is similar to that of predicate, except that a function returns objects that are related to their arguments (for example, the function father-of(X,Y) will return a name, not a true/false value).

Figure 1.9 illustrates the fact that a person X works in the production department if that person works in building-1 and if that person is a manager or if that person is at least 40 years old.

works-in(X,production) = works-in(X,building-1) &
is-a(X,manager) v
is-at-least(X,40-years-old)

Figure 1.9 : example of predicate calculus

5.2 CONTROL STRATEGIES

One important issue in the design of the control portion of the expert system is the decision of the search procedure, the order in which the rules are scanned for triggering. This involves a decision on the direction of the search and the search method. Control procedures are normally a part of the inference engine, and the knowledge engineer has only limited control over the procedure in most of the systems.

There are two search decisions that must be made in designing the control strategy [TOW-86] :

1. What is the starting point ? The starting point determines whether the search uses forward chaining or backward chaining.

2. How can the search be more efficient ? This involves developing heuristics[3] to resolve conflicts when multiple paths exist and for eliminating paths that are not useful.

5.2.1 Backward and forward chaining

In a *backward chaining* system, a conclusion is assumed and the inference engine works backward in an attempt to find the facts supporting that conclusion. As soon as it finds a valid conclusion, it moves to a subgoal for that conclusion and then tries to prove this. This type of search is often called a goal-driven search.

In a *forward chaining* system, the inference engine starts with the facts, and then works forward to find a conclusion that is supported by the facts. As each new conclusion is reached, it is added to the working memory. Forward search strategies are often called data-driven searches.

[3] heuristic : from a greek word "heuriskien", meaning to discover or invent by onself, to find.

5.2.2 Improving search efficiency

In a system with hundreds of rules, it is often advisable to implement strategies to minimize or improve efficiency of the search. A few of the search strategies used to improve efficiency are depth-first search, breath-first search and problem-reduction.

5.2.2.1 Depth-first versus breath-first search

In a *depth-first search*, details are pursued as deeply as possible until the search fails. In a *breath-first search*, all possible details at one level are scanned before moving to the next detail level. An expert generally appreciates the depth-first search because all details relative to a particular conclusion are presented together. A generalist prefers the breath-first search, as such an analysis is not restricted by previously conceived relationships of the details.

5.2.2.2 Problem reduction

In this strategy, subgoals are defined and then an attempt is made to prove these as intermediate conclusions to the final goal. By carefully defining the rules so that the problem solution is a hierarchy of goals and subgoals, the search path can be minimized in the problem solution space.

CHAPTER 6 : KNOWLEDGE ENGINEERING

Building an expert system is vastly different from coding a program. It is an entirely new area of computing. This chapter addresses a new area of expertise : *knowledge engineering*. Knowledge engineering is the process of synthesizing knowledge into a computer system so that problems are electronically solved through symbolic manipulation and reasoning of the knowledge base [WOL-87]. A knowledge engineer differs from a programmer in that the former must not only be technically adept with expert system software, but must also possess superior psychological, communicative, and management skills. A programmer, given an application with known inputs and outputs, must provide the means to transform the inputs into outputs. The primary task of a knowledge engineer is to identify critical inputs and outputs, discern the inner process by which the human expert transforms these inputs into outputs, establish this knowledge in the appropriate computer system, and finally, encourage the successful integration of the expert system into the organisation.

6.1 PRINCIPLES FOR DESIGNING EXPERT SYSTEMS

About 20 years have elapsed since the first known expert systems. One might thus expect the development of a clear set of general principles for designing expert systems. However, the design of expert systems is still an art and the principles which have emerged are far from general, representing little more than a summation of conventional wisdom. Therefore, the principles shown in Figure 1.10 are seen as a powerful starting point for expert system design, but they do not represent a universal theory for designing such systems.

Bounded domain [SIE-86]. To be able to build a practical expert system which offers advice in a given domain, you should be able to clearly define the limits of that domain. The domain should have a finite number of goals, solution approaches, relationships and factors. Furthermore, the number of goals and solutions approaches should be small and the number of relationships and factors should not be so large as to cause confusion.

Separate knowledge from inference engine [SIM-84]. In this way, the expert system is divided into an inference engine and a knowledge base, allowing the knowledge to be more easily identified, more explicit and more accessible. If the two are intermixed, domain knowledge will get spread out through the inference engine, and it becomes less clear what we ought to change to improve the system.

1. Define the limits of the domain
2. Separate the inference engine and the knowledge base
3. Use representation as uniform as possible
4. Keep the inference engine simple
5. Exploit redundancy
6. Build a prototype as soon as possible
7. Expert should be available throughout the project
8. Keep the expert's interest in the project

Figure 1.10 : principles for designing expert systems

Uniform representation [SIM-84]. This cuts down the number of mechanisms required, keeping system design simpler and more transparent. Each time a new representation is added to the system, something else in the system has to be able to handle it, has to know its syntax or semantics to be able to use it. Hence fewer representations mean a simpler, more transparent system.

Simple inference engine [SIM-84]. Keeping the inference engine simple helps in several ways. Since explanations are generated by replaying the actions of the system, keeping those actions simple means that little work is necessary to produce comprehensible explanations. Knowledge acquisition is also easier. When the inference engine is less complicated, less work is needed to determine exactly what knowledge has to be added to improve system performances.

Exploit redundancy [HAY-83]. Redundant data, hypothesis structures and inference rules can be useful to avoid problems caused by erroneous or missing information. No conclusion should rely too strongly on a single piece of evidence.

Build a prototype [WEI-84]. Because expert reasoning problems are frequently poorly specified, one needs to have something concrete to view. It is particularly important for the expert to see something running early. A running program is worth thousands of words from an unformalized interview with the expert. The initial prototype may be crude, will certainly be incomplete, and may contain inaccuracies but at least it provides a good starting point.

Expert availability [BON-85]. It is essential that genuine expert is available throughout the whole period of the

project, and that several other experts can be called upon to criticize and improve the prototype version. It is of prime importance to include all the unusual cases met by these experts. An experienced specialist will often go straight to the heart of a problem because he knows the right questions to ask.

Keeping the expert's interest [BON-85]. Expert systems suffered for a long time from predictions of enthusiastic practitioners who made extravagant claims about what the computer would be able to do. The scepticism this bring along should then be overcome by quickly developing a prototype and showing that it actually works, even if its achievements are not quite up to the original aim. The expert who was involved in the construction of the system will be encouraged to continue if he sees some of his own reasoning processes represented in the machine.

6.2 STAGES IN BUILDING EXPERT SYSTEMS

As said in part 6.1, expert system builders don't have a series of well-defined steps that they follow when constructing a system. The inherent complexity of the system building process precludes laying out all the steps in advance. As a result, system builders have pointed out that an evolutionary development is the most effective way to proceed.

The evolution of an expert system normally proceeds from simple to hard tasks by incrementally improving the organization and representation of the system's knowledge. This incremental approach to development means that the system itself can assist in the development effort. As soon as builders acquire enough knowledge to construct even a very simple system, they do so and use feedback from the running model to direct and focus their efforts. The incremental approach also means that the system builders can profit from what they learn in implementing the initial aspects of the system.

Expert system development can be viewed as five highly interdependent phases [WAT-86] : identification, conceptualization, formalization, implementation and testing. Figure 1.11 illustrates how these phases interact.

6.2.1 Identification stage

The first stage in building an expert system is to characterize the important aspects of the problem. This involves identifying the participants, problem characteristics, resources and goals.

6.2.1.1 Participant identification and roles

Before the development process can begin, the participants must be selected and their roles defined. The most common scenario involves interaction between the domain experts and the

knowledge engineers. Domain experts act as informants who tell the knowledge engineers about their knowledge or expertise. The knowledge engineers act as designers and builders of the expert system through the whole development process. But the construction of an expert system can also include other participants such as the future users of the system. These participants may help the knowledge engineers to design a better environment for the final product.

6.2.1.2 Problem identification

Once the participants are chosen, the knowledge engineers and domain experts can proceed toward identifying the problem under consideration. This involves an informal exchange of views on various aspects of the problem, its definition, characteristics, and subproblems. The objective is to characterize the problem and its supporting knowledge structures so that the development of the knowledge base may begin. Several iterations of the problem definition may be necessary since the knowledge engineers or domain experts may find that the initial problem considered is too large or unwieldy for the resources available. In short, the participants isolate and verbalize the knowledge that is relevant to the solving of the problem and identify the key elements of the problem description.

6.2.1.3 Resource identification

Resources are needed for acquiring the knowledge, implementing the system, and testing it. Typical resources are knowledge sources, time, computing facilities, and money.

The domain experts and knowledge engineers must use various sources to obtain knowledge relevant to building expert system. For the domain experts, these include past problem-solving experience, textbooks, and examples of problems and solutions. For the knowledge engineers, the sources include experience on analogous problems and knowledge about methods, representations, and tools for building expert systems.

Time is a critical resource. Both knowledge engineers and domain experts must be able to devote many months of intensive activity just to get the first prototype running. Obviously, computing and financial resources are critically scarce.

6.2.1.4 Goal identification

It is likely that the domain experts will identify the goals or objectives of building the expert system when identifying the problem. It is helpful, however, to separate the goals from specific tasks of the problem, since they constitute additional constraints that can be useful in characterizing the feasibility of certain approaches.

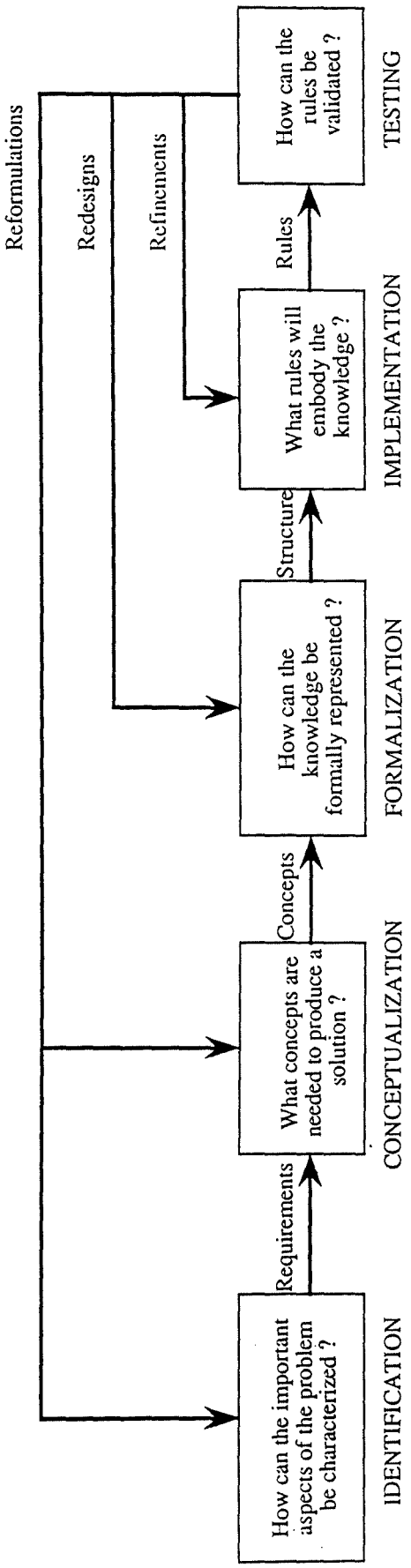

Figure 1.11 : stages in building expert systems

6.2.2 Conceptualization stage

The key concepts and relations, already mentioned during the identification stage, are made explicit during the conceptualization stage. This stage, like the previous one, involves repeated interactions between the knowledge engineers and the domain experts that are important, difficult and time-consuming. It will be tempting to try to analyze the problem correctly and completely before implementing a trial system. During this stage, subtasks, strategies and constraints related to the problem-solving activity are also explored. At this time, the issue of the detail level of the knowledge representation is usually addressed.

6.2.3 Formalization stage

The formalization process involves mapping the key concepts, subproblems, and information flow characteristics isolated during conceptualization into more formal representations based on various knowledge engineering tools or frameworks. The knowledge engineers now take a more active role, telling the domain experts about the existing tools, representations, and problem types that seem to match the problem at hand.

The result of formalizing the conceptual information flow and subproblem elements is a partial specification for building a prototype. This specification follows from the choice of organizing framework and the explicit sketch of the concepts and relations essential to the problem.

6.2.4 Implementation stage

Implementation involves mapping the formalized knowledge from the previous stage into the representational framework associated with the tool chosen for the problem. As the knowledge in this framework is made consistent and compatible, and is organized to define a particular control and information flow, it becomes an executable program. The knowledge engineers evolve a useful representation for the knowledge and use it to develop a prototype expert system.

The knowledge made explicit during the formalization stage specifies the contents of the data structures, the inference rules, and the control strategies. The tool or representation framework specifies their form. Local consistency of the problem-solving primitives will already have been worked out in previous stages but does not guarantee an executable program, since there may be global mismatches between data structures and some rules or control specifications. Such inconsistencies must be eliminated to ensure rapid development of the prototype expert system.

6.2.5 Testing stage [HAY-83]

The testing stage involves evaluating the prototype system and representational forms used to implement it. Once the

prototype system runs from start to finish on two or three examples, it should be tested with a variety of examples to determine weaknesses in the knowledge base and inference structure. The experienced knowledge engineer will elicit from the domain expert those problems likely to challenge the performances of the system and reveal serious weaknesses or errors. The elements usually found to cause poor performance because of faulty adjustment are input/output characteristics, inference rules, control strategies, and test examples.

The primary *input/output characteristics* are data acquisition and conclusion presentation. The method of acquiring data may be faulty or inadequate due to the fact that wrong questions are being asked or not enough information is being gathered. The conclusions output by the program may either be adequate or inadequate. There may be too few or too many conclusions, with not enough or too many intermediate hypotheses specified. The conclusions may not be appropriately organized or ordered, and the output may be at an inappropriate level of detail, either too verbose or too sparse.

The most obvious place to look for errors in reasoning is in the set of *inference rules*. Among other things, rules may be incorrect, inconsistent, incomplete, or entirely missing. If a rule's premises are incorrect, they may lead to an inappropriate context of application, thus invalidating its logic. Similarly, the conclusion of a rule may be incorrect, often in scope or in failure to separate subcases. And even if both premises and conclusions are correct, they may be linked via incorrect measures of association.

Errors in a prototype system often occur in the *control strategies* used. When the system considers items in an order that differs from the "natural order" of the experts, the knowledge engineers must look to the control structure for problems. Sequencing is more than cosmetic : there are often good reasons why data are considered in a particular way.

Finally, problems with a prototype system may arise from selecting poor *test examples*. Sometimes, failures can be traced to particularities of the test problem that were outside the intended scope of the system. More often, however, the set of test cases is too homogeneous and fails to test the program adequately. To ensure against such homogeneity, the test problem examples must be organized so that they cover the subproblems, probe the boundaries of expected "hard" cases, deal with the "classical" or prototypical cases of a problem, exhibit ambiguity, and provide for situations involving both hard and soft data.

6.2.6 Prototype revision

In the course of building an expert system, there is almost constant revision, which may involve reformulation of concepts, redesign of representations, or refinement of the implemented system. Refinement of the prototype normally involves recycling through the implementation and testing stages in order to tune or adjust the rules and their control

structures until the expected behaviour is obtained. The result of revision should be a convergence of performance. If this does not occur, the knowledge engineer must undertake more drastic modifications of the architecture or knowledge base. This is called redesign, a recycling back through the formalization stage with the new representation. If the difficulties are even more serious, they may involve mistakes of conceptualization or identification that will necessitate a reformulation of some of the concepts (objects, relations or processes) used in the program.

6.3 KNOWLEDGE ACQUISITION

6.3.1 The problems of the knowledge acquisition process [WAT-86]

One of the most critical responsibilities of the knowledge engineer is *knowledge acquisition*. Acquiring the knowledge needed to power an expert system and structuring that knowledge into a usable form is one of the primary bottle-necks in expert system development (this phenomenon is amplified by the fact that no automatic methods for doing this exist, with the exception of some very simple system-building aids).

Knowledge acquisition is defined as the process of identifying, extracting, documenting, and analyzing the information processing behaviour of domain experts in order to define the knowledge base and inference engine of an expert system. Knowledge in an expert system may originate from many sources, such as textbooks, reports, databases, case studies, empirical data, and personal experience. However, the dominant source of knowledge in today's expert systems is the domain expert. A knowledge engineer usually obtains this knowledge through direct interaction with the expert.

This interaction consists of a prolonged series of intense, systematic interviews, usually extending over a period of many months. During the interviews, the knowledge engineer presents the expert with realistic problems to solve that are the type of problems the expert system is being designed to handle.

The knowledge engineer must work with the expert in the context of solving particular problems. It is seldom effective to ask the experts directly about their rules or methods for solving a particular type of problem in the domain. Domain experts usually have great difficulty expressing such rules. One reason for this is that experts have a tendency to state their conclusions and the reasoning behind them in general terms that are too broad for effective machine analysis. Experts make complex judgments rapidly, without laboriously reexamining and restating each step in their reasoning process. The pieces of basic knowledge are assumed and are combined so quickly that it is difficult for them to describe the process.

Even worse, when experts attempt to explain how they reached a conclusion, they often construct plausible lines of reasoning that bear little resemblance to their actual problem-solving activity. This effect has at least two important implications for the building of expert systems : first, it suggests that domain experts need outside help to clarify and explicate their thinking and problem solving. Secondly, it suggests that the knowledge engineer should believe in a legitimate rule of expertise only if the expert has demonstrated the accuracy's use of the rule during problem solving.

Sometimes, the behaviour of experts appears more intuitive than intelligent. Considerable knowledge has been found to be an essential prerequisite to expert skill. Large numbers of patterns serve as an index to guide the expert in a fraction of a second to relevant parts of the knowledge store. This capacity to use patterns to guide a problem's interpretation and solution is probably a large part of what we call physical intuition (this is due to the fact that when experts solve problems in their area of expertise, they recognize new situations as instances of things with which they are already familiar). However, when experts are faced with new or novel situations, they behave more like intelligent novices. They tend to apply general principles and deductive steps that provide casual links between various stages of a problem-solving sequence.

These differences in problem-solving strategies suggest some techniques for decompiling an expert's knowledge. One possibility is to present novel situations (perhaps suggested by other experts) and note the process they perform to solve the problem. An alternative is to present an intelligent novice with a standard problem to gain insight into the actual problem-solving activity.

6.3.2 Some techniques to acquire knowledge [GOO-85]

The first essential in any knowledge engineering involving experts is close contact between the expert and the knowledge engineer. Once a working relationship has been established, the knowledge engineer needs a battery of techniques for eliciting the knowledge from the expert. The most common techniques are :

Examples problems. In this form of interview, the expert is presented with an example of the sort that the expert system will be expected to tackle.

Classification interviews. The purpose of this form of interview is to gain insight into the expert's mental model of the problem domain. In that analysis, the expert is presented with triple of objects from the domain and asked to say what features distinguish any two from the third.

Directed interviews. The knowledge engineer picks a set of representative problem and informally discusses them with the expert. The goal is to determine how the expert organizes knowledge about each problem, represents concepts and hypotheses, and handles inconsistent or imprecise knowledge.

Discussion of a prototype system. The expert examines and critiques each rule in the prototype system and evaluates the control strategies used to select the rules.

On-site observation. The knowledge engineer observes the expert solving real problems on the job rather than realistic problems in a laboratory setting.

6.4 PROTOTYPING

6.4.1 The importance of a rapid prototype

We have seen in part 6.2 that, as soon as builders acquire enough knowledge to construct a system even very simple, they do so and use feedback from running model to direct and focus their efforts. This very early system, also called *rapid prototype*, has a significant importance when building expert systems. Many reasons help us understand why a rapid prototype should be implemented as soon as possible.

First of all, a rapid prototype might positively influence the decision concerning the final acceptance of the project. A rapid prototype takes just a little time to develop, and, if this one already unveils general good performances, it could influence the verdict to fund the development of a commercial product.

Moreover, the rapid prototyping approach is based on the belief that it significantly reduces the risk of an incorrect identification of the problem, experts, or problem-solving methods. As discussed previously, most expert systems effectively deal with ill-defined, poorly structured problems. No conceptual model of the problem may have been formulated previously, and intended users may not be able to express their ideas and objectives correctly. The use of rapid prototyping can help the developer and the intended user to define the requirements and design specifications for the full-scale expert system.

Another reason for developing a rapid prototype is to make something work early in a project, rather than waiting until the end of lengthy paper analysis and design phases to begin coding. That working prototype will stimulate the intended user as well as the developers since it allows them to better visualize the enhancements of the project.

Finally, prototyping improves the communication between the knowledge engineer and the human expert. By getting experts involved in discussions of the functional areas of daily operations, the knowledge engineer helps the experts to better understand the evolution of the project, and to be more confident concerning the impacts of the expert system. A more confident expert and a better assisted knowledge engineer improves the atmosphere of work.

All these reasons make us believe that rapid prototyping is a key concept that has the potential to produce more complex systems, more quickly and successfully than by other means.

6.4.2 Development stages of a prototype [WAT-86]

Because of the incremental approach chosen for the development process, most expert systems evolve from the initial prototype to the final commercial product through certain stages. We identify five development stages.

Most expert systems begin as a *demonstration prototype* (or *rapid prototype*), that is, a small demonstration program that handles a portion of the problem that will eventually be addressed. This type of program is often used in two ways : first, to convince potential sources of funding that expert system technology can effectively be applied to the problem in question; and second, to test ideas about problem definition, scoping, and representation for the domain.

Most current expert systems have evolved to the stage of *research prototype*, a medium-sized program capable of displaying credible performance on a number of test cases. These systems tend to be fragile; they may fail completely when given problems that fall near the boundary separating problems they can handle from those they can not. Because they lack sufficient testing, they may also fail in some problems well within their scope.

Some expert systems have evolved past the research prototype to the stage of *field prototype*. These systems are medium- to large-sized programs that have been revised through testing on real problems in the user community. They are moderately reliable, contain smooth, friendly interfaces, and address the needs of the end-user.

A few expert systems have reached the stage of *production prototype*. These systems are large programs that have been extensively field-tested and are likely to have been reimplemented in a more efficient language to increase speed and to reduce computer storage requirements.

Only a few expert systems have reached the stage of *commercial system*. These systems are production prototypes used on a commercial basis.

6.5 EVALUATION OF EXPERT SYSTEMS

Evaluation of expert systems, unlike knowledge acquisition or inferencing mechanisms, is a topic that has been minimally addressed. However, evaluation, whether informal or formal, is an important element in expert system development because it enables a feedback process to take place, whereby the comments serve as a basis for iterative refinements. But expert system evaluation is often nebulous since there are no universally accepted or unbiased formal specifications against which the

system can be judged. Expert system evaluation is analogous to evaluating a financial portfolio. Several performance indexes exist (e.g. liquidity ratios, profitability measures, leading indicators, ...), but no measure or group of values has been universally accepted as a standard to evaluate portfolio performances. A portfolio can be measured against several criteria for analysis, but the analysis will always be subject to personal interpretation. Prototype development standard setting is not overly useful either. If formal evaluation specifications are written after the prototype is developed, a biased evaluation will most likely result.

There are two approaches which illustrate the two extremes in evaluating expert system performances [WEI-84]. One is the *anecdotal approach*, the other is the *empirical approach*. With the anecdotal approach, the model designers describe their good experiences, or those situations where the program performed well. They describe the situations, perhaps even recreating a session with the program, that duplicate the original performances. In those situations where the program performs poorly, attempts are made to correct the program. And therefore, as new problem situations arise, new information is incorporated into the model.

The second approach places its emphasis on the empirical evaluation of performances over many problem cases stored in a database. Some kind of rigourous testing procedure must be specified to compare the model-produced interpretations with independently obtained, conclusive interpretations for the same problem cases. Traditionally, the testing is carried out once the system is relatively well developed. While an empirical approach to testing may be clearly superior to an anecdotal one, implementing the methods and obtaining the representative cases for the database often comes up against severe practical obstacles. In some domains, such as medicine, it may be possible to gather large numbers of cases for the relatively common diseases, but rare diseases always present a problem in getting enough representative coverage for validation. In other domains, such as geological exploration, the cost of obtaining sample cases may be very high, and only a few of any type may be readily available.

Many dimensions exist in evaluating the effectiveness or utility of an expert system in an organisation - evaluations relating to [WOL-87] :

- user acceptance,
- performance of the system, completeness and accuracy,
- utility or value-added benefits,
- liability and risk,
- feedback to the knowledge engineers.

From the user's perspective, the system needs to have a smooth, efficient, and "natural" interface. The advice of the expert system must be useful, and an explanation and justification module needs to be present to explain its

decisions in case any questions should arise during the working session.

A critical performance and validation check is to execute several test scenarios that the human experts have already solved to see if the expert system produces the same answers. The evaluation is broken down into two areas : a *static evaluation* and a *dynamic evaluation*. The static evaluation is the testing of the knowledge base for consistency and completeness. Is the knowledge base sufficient in its representation of the domain ? Is it too limited a domain ? The dynamic evaluation is the testing of the reasoning process and advice given. Is the decision reliable and accurate ? The static and dynamic evaluations are best judged against the correctness of the advice from the expert system against the human experts.

The question of competence or performance can be weighed against the utility or value-added benefits of an expert system. By establishing an expert system, how much time and money did the organization save ? Did the expert system increase productivity ? Was it effective ? This evaluation is organisationally dependent, as each organization operates under a different utility function.

One aspect of the evaluation phase is the liability risk an organization can incur when using an expert system. Just how costly are expert system errors ? What happens if a problem exists, but the expert system does not act ? As the focus of decision makes shifts to computers, away from human experts, all sorts of liability questions arise.

And, as said before, evaluation enables a feedback process to take place, whereby the comments serve as a basis for iterative refinements. Establishing hurdle points and feedback evaluations will greatly enhance the chances that the expert system development is proceeding in the right direction.

CHAPTER 7 : TOOLS AND LANGUAGES

Once the knowledge for an expert system has been acquired, the knowledge engineer has to decide which software tools he should use to build the expert system itself. A wide range of tools, incorporating many facilities and features, is now commercially available. Choosing a tool is not just a question of selecting a convenient piece of software. The tool to be used usually dictates the knowledge representation system to be used, and so a decision about the knowledge representation has to be made first. Deciding the appropriate knowledge representation methods is one of the main skills an experienced knowledge engineer can offer.

7.1 BUILDING EXPERT SYSTEMS USING PROGRAMMING LANGUAGES

Expert systems can be built using different types of tools. The most basic tool is the programming language. Using a high-level programming language means building both the knowledge representation mechanism and the reasoning mechanism from scratch. The usual alternative is to use an expert system shell, which contains both a ready-made inference mechanism and some form of knowledge representation scheme.

The advantage of building expert systems in high-level languages is that it allows great flexibility. The inference mechanism and the knowledge representation system can be tailored exactly to the application in hand. The disadvantage is the amount of time taken to build the system, compared with using a shell, and the amount of effort that may go into re-inventing the developing facilities that are available in existing shell systems.

If a high-level language is chosen, there is a further choice. There is a wide range of traditional programming languages, and there are also specialised artificial intelligence programming languages.

7.1.1 Traditional and artificial intelligence languages [GOO-85]

A. Goodall considers languages in three classes : the first two classes are traditional languages; the third class gathers the artificial intelligence languages.

The first class of languages includes Algol, Basic, C, Cobol and Fortran. These languages suffer from a disadvantage; they can manipulate only a small range of data types. In all these languages, the range of types includes no more than numbers, logical values, and character strings (where as writing an expert system, the programmer needs to handle objects like rules, nets, menus, recorded explanations, and grammars).

The second class of languages includes Algol68 and Pascal. Unlike the first class, these languages do provide a convenient way of encoding rules, nets, etc ...

The third class of languages is the artificial intelligence languages of which the prime members are Lisp and Prolog.

With the two first classes of languages, the programmer must : break his problem into a set of subtasks to be implemented; apportion each subtask as a "procedure", "function", "subroutine" or "routine"; encode each procedure as a chunk of programming language text; compile the entire text, turning it into machine-code; run the machine-code. If he has designed the program correctly, then it will run the first time. But this rarely happens, and so the program must be debugged.

In Prolog and Lisp, the programmer can debug his program by feeding all of it into the Prolog or Lisp system. He can then call each procedure in turn and check that it does the correct thing and generates the correct result. The system will help with this task by giving "trace" output when requested : this output can show the sequence in which procedures have called other procedures, and can depict the value of various variables. By contrast, in the second class of traditional languages, the programmer must rewrite his program to incorporate statements which explicitly call each procedure to be checked, and which explicitly demand "trace" output through "write" or "print" statements. He must then recompile the program, and re-run it. If the debugging output is not adequate, he must repeat the process. It is also easier with artificial intelligence languages to test programs which are incomplete but still runable. Such programs can arise because a set of rules is not fully known, or because only part of an inference method has been decided on.

Having said this, why use traditional languages when building expert systems ? The most likely reason is that no other language is sold for the hardware on which the expert system is to run. The next reason is that some expert systems must run on small machines with not much memory. Implementations of the more advanced languages may take up too much space, run slowly, or not give the programmer the fine control he needs to save space and time.

7.1.2 Lisp

Lisp was invented in the 1950s by John McCarthy, who was motivated by a desire to implement a practical list-processing language for symbolic artificial intelligence work. Lisp allows the programmer to represent objects like rules and nets as "lists" - as sequence of numbers, character strings or other lists. Mathematical functions, predicate logic, logical connectives, and list manipulations can be applied to these lists. Lisp is a highly interactive, flexible, and recursive language. The major advantage of Lisp is its nesting nature, which lends itself to many problem-solving techniques, such as searching.

Several limitations were evident with early versions of Lisp: problems such as large memory requirements, extensive CPU demands and limited applicational use. But thanks to recent innovations in hardware design, Lisp has been given renewed power. Today, it is a more sophisticated language that has taken on a variety of dialects (descendants of Lisp include MACLISP, ZETALISP, COMMON LISP, FRANZLISP, INTERLISP, T-LISP, NLISP, LISP/VM, ...).

One of the primary reasons Lisp is so popular (especially in North America) is that a symbolic program is naturally represented in Lisp data structures. Programs and data have the same form and can thus be treated somewhat interchangeably, allowing Lisp users to write programs capable of running and modifying other programs. This allows an expert system program to make changes to lines of its own code while running.

The Figure 1.12 shows how a rule might be written as a list in Lisp [WOL-87].

```
IF      the time of the culture is recent
   AND  the site of the culture is blood
THEN    the locus of the culture is abdomen
   OR   the locus of the culture is pelvis


(IF
      (AND
            (of culture time recent)
            (of culture site blood)
      )

      (OR
            (of culture locus abdomen)
            (of culture locus pelvis)
      )
)
```

Figure 1.12 : example of rule in Lisp

7.1.3 Prolog

Invented around 1970 by A. Colmarauer and his associated at the University of Marseille, Prolog is a major competition to Lisp in the artificial intelligence community. The basis of Prolog is the notion of logic programming in which computation can be viewed as controlled, logical inferences. Prolog is a language suitable for applications requiring the simulation of intelligence - applications in such areas as expert systems, deductive databases, language processing, planning systems, and design applications.

It is not an algorithmic language such as Cobol or Pascal, but it is based on the concept of formal logic (predicate calculus). It dispenses with the notions of "goto", "do for",

and "while do", and instead incorporates the mechanisms required by intelligent programs - advanced pattern matching, generalized record structuring, list manipulation, and depth-first search strategy based on backtracking.

Prolog uses symbolic representations of the objects and relationships between those objects to specify known facts and relationships about a problem, thus creating "clauses". Clauses make up the program, with the conclusion being stated first. Prolog's power lies in its ability to infer facts from other facts. The user can give the computer nonnumeric information and have it deduce additional nonnumeric information.

Another advantage of Prolog is its compactness. A three-page listing in Lisp can be condensed to one page of Prolog. This can be accomplished because control is implicit - it is already provided by the system - whereas in Lisp, a control system must be written to run the application.

And finally, since a Prolog program is a series of statements in logic, it can be understood declaratively. That is, it can be understood quite separately from considerations of how it will be executed. Traditional languages can only be understood procedurally. That is by considering what happens when the program is executed on a computer. This is an important issue because it takes us one step nearer the goal of all programming languages - being able to specify *what* is to be done and leaving the *how* up to the computer.

Prolog, like Lisp, has its language derivations, such as PROLOG-2, MPROLOG, LM-PROLOG, C-PROLOG, QUINTUS PROLOG, TURBO PROLOG, ...

7.2 BUILDING EXPERT SYSTEMS USING DEVELOPMENT TOOLS

Instead of building an expert system from scratch, it is sometimes possible to speed up the development stages with the use of *expert system shells*. Shell systems are generally regarded as knowledge engineering tools that include most of the elements necessary to build a complete expert system. The intention is that shell systems should attract applications from many different subject areas. Basic inference procedures, for example, are quite universal, whether used in medical diagnosis, chemical analysis or military planning.

A shell system usually contains [WOL-87] :

- a predefined inference engine that knows how to use the knowledge base to reach conclusions,
- a knowledge base development engine (editor) for constructing and editing the knowledge base,
- a spelling checker to make sure words are typed correctly,

- an on-line logic reasoning base for explaining how and why a conclusion was reached,

- performance monitoring tools for testing and debugging the expert system during development; this enables the knowledge engineer to set trace and break point conditions (for debugging purposes) based on the knowledge base versus a location counter in the program,

- a graphic/windowing package for ease of interactive use to illustrate information and relationship, allowing knowledge engineers to switch from one environment to another without losing context,

- integration with traditional software tools such as word processors, spreadsheets, and communication programs.

Figure 1.13 illustrates the major components of an expert system shell [WAT-86].

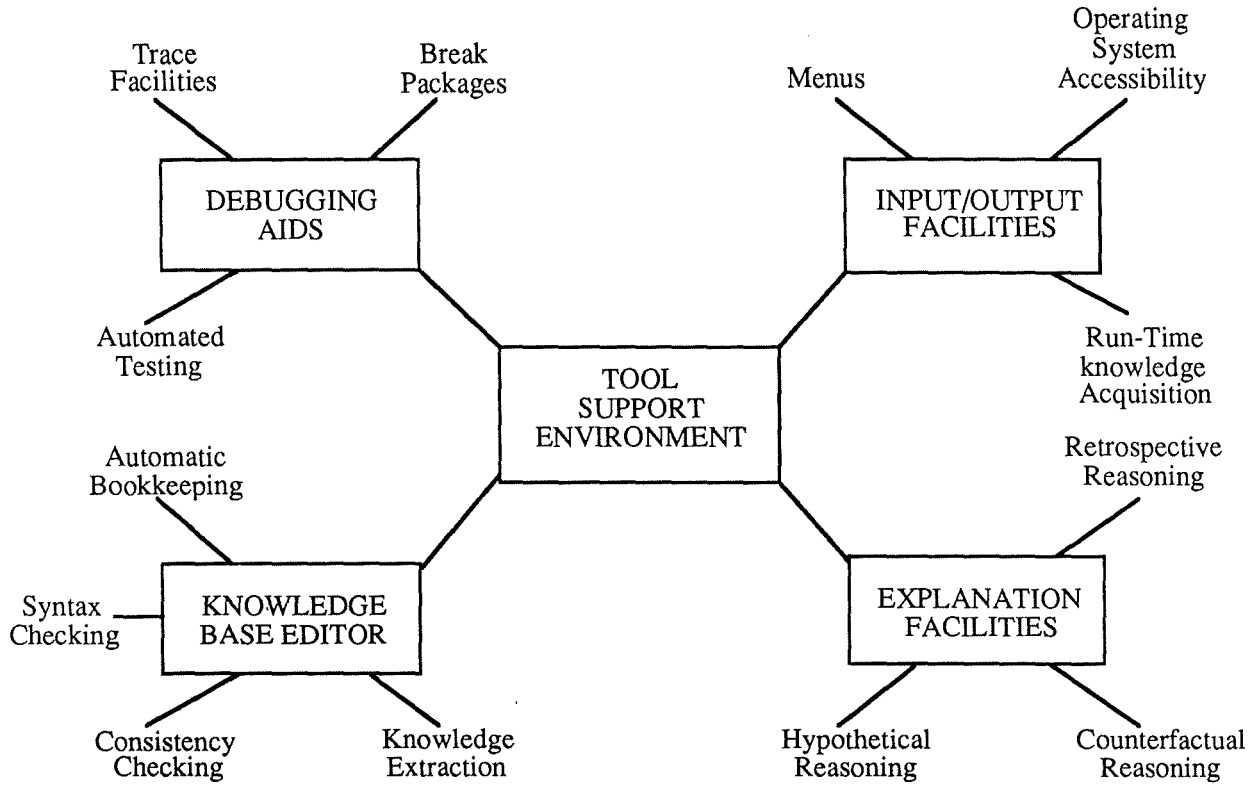

Figure 1.13 : major components of a shell

The advantage of starting with a development tool is that the knowledge engineering acquisition tools and utilities have already been built. Development tools can drastically reduce the time spent to create the prototype, allowing for more time to test and debug the prototype. Another reason in favour of purchasing a development tool is to exploit an existing rule language and inference engine.

The disadvantage of a development tool is that it will generally embody only one reasoning methodology and knowledge representation technique, while sophisticated applications often require a combination of techniques. Most development tools available are rule-based. The knowledge engineer is thus restricted to construct the knowledge base as a series of If-Then, or If-Then-Else statements. Another possible disadvantage of development tools is cost. Minicomputer and mainframe development tools, while declining in cost, still require a substantial investment (prices range from $1.000 to $18.000).

Some of the well-known tools available are PERSONAL CONSULTANT PLUS, KES, S1, LOOPS, EMYCIN, HEARSAY-III, APES, ES/P ADVISOR, EXPERT-EASE, M1, KEE, AGE, GURU, TIMM, RULEMASTER, EXSYS, ...

CHAPTER 8 : THE FUTURE

In this chapter, we discuss the future of expert systems. But we should warn against being too optimistic. In the first chapter, we showed how two approaches to artificial intelligence seemed, in their early stages, to promise much more than they could deliver - these approaches were the cybernetic neural-net approach to artificial intelligence and the General Problem Solver. Due to the focus on specific knowledge, expert systems can solve problems which general methods could previously not touch. But they can not solve all problems ! Other techniques, perhaps not yet dreamed of, will cope with the remaining tasks.

8.1 COMMERCIAL TRENDS [HU-87]

Most forecasts for expert system market are included in those for artificial intelligence. However, more than 25% of the artificial intelligence market is for expert systems. Estimates of the combined artificial intelligence / expert system market size are shown in Figure 1.14.

YEAR	DOLLARS
1983	$ 80 million
1984	148 million
1985	250 million
1990	3-12 billion
1995	40-70 billion
2000	50-120 billion

Figure 1.14 : artificial intelligence / expert system market estimates

These estimates were made in 1985 and anticipate a general trend in market growth for artificial intelligence / expert system of about 60% compounded annually.

The market size for expert systems alone is shown in Figure 1.15. Since the late 1970s, the expert system market has been the fastest growing segment of the artificial intelligence industry. This market segment grew from $9 million in 1982 to

$74 million in 1985, and would grow to $810 million (projected) in 1990. The market growth rate during this period is averaged at 65% annually. The number of new start-up companies pursuing the expert system market increased from 7 in 1982 to 28 in 1985. And many of these companies are expanding rapidly.

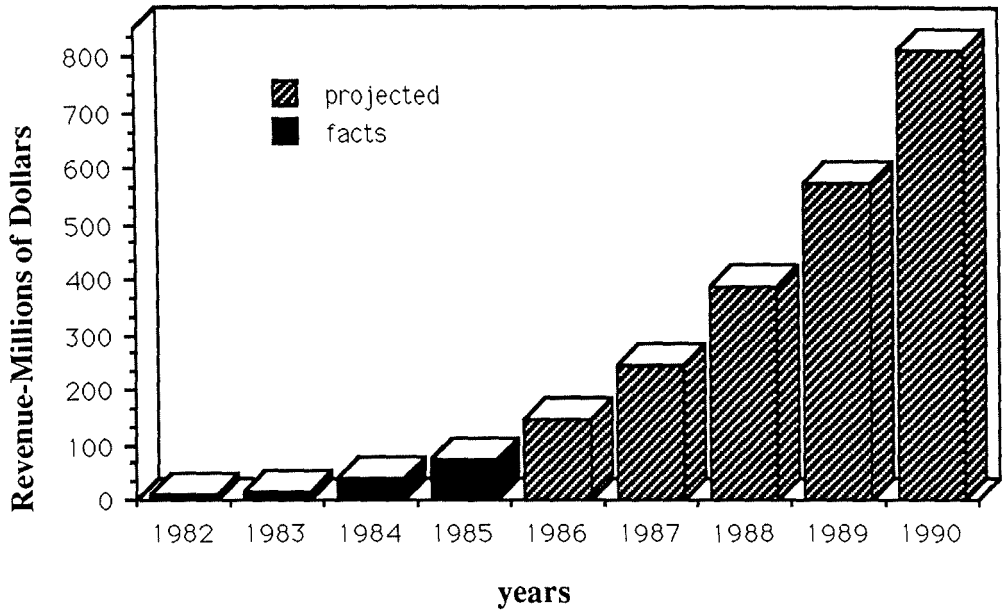

Figure 1.15 : expert system market growth

8.2 FUTURE DEVELOPMENTS IN EXPERT SYSTEM TECHNOLOGY

According to [MAR-88], improvements will be made in four areas:

Knowledge representation. Existing knowledge representation schemes will be extended to handle inheritance directly and to handle hundreds of thousands of rules. Concurrently, new knowledge representation schemes will be developed and provided in new expert system tools or new releases of existing tool products.

Knowledge acquisition. The majority of the time spent in developing an expert system, when using an expert system shell, is dedicated to knowledge acquisition activities. Decreasing costs related to knowledge acquisition would significantly decrease expert system development costs. Decreasing costs can be achieved through automated editors and self-learning systems. Self-learning systems are truly the hope of the future. With these systems, knowledge acquisition could be accomplished much more quickly. However, research in this area has yet to produce effective tools. The only tools that have been produced so far are induction tools. Human

experts gain a lot of their expertise from experience. Each event helps human experts by providing them with new data about the subject domain. As they experience more events, they find meaningful causal relationships between the various pieces of data. Attempts have been made to develop knowledge acquisition tools based on this inductive process, but limited success has been realized so far. Another area of interest in self-learning research is tied to another knowledge source used by human experts : textbooks and subject-specific magazines. It is hoped that someday, we will use electronic text scanners to read this material and then use an intelligent knowledge acquisition program to derive domain knowledge from this material.

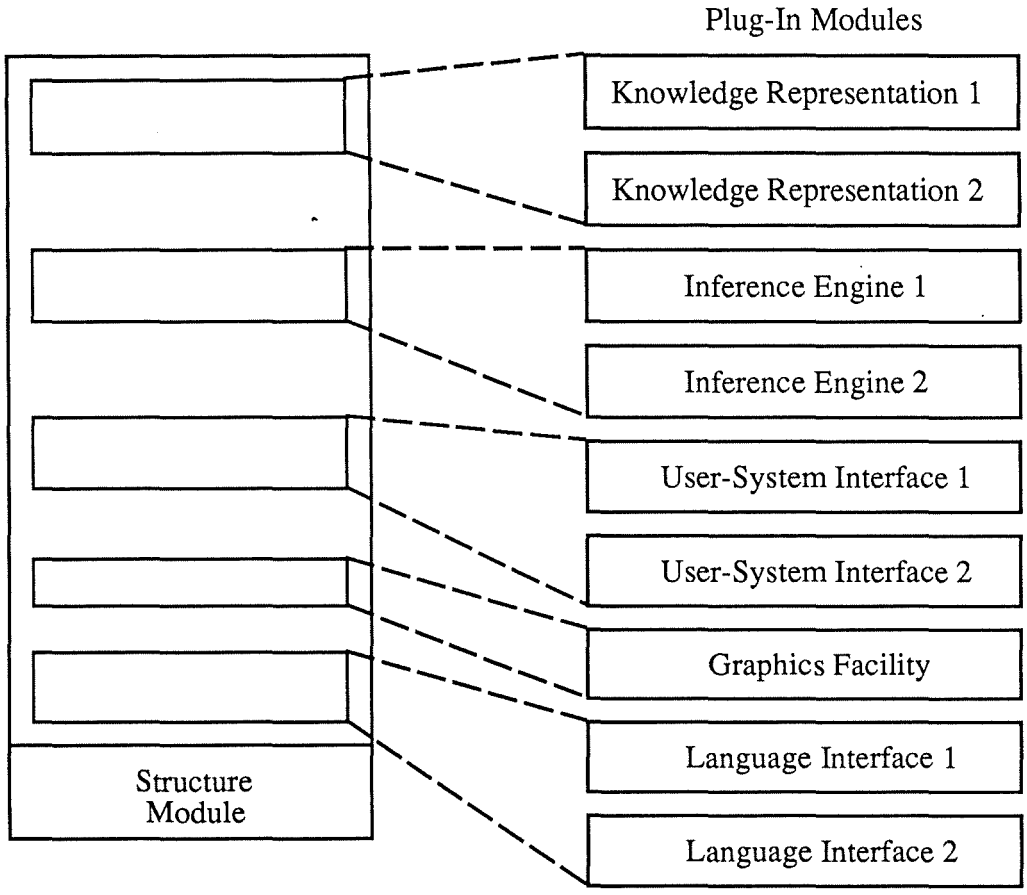

Figure 1.16 : structure of future expert system shells

Expert system tools. The second generation of expert system shells should be built in a modular fashion. The basic shell product will be a structural module. As shown in Figure 1.16, this module will provide some basic development capabilities, such as general knowledge data management and a knowledge engineer building tool interface. This module will be built to accept other more specific function modules. These function modules will plug into the structural module and provide the expert system shell with requisite functions. These function modules will provide various functional capabilities; for example, there might be two inference engine modules

available, one for backward chaining and one for forward chaining.

Expert system interfacing. User interfaces of future expert systems might be integrated with speech recognition and synthesis systems. Users will be able to vocalize their queries and responses, and expert systems will provide questions and consultations to the user verbally. These speech interfaces will be helpful to users such as machinists who have their hands full. Other interface system possibilities include vision and advanced graphics.

In short, the future of expert system technology looks promising. The successes already experienced with the technology, though modest, have generated great interest, research supports and business development. The research should provide us with more capable systems that can be developed with fewer resources, particularly human resources.

SECTION 2

EXISTING EXPERT SYSTEMS IN THE COMPUTER SYSTEM DOMAIN

The idea of on-line monitoring or controlling of a computer by another is not new. Watch-dog processors and maintenance processors have been designed to assist in the recovery from software errors and hardware errors while the subject computer is in operation. What is new is the application of an expert system approach to the control of computer operations.

Since we have used this new approach to build a prototype of a spool scheduler expert system (described in section 3), a better knowledge of the few existing expert systems dedicated to the computer system domain appears to be helpful. Therefore, this section will give a short overview of what has already been done in the computer system domain. Some of the best known expert systems in that field will be shortly analyzed, others will be quickly described and the remaining systems will just be mentioned.

CHAPTER 1 : XCON

AN EXPERT SYSTEM FOR CONFIGURING COMPUTERS

1.1 INTRODUCTION

The computer manufacturer DEC (Digital Equipment Company) is well-known for the flexibility of its Vax computer range. For instance, a customer who needs a Vax has a choice of at least four or five kinds of processor, perhaps ten models of terminal, and disk drives with capacities going from 50 megabytes to at least 300 megabytes. A customer who orders a Vax will explicitly specify some components, such as the processor type, and the size of memory. He will leave other components, perhaps the kind of power supply, for DEC to decide on.

When given an order, Dec must first check that it is complete. It must then see whether all the items are compatible. After which it will then lay out the floor plan and finally design the cabled connections. As the complexity of its systems increased, the company found it harder to carry out these configurations; several attempts to write conventional programs to solve this problem failed and there were not enough skilled people in the company to keep up with demand. Moreover, DEC wanted to avoid scenarios like the following : delivering a complete $500.000 Vax system to the customer's site but forgetting a small $5 cable. Although allowance losses like this add up over time, there is a more significant cost in terms of lost time and goodwill when the customer must wait for something as minor as a missing cable.

To counter all these problems, XCON, also known as R1, began in 1978 as a joint effort between DEC and Carnegie-Mellon University. Two years later, DEC took the prototype out of the university environment and started to use it in the real world of day-to-day manufacturing.

XCON configures Vax systems at a very detailed level based on its task-specific knowledge. It determines necessary modifications on each order, produces diagrams of spatial and logical relationships between hundreds of components in a complete system, and defines cable lengths between components.

1.2 INSIDE XCON

XCON contains about 4500 rules[1] that are located in a section called production memory. Another section of memory where the configuration work is actually performed is called working memory. At any time, the working memory contains descriptions of certain database components and of partial configurations that have been determined so far.

XCON's approach to configuration can be seen as a search through all its rules to find one rule whose conditions match the items that happen to be in the working memory at that moment. Every time XCON instantiates a rule, it adds, deletes or modifies an item in the working memory. The changes in working memory indicate increasing progress in configuring computer systems. As a result of the changes to working memory, the next time XCON cycles through its rules, it finds new matching patterns plus old ones that no longer match. In this way, XCON progresses from configuration task to task or context to context. It begins with a starting state that contains a description of the components in the customer's order. It progresses through a series of intermediate states that describe partial configurations and the as-yet-unconfigured components. Finally, it reaches a solution state (when no more rules can be fired), in which the entire computer system is configured. A consequence of this pattern-matching technique is that XCON almost never generates any dead-end paths. Therefore, it rarely has to backtrack to an earlier situation in order to find the true path.

The rule DISTRIBUTE-MB-DEVICES-3 for the distribution of massbus devices among the massbuses in the Vax computer is shown in Figure 2.1 [HU-87].

This rule shows that there are 6 conditions to be met before one of the single-port disk drives on the order is assigned to one of the massbuses.

[1] by the end of 1986.

Rule: Distribute-MB-Devices-3
IF : The most current active context is
distributing massbus devices
And there is a single-port disk drive that
has not been assigned to a massbus
And there are no unassigned dual-port disk
drives
And the number of devices that each
massbus should support is known
And there is a massbus that has been
assigned at least one disk drive and that
should support additional disk drives
And the type of cable needed to connect the
disk drive to the previous device on the
massbus is known

THEN : Assign the disk drive to the massbus

Figure 2.1 : example of rule in XCON

1.3 THE BENEFITS OF XCON

It took some time to incorporate all the knowledge of how to configure in XCON, but by the end of 1983 it was achieving 98% correct configurations. XCON has allowed DEC to serve 4 times as many orders without increasing the number of human experts. XCON performs jobs that were previously undertaken by an experienced technical editor much faster and more accurately. For example, a Vax system order that may take the editor as long as 20 minutes to examine and determine what computer components need to be replaced or added can typically be completed by XCON in less than a minute. XCON has also reduced the error rate on orders from 35% to 2%. The total savings for DEC between 1980 and 1985 is estimated to be about $12 million. This saving does not take into account customer satisfaction with accurate and on-time delivery and related reduction in labour and material cost.

1.4 A KNOWLEDGE NETWORK

XCON is now firmly entrenched and routinely used at DEC. Since its acceptance, the DEC artificial intelligence group is integrating a series of other expert systems with XCON. These integrated expert systems will eventually form what DEC calls a *knowledge network* to integrate sales, engineering, configuration, manufacturing, distribution, installation and field service [RAU-88].

Among the expert systems shown in Figure 2.2, we find :

- XSEL which helps salespeople quote prices on multiple versions of configured computer system orders at their point of sale,

- XSITE which is used by field service people for site preparation and site management,
- ISA which schedules customer system orders against the current and planned material allocations,
- IMACS which schedules the floor capacity for the system being built, manages inventory and manages problems that arise during the manufacturing process,
- ILOG which helps manage the distribution of the computer products from the plant to the customer,
- INET which helps make organizational and distribution management decisions.

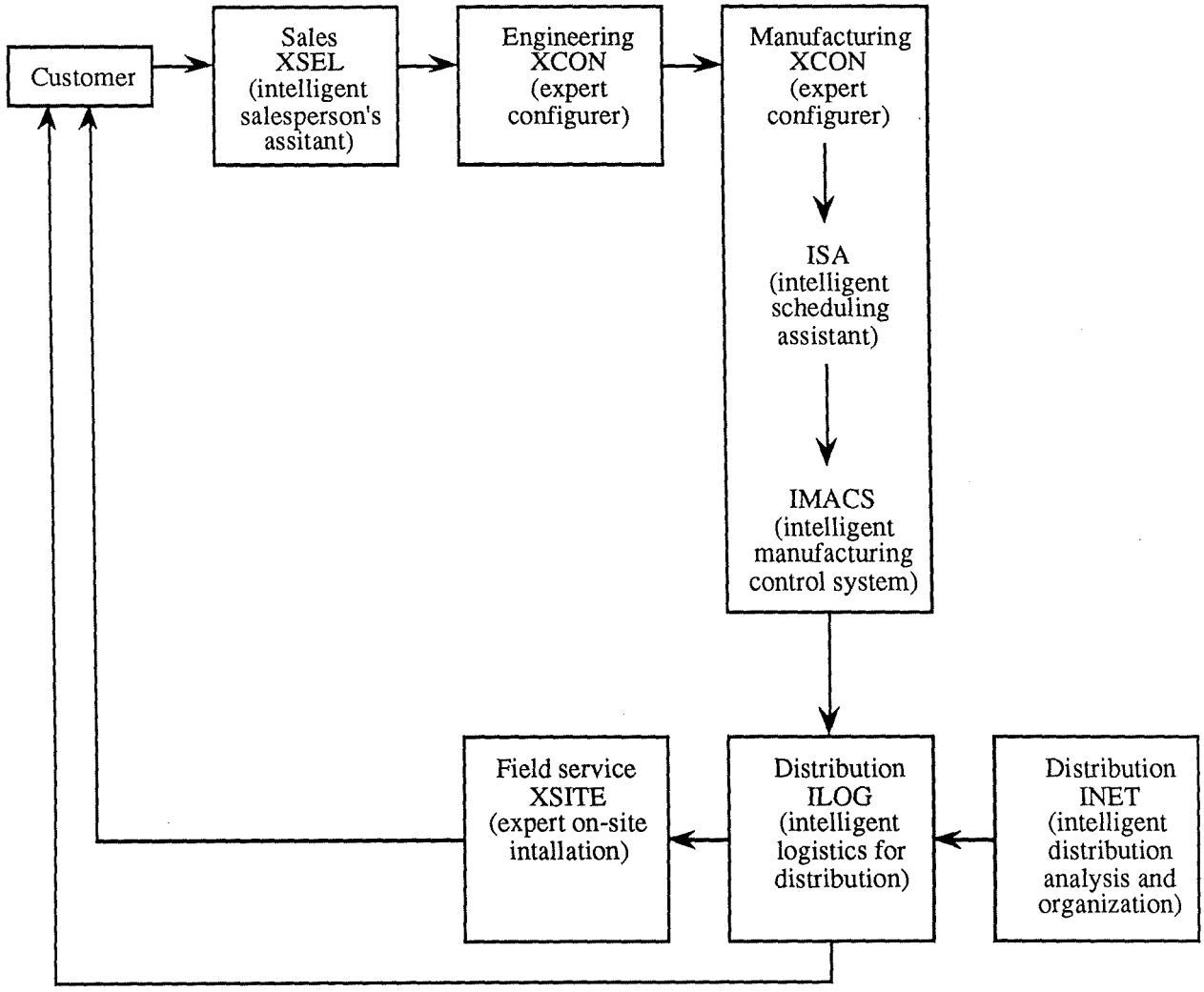

Figure 2.2 : DEC's expert knowledge network

CHAPTER 2 : YES / MVS

A CONTINUOUS REAL TIME EXPERT SYSTEM FOR MVS OPERATORS

[GRI-84]

The Yorktown Expert System for MVS[2] operator (YES/MVS) is a continuous real time expert system that exerts interactive control over an operating system as an aid to computer operators.

2.1 INTRODUCTION

Computer operation is a monitoring and problem-solving activity that must be conducted in real time. It is becoming increasingly complex as data processing installations grow. The control of a typical large system rests largely in the hands of just a few operators. Beside carrying on such routine activities as mounting tapes, loading and changing forms in printers, an operator continuously monitors the condition of the subject operating system and initiates queries and/or commands to diagnose and solve problems as they arise. To deal with such complexity, operators and system programmers often rely on many "rules of thumb" gained through experience. A long training period is also required to produce a skilled operator. The resulting shortage of skilled operators and the increasing complexity of the operator's job call for more powerful installation management tools. Therefore, the expert system approach was a natural choice.

There are many new requirements in building a real time expert system to assist a computer operator. The environment is too complex and dynamic for obtaining information by querying the human operator. Unlike many other expert systems, this means that conclusions are based on primitive facts obtained directly from the system being monitored and not from human interpreted inputs.

To be able to handle real time on-line data, the inference engine needs to be mainly data driven. The OPS5 production system was chosen primarily for this reason.

---

[2] MVS : Multiple Virtual Storage operating system is the most widely used operating system on large IBM mainframe computers.

2.2 THE DOMAIN OF YES/MVS

The MVS operating system running with a Job Entry System (JES), puts out various system messages to the operator. While there are literally hundreds of different of messages, the number that are relevant to an operator is much smaller. The majority of purely informational messages may usually be statically filtered and diverted to a log. Even then, the peak message rate from MVS to the operator sometimes exceeds 100 a minute.

When a problem is detected, the operator may query MVS for additional information and send one or more corrective commands. The operator must often anticipate informational needs and dynamically keep track of a number of relevant status variables. There are many different subdomains in the domain of operator activities. The six subdomains described below were selected for the implementation of the expert system :

- Jes queue space management
- problems in channel-to-channel links
- scheduling large batch jobs off prime shift
- MVS detected hardware errors
- monitoring software subsystems
- performance monitoring

2.3 THE SYSTEM ORGANIZATION

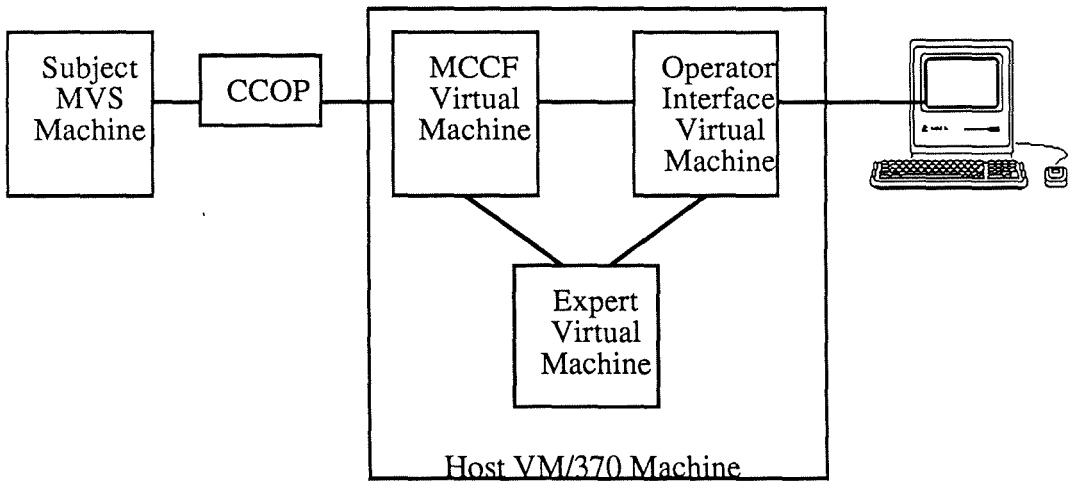

Figure 2.3 : YES/MVS system organization

Because YES/MVS and the subject MVS system are resident in different computers, problems in MVS do not interfere with the operation of the expert system. YES/MVS is partitioned into three virtual machines (as shown in Figure 2.3). One of these contains the MVS operator expert, the second one contains the MVS Communication Control Facility (MCCF), and the third one is

used to control the YES/MVS operator's display console. The MCCF communicates with the MVS system through a separately developed facility, called CCOP. The CCOP provides centralized control and filtering of messages to the computer and their operators.

The YES/MVS operator console displays on-line messages on the top level input screen describing events that are related to the various tasks YES/MVS is concerned with. The operator may select one of the displayed messages and request further detail. The detail level screen contains the recommend action or information along with an explanation. If an action is called for, the operator is given the choice of automatically issuing the command, showing that he will manually type the recommended command at another terminal, or rejecting the command being proposed. If the command is rejected, the operator is prompted to enter the reasons for the rejection which is fed back to the YES/MVS knowledge engineers.

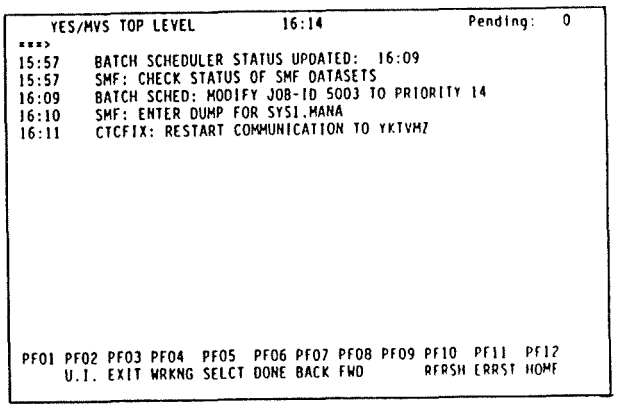

Figure 2.4 : YES/MVS operator console top level screen

2.4 BUILDING THE KNOWLEDGE BASE

Most of the expertise is encoded in over 500 OPS5 rules distributed between the expert virtual machine and the display control virtual machine. Some of the expertise was encoded in relational tables. Some expertise was implemented in the MCCF translation tables for more direct execution. Therefore, the knowledge base is not restricted to the rule base only.

Figure 2.5 is an example of a pair of rules that stop the reception on an incoming link with JES queue space that is critically low and restart the reception when it improves.

YES/MVS extends the use of expert system techniques to continuous real time control applications. The success gained with this approach allows applications of expert system technology to other areas of computer installation management : capacity planning, configuration and installation..

```
(p stop-reception
      (Task   ^task-id jes-q-space)              ; If the task of
      (JES-Q          ^mode panic)                       ; maintaining JES-q-space
      {<the-Link>(Link  ^id <L-id>               ; is active, the space
            ^status <<active i/o-active>>        ; is critically low, and
            ^receive yes)}                             ; there is an active

      (Call remote-make                          ; receiving Link
            Link-command  ^id <L-id>             ; then cut the Link
                          ^receive no            ; and mark the Link
                          ^rm-to: MCCF)          ; reception status as
      (Modify <the-Link>    ^receive to-be-no))  ; about to be no.


(p start-reception                               ; If the task of maintaining
      (Task             ^task-id jes-q-space)              ; JES-q-space is active,
      (JES-Q  ^mode <>      ¬panic)              ; the space is not
      {<the-Link>(Link      ^id<L-id>            ; critically low, and
            ^status <<active i/o-active>>        ; there is an active Link
            ^receive no)}                        ; not receiving,

      (Call remote-make                          ; then
            Link-command  ^id <L-id>             ; reopen the Link
                          ^receive yes           ; and mark the Link status
                          ^rm-to: MCCF)          ; as about to be yes.
                        (Modify <the-Link>   ^receive to-be-yes))
```

Figure 2.5 : rules from the JES queue space subdomain

CHAPTER 3 : PERMAID

A MULTIPARADIGM KNOWLEDGE-BASED SYSTEM FOR DIAGNOSIS OF LARGE MAINFRAME PERIPHERALS [ROL-87]

3.1 INTRODUCTION

PERMAID is a mutiparadigm expert system that is used for diagnosis and maintenance of approximately 10.000 large fixed-head disk subsystems that are components of mainframe computers. PERMAID performs three primary functions :

- trouble-shooting of observed problems,
- predictive maintenance,
- media and file recovery.

It also provides training as a secondary function. The trouble-shooting function is used by a field service engineer to identify and repair faults in the disk subsystem (i.e. the disk drive, disk controller and the associated operating system software). The media and file recovery function is used by the field service engineer in collaboration with the customer's site representative to perform file recovery and to perform "soft repair". The predictive maintenance function is used to direct the field service engineer through a routine "check up" procedure that determines whether any preemptive maintenance action should be taken to prevent serious faults from occurring.

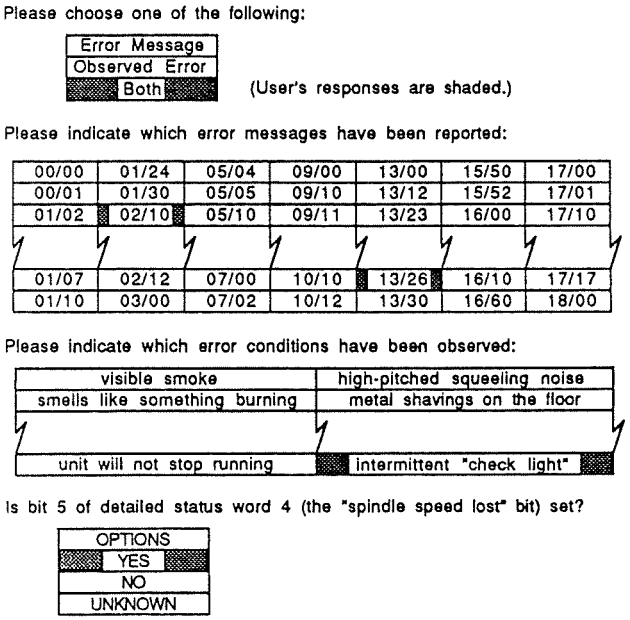

Figure 2.6 : example of PERMAID trouble-shooting session

The trouble-shooting portion of PERMAID operates using a question-and-answer consultation model. After the user has indicated which initial symptoms are present, the trouble-shooting process asks the user to run tests and make observations that are required to identify the problem. Figure 2.6 shows an example of a small part of PERMAID trouble-shooting session.

3.2 INTERNAL DESIGN

The heart of the system is the "fault-resolution kernel" which is used to identify faults. The knowledge representation for this kernel is based on the use of frames and networks.

PERMAID supplies explanation information that is tailored for the five different types of user : expert, knowledge engineer, specialist user, field service engineer, and trainee.

At the end of each session, a complete trace of the session is written to a long-term history file that is used to analyse and modify PERMAID's performance.

The kernel knowledge base is a directed acyclic graph that represents cause/effect relationships (the graph currently includes approximately 2500 nodes). A portion of the kernel knowledge base is shown in Figure 2.7 in the form that is displayed by the knowledge base editor.

Note that the structure is a graph (and not a tree) because any given cause can have many effects and any given effect can have many causes. Conceptually, each node is a frame. Each frame includes a number of slots that contain information that is either local to the particular frame or inherited from a higher frame.

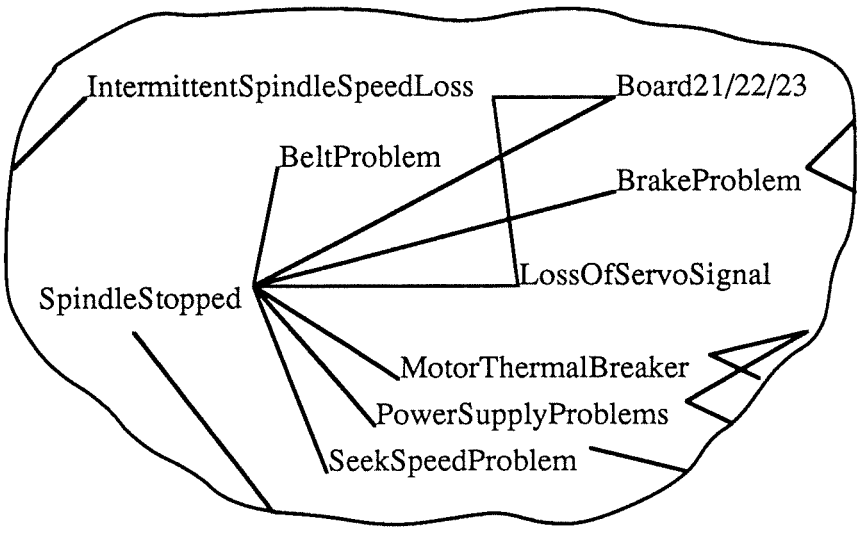

Figure 2.7 : portion of the PERMAID kernel knowledge base

3.3 THE INFERENCE ENGINE

The trouble-shooting inference process begins with the user selecting any number of starting symptoms. The initial symptoms are then evaluated to select one primary symptom that will be the starting node for the search process. Once the primary symptom has been identified, the process is based on the systematic selection and test of possible causes for a given effect. Once the cause of a given effect is identified, the process "moves" to the new cause and is recursively restarted with the newly identified cause being treated as an effect.

A bad spot on a disk can produce a solid visible error. It is possible, however, for a bad spot to result in many retriable errors. In this case, no explicit error message is printed but information is logged on the site error log. This type of situation is detected by the use of the predictive maintenance function. Processing in this section initially focuses on the interpretation of an error-summary log which isolates potential problems to specific areas of the disk subsystem and then classifies possible problems within the isolated area.

The heart of the support-environment is a graphic-oriented knowledge base editor that displays the graphs on which PERMAID is based. The graphic display is updated dynamically during execution. Nodes that are on the current search path are darkened and the node that is currently being tested is shaded. As the selection process decides on the next node to expand, each of the nodes that are being considered flash as they are being considered.

3.4 PERFORMANCES

The first use of PERMAID was for training, but it is increasively used in every day life. However, it is important to realize that even if the system is perfectly implemented, it may still produce advice that is "incorrect" in the sense that it doesn't actually fix the problem, but "correct" in the sense that it is the best possible advice. Let us say that the performances of PERMAID are not as good as that of an expert but are really better than the typical field service engineer.

CHAPTER 4 : AI-SPEAR

A COMPUTER SYSTEM FAILURE ANALYSIS TOOL [BIL-84]

4.1 INTRODUCTION

Computer system failure may be caused by hardware or software faults, system load or other factors. Many intermittent failures can be analyzed using symptom-directed diagnosis based on events in the error-log. This technique is being used by major manufacturers of computer hardware. Bossen reports on its use in IBM systems and DEC have introduced SPEAR (Standard Package for Error Analysis and Reporting), a library of such programs for their major operating systems.

A further step was made when DEC created a rule-based implementation of SPEAR's analysis function, leading to an expert system called AI-SPEAR. DEC implemented approximately 150 SPEAR theories (which require about 700 OPS5 rules) and uses AI-SPEAR as an in-house tool.

4.2 PROBLEM DESCRIPTION

SPEAR's knowledge of how failures relate to faults is summarized by a set of rules ("theories" in SPEAR parlance), each of which hypothesizes the presence of one or more hardware faults. With the previous implementation of SPEAR, there is no way for a user to ask for an explanation of the reasoning that led to a conclusion. Many of the SPEAR theories are simple enough for a complicated explanation to be useless. However, some of the conclusions are triggered by a complicated juxtaposition of events, or are reached by a long chain of inferences.

The previous version of SPEAR does not handle differential diagnosis in hierarchical hardware configurations correctly . If two tape drives connected to a single controller each report errors, the diagnosis made by SPEAR is that something is wrong with the controller since it was the common element in the observed failures.

For all these reasons, DEC felt that these sorts of analysis of failure patterns would bring a lot more satisfaction in a rule-based expert system.

After a careful study of available rule-based languages and systems, the OPS5 production system was chosen for AI-SPEAR. This was encoded since a version has been developed within DEC and since the XCON program has shown that OPS5 can run well with a large number of rules.

4.3 EXAMPLE DIAGNOSIS

Figure 2.8 is excerpted from runs of AI-SPEAR. This diagnosis demonstrates the program's ability to explain its inference process.

```
$ run tap01

AI-SPEAR TAP V1.2
TU78 Tape Analysis Program

Error file: rep322.dat

34 records read from file: rep322.dat

**************************************************
THEORY-2.3.3.22 B.2
TM Fault B - Read Path Failure Error

Diagnosis:

Replace the M8950 board for channel 3

Additional Information:

device name:          MFAO Serial Number:  3737

eccsta-err:           0
chxtie:               3
earliest time:        1983-08-07 18:05:36.00
latest time:                1983-08-07 18:35:36.00
error count:          12

--------------------------------------------------------------------------

Do you want an explanation? y

TM Fault B - Read Path Failure Error was inferred because
int-code is 32 and failure-code is 27

It was next observed that eccsta-err is 0

It was next observed that chxtie is not 0

leading to the stated diagnosis.

                Press return to continue . . .
```

Figure 2.8 : excerpt from runs of AI-SPEAR

CHAPTER 5 : OTHER EXISTING EXPERT SYSTEMS

5.1 HUMAN

Unix is a high performance operating system, but unfortunately, it has always been the antithesis of user-friendly. Therefore, a company named Cognosys in conjunction with British Telecom has developed a product called HUMAN. HUMAN is an expert system consisting of a set of 30 files including 800 to 1000 rules each. Human appears to the user as an extension of Unix itself and provides easy to use assistance to operators. The product covers four areas of administrative tasks: user and group administration, adding printers, terminals, and backup management. The result is a faster and easier way of obtaining information from booklets which often hold a simple solution to a problem within pages of intimidating complexity.

5.2 CRIB [WAT-86]

CRIB is an expert system which helps computer engineer and system maintainers locate computer hardware and software faults. The engineer gives the system a description of his observations in simple English-like terms. CRIB matches this against a database of known faults. By successively matching larger and larger groups of symptoms with the incoming description, CRIB arrives at a subunit which is either repairable or replaceable. If a subunit is reached and the fault is not cured, the system backtracks automatically to the last decision point and tries to find another match. CRIB contains hardware and software fault diagnosis expertise as a collection of action-symptom pairs where the action is designed to elicit the symptom from the machine. CRIB models the machine under diagnosis as a simple hierarchy of subunits in a semantic net. This system was developed by ICL.

5.3 IDT [WAT-86]

This expert system assists technicians to locate the field replaceable units that should be replaced to fix faults in PDP 11/03 computers. The system uses knowledge about the unit under test, such as the functions of its components and their relation to each other, to select and execute diagnostic tests, and to interpret the results. The system is rule-based and was written by DEC in OPS5.

5.4 MESSAGE TRACE ANALYSER [WAT-86]

This expert system helps debug real time systems such as large telecommunication switching machines containing hundreds of processors. The MESSAGE TRACE ANALYSER examines interprocess message traces, identifying illegal message sequences to localize the fault within a process. The system considers the sender process identifier, the receiver process identifier, the message type, and the time stamp fields of messages in the trace. General debugging heuristics and facts about the specific system being debugged are represented as rules and applied using both forward and backward chaining. The system contains a limited explanation facility that allows it to answer questions about its reasoning. MESSAGE TRACE ANALYSER is written in Prolog and was developed at the University of Waterloo.

5.5 OTHER PRODUCTS [HEN-85] [WOL-87]

A few more products exist such as :

- EDD, to design databases
- EXSYS, to develop software from entity-attribute relationship information provided by system analysts
- PQCC, to develop compilers from supplied language and target machine specification
- PSI, to compose computer programs based on descriptions of tasks to be performed
- CDX, to analyze VMS dump files after system crash
- PROUST, to analyze Pascal programs written by novice programmers
- QUEST, to search databases
- DIAG8100, to diagnose failures in data processing equipment
- FAULTFINDER, to diagnose failures in disk drives
- PROGRAMMER'S APPRENTICE, to assist programmers with software construction and debugging.

SECTION 3

A PROTOTYPE OF A SPOOL SCHEDULER EXPERT SYSTEM

This third section focuses on the four-months' work we have realized in collaboration with Siemens Software. The detailed analyse of this work will enable us to illustrate the more theoretical concepts described in section 1.

Since the main objective of this work was to build a prototype of a spool scheduler expert system, the first chapter of this section focuses on a brief overview of the existing expert systems developed in the area of scheduling problems. The second chapter gives rudimentary knowledges about the structure of a spool. The final five chapters apply the general methodology for building expert systems to our problem.

CHAPTER 1 : SCHEDULING PROBLEMS AND EXPERT SYSTEMS

The scheduling of logistical activities and the assignment of discrete resources (people, machine, etc ...) to tasks or responsibilities are well-studied problems, and algorithms have been developed to determine optimal or near-optimal schedules. People have been using the techniques of linear programming[1] to do the scheduling for a long time, and apparently, it works well [KEL-87]. But there are some important restrictions on mathematical optimization problems, the most severe being that all the numbers in the function to be optimized must be known constants, and all of the constraints must be well-defined. In other words, the problem must be a well-defined, deterministic optimization. For many situations involving production resources, these requirements can be closely approximated. But in some other situations, many of the criterions are not well-defined or deterministic, and so the methods of heuristic inference may be applicable.

Some expert systems in the area of scheduling have already been developed [SIL-87]. An example of such a system is NUDGE, an expert system for scheduling business meetings. The user enters a set of requirements for the meeting (e.g. the type or purpose of the meeting, who is to attend, a range of times within the meeting should be held, etc ...), and NUDGE interacts with its user to find a suitable time, place, and attendance list. Other expert systems for scheduling and assignment have been developed in the problem domains of factory scheduling, office automation, and human resources management. An example is ISIS, a frame-based system for factory scheduling. ISIS performs a constraint-directed search for an acceptable production schedule when the constrains involve due dates, inventories, costs, schedule stability, resource availability, machine capacities, precedence of operations and certain preferences of factory management (e.g. a preference for using one machine over another whenever possible). Another such system is ODYSSEY, a frame-based expert system for scheduling business trips. ODYSSEY contains procedures for resolving inconsistencies and ambiguities in a user's stated trip requirements. Another system is OMEGA, a rule-based expert system for personnel assignment. OMEGA helps the user to assign available personnel to job openings when inconsistencies in job requirements, personnel qualifications, and the availability of travel funds for reassignment must be resolved.

As we can see, the approach of using expert system techniques in the area of scheduling is not new and a few products already exist on the market. But we will see later on

[1] Linear programming is one of many mathematical techniques for optimizing certain well-defined combinations of variables. It has been used often for dynamic production planning, as well as distribution and product mix problems.

that a similar (and still experimental) approach can be used to resolve spool scheduling problems.

CHAPTER 2 : THE STRUCTURE OF A SPOOL

What is a spool ? A spool is an independent part of the control system in an operating system. By means of independent internal task executions, a spool controls input/output operations for particular device families either while the program processing continues or after it has been completed. In other words, a spool provides coordination between rapid processing in the central processor and the less rapid input/output operations.

Spool input/output operations comprise :
- reading / punching punch cards,
- reading files from / writing files to floppy disks,
- printing listings,
- outputting files to tapes,
- controlling remote batch tasks.

Figure 3.1 illustrates the devices and supported by a spool :

for input :
- punch card
- floppy disk
- magnetic tape
- terminal

for output :
- printer
- punch card
- floppy disk
- magnetic tape
- terminal

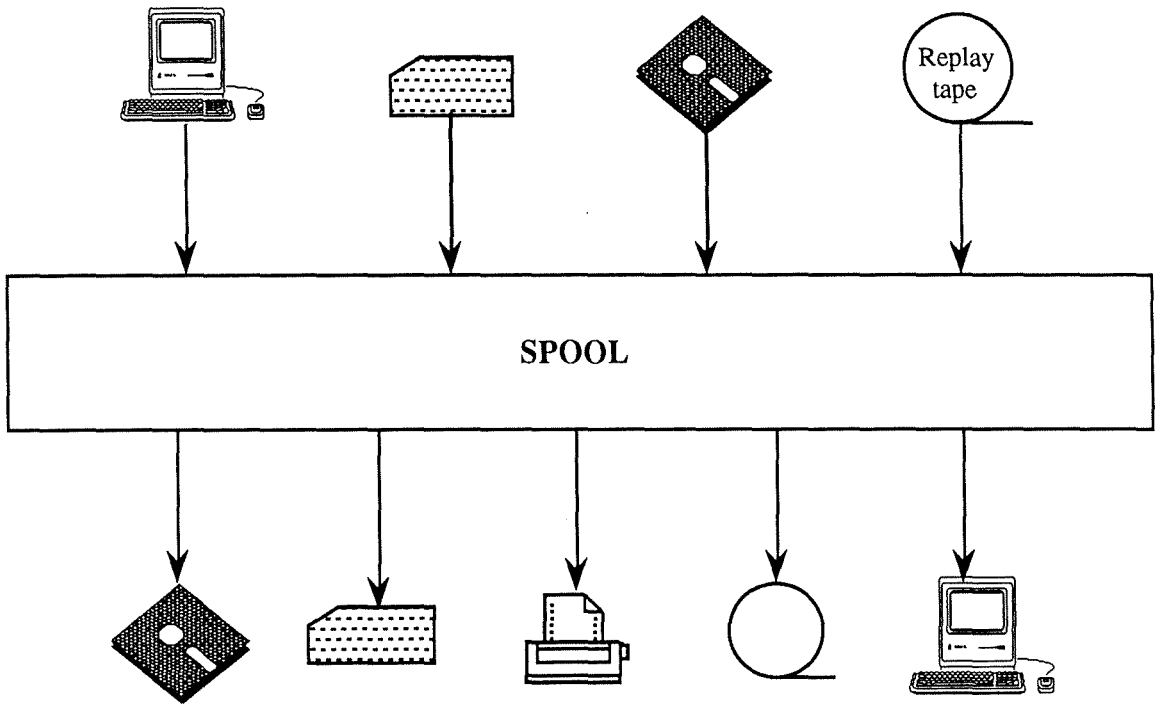

Figure 3.1 : spool supported devices

Figure 3.2 illustrates the functions of a spool[2].

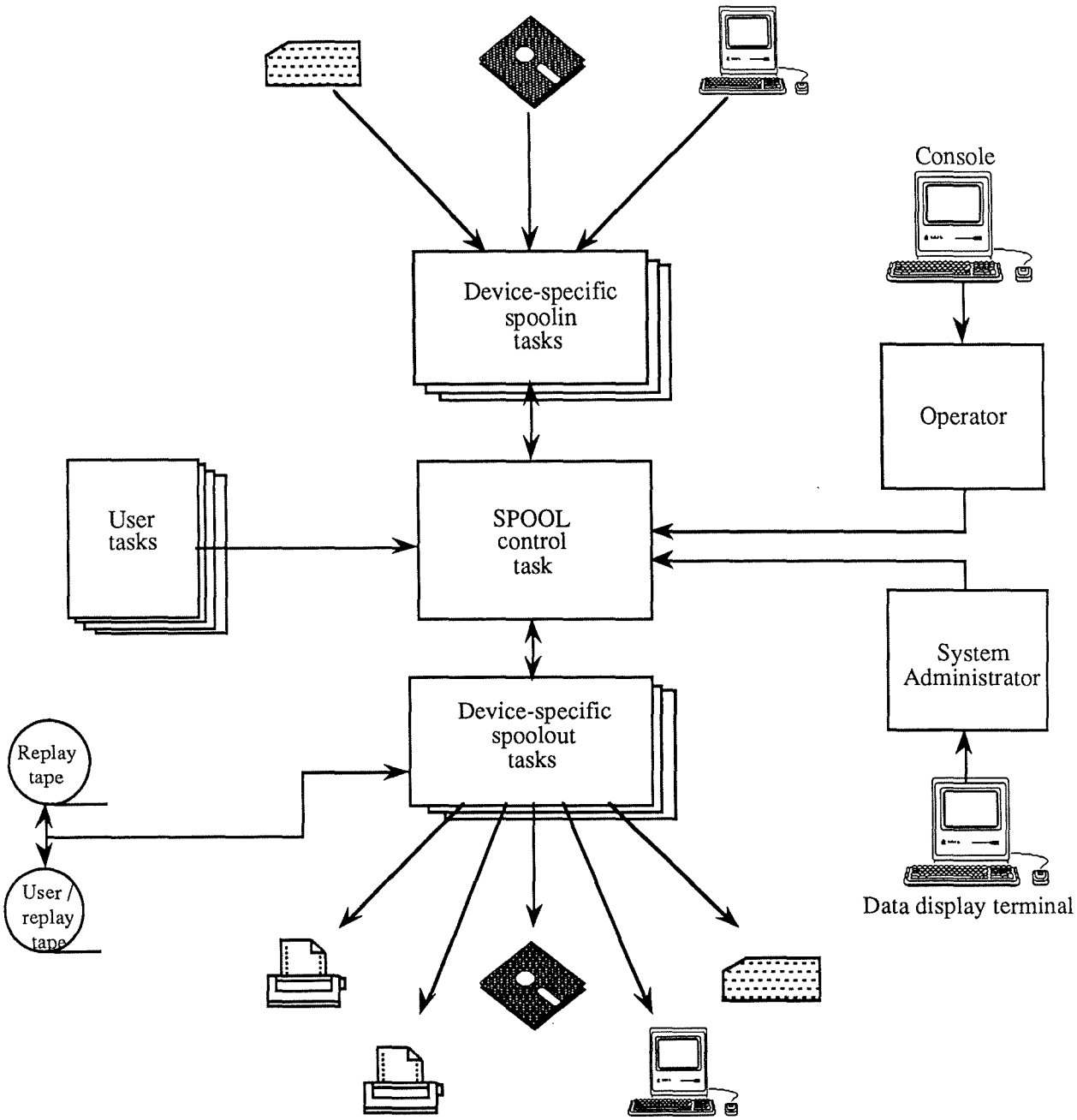

Figure 3.2 : spool functions

[2] In the following pages, we will concentrate only on the printing function.

CHAPTER 3 : IDENTIFICATION

3.1 IDENTIFICATION OF THE PROBLEM

3.1.1 Purpose and characteristics of a spool job

The main purpose of a spool job is to print a file or part of a file on a printer. Its main characteristics are the identification of the file or part of file to be printed, but several other characteristics are important for the caller.

The caller can specify on which paper, with which character sets and which form-overlays the job must be executed. Further on, he can specify that the file to be printed contains control characters which are specific to a particular hardware. He can assign a name to his job, so as to recognize it later on. He can also assign a priority to the job.

At an other point of view, the user can specify where the print has to occur. Basically, we distinguish the local printers, which are accessed by a channel, and the remote ones, which are connected to the network. The user can specify that the print is to be done on any local printer or on a given remote one. It is also possible to assign a "destination" name to one or several devices. The user has also the possibility to specify that the print is to be sent to this destination, i.e. to be processed on any device with this destination name representing a pool of printers.

The characteristics described above are explicitly chosen by the user. The spool jobs also have characteristics which are not explicitly specified. They are : the size of the file to be printed, the identification of the user, the account number under which he works, a spoolout-class number associated to this account number.

All these informations concerning a spool job are stored in a "Spool Control Block" (SCB) which is kept in a special file. This SCB is in fact an order-form for the spool job. This file is used by the scheduler to attribute the jobs to the printers.

3.1.2 Purpose of the scheduling process

The purpose of the scheduling is to treat all spool jobs under optimal conditions. However, the word "optimal" has not necessarily the same meaning for everybody.

For the user, the criterion is to minimize the waiting time, at the condition that all his requirements are satisfied.

For the system administrator, the criterion is the best use of the computer centre, i.e. to satisfy the user's needs at

the minimal costs, or rather, to earn the maximum for the minimal expense. Let us remark that he can charge a job more when it is processed in a shorter delay, particularly if he can guarantee this delay in advance.

For the operator, the criterion would be rather to minimize his efforts. This particularly concerns the changing of paper or form-overlays. Once a given paper is loaded on a printer, the best for him would be that the complete box is used before loading another one. Further on, he prefers that the same type of paper is used for a long time (i.e. several boxes), so that he can bring several boxes together.

These requirements are not always contradictory. For example, to change the paper frequently is not pleasant for the operator, but also not for the system administrator because during the change over, the printer remains unused. On the other hand, the best use of the computer centre necessarily has a positive effect on the waiting time for the user.

But in any case, even if a unique criterion is found, it is not possible to find the mathematically optimal solution, this for several reasons.

The first reason is the fact that the load is not known in advance. At any time, there is a certain amount of jobs to be done for which an optimal program of scheduling could be found. But this program becomes obsolete as soon as a new job is entered into the queue. The problem would be different if the jobs to be done were registered during a session, and executed only during the following one.

Another reason is that the load to execute a job is only approximately known. The size of the file is certainly known, but not the number of records, and still less the number of lines. There is an estimation based on the filesize : it is supposed that there are 17 lines in each 2048 bytes block, and this factor can be changed by the system administrator. However, it is easy to understand that this estimation is not accurate enough for an optimisation. It would, of course, be possible to scan the file during the PRINT command validation, and simulate the processing in order to know to exact number of lines, but this would be too expensive. An optimisation should not be more expensive than the process to be optimized itself.

A further problem is the reactions of the operator. When the spool requests the loading of a paper, it occurs that a box of paper is available close to the device, but it is not always the case; it is even possible that there is no paper of this type at all. If we also take into account the operator's skill, availability and motivation, the delay to load paper ranges from less than one minute to 15 minutes, after which the request can also be rejected. let us also remark that the spool is not aware of the quantity of paper which is still available on the device, before a new loading. It can occur that the paper remaining on the device is not sufficient for a job, in which case, it would possibly be better to directly request another paper right away.

This is not taken into account because the spool does not know the size of the loaded paper.

Let us mention a last point : the spool does not take into account the fact that a file is on tape or not. But opening a file on tape also means an operator reaction, which is also unknown. During the time spent to load the tape, the printer also remains unused.

3.1.3 The existing algorithm[3]

3.1.3.1 Physical and logical fit

Due to its characteristics, a spool job is not necessarily executable on any device type. It can use features (rotation, characters sets, overlays, ...) which are only available on certain device types. It is also possible that the form it uses is defined only for certain device types. Each SCB thus contains the indication of the possible device types, which is of course considered by the scheduling algorithm.

In addition, the system administrator has the possibility to specify "logical" criterions for the scheduling. When starting a device, he has operands to restrict it to jobs using certain values of the characteristics. For example, he can restrict it to certain users, or account numbers, or forms, and so on, or to a given range of priorities. A job which satisfies all the logical criterions for a given device has a so-called *logical fit*. If in addition, it can be processed on a device of this type, it also has a *physical fit*. But this does not mean that it will necessarily be processed on it.

3.1.3.2 Selection of a device for a job

When a job has a physical and a logical fit on several devices, it is still possible that it has a *better fit* on a certain device that on the other ones. For example, a device which accepts only one user-id becomes a kind of exclusiveness of this user-id, and should thus have a better fit than devices accepting all user-ids. The problem is however not so simple. Let us suppose that a device is reserved to a given user-id but accepts jobs of all priorities, and another one is reserved to a given range of priorities but accepts all user-ids. A job from this user-id and a corresponding priority has a logical fit on both. But for the system administrator, the criterions have not necessarily the same importance. For example, he could send all jobs with a lower priority onto a tape, in order to replay them during the night. He can thus consider that the priority is to be considered first. In older spool versions, five criterions were considered for the better fit : the priority which was only allowed for tapes, the form, the user-id, the class and the dia. A device with a priority had a better fit than any other one, while a device with one or several of the four other criterions had a better fit than devices without criterion.

---

[3] This algorithm is the one developped by Siemens Software S.A.

For each active device, it is thus possible to compute a weight which is the sum of all specified criterions. When a job has a physical fit and a logical fit on several devices, it is scheduled on the one with the greatest weight. This device is said to have the best fit for the job.

3.1.3.3 Optimisation criterion for the scheduling

In addition to the rules expressed above, the scheduling must keep each device active, as long as one or several jobs that have a best fit on it are present. With this set of constraints, the goal is to minimize the change-over cost when starting a new job on a device.

In older spool versions, two criterions were considered to compute the cost of the change over : the paper change and the form-overlay change (the dia). It was considered that the change of paper was more expensive than the dia change, and that all other criterions were free. In further versions, the number of character sets to be loaded, and the size of the fob (the dia on laser printers) to be loaded were also considered, but with a lower cost because these loadings cost a waiting time, but no operator action.

Since the spool version 2.4a, it is possible to choose any criterion in any order, to compute the change over cost. For example, it can be interesting to take the user-id into account, because sequential listings for the same user do not need to be separated by the operator.

The scheduling thus tries to find a job with the best fit, which has all the chosen characteristics identical to the ones of the preceding job. If none is found, it tries to find a job with all identical characteristics, except the last one, then except the two last ones and so on.

If several jobs exist which identically satisfy these conditions, there is still a set of characteristics which determines the first one : the priority and the age of the job. The system administrator can choose in which order these characteristics are to be considered.

3.1.3.4 Discussion this algorithm

This algorithm ignores several aspects of the problem :

- It ignores the size of the jobs; if one of the waiting jobs is longer than the sum of all the others, it would be interesting to start it at once on a device, and process all the other jobs on the other devices. But the scheduling completely ignores this aspect. Eventually, it will schedule the longer job after all the other ones.

- The scheduling ignores the possibility of paper-end, despite the fact that this is also a paper change, possibly as expensive as when another paper is required.

- The scheduling ignores the fact that opening a tape file needs to find a free tape device, and requires an operator reaction.

In summary, one could say that some simplifications have been arbitrarily introduced in the problem in order to make it determinist.

However, these simplifications are somehow crude. The provision of paper is taken as the most expensive change-over when it is decided by the scheduling itself, and completely ignored when it occurs at paper-end. The validity of this processing, though, depends on the computer centre itself. However, if the paper remains the most expensive component of the change-over cost, a job using a non-standard paper will be delayed until no other job remains in the waiting queue at all. If the printing capacity is computed to have a high use rate, such an event could possibly never occur ! Should it be computed to have a lower rate, but a fast response time, the optimisation is likely to completely fail : in most cases, when a job is terminated, there is no choice between waiting jobs.

Let us also mention that it is not possible to guarantee that a job will be processed in a given delay.

3.1.4 Possibility of other algorithms

Do other algorithms exist ? Most certainly. Each customer has probably his own needs, which could be covered by a corresponding algorithm.

The one provided by the spool is to be considered as a general one, but without real flexibility. It offers multiple commands, with numerous operands which have an impact on the processing but a rather indirect one. The selection criterions for a device have an effect on this device, but also on the other ones, due to the best fit. This never forces a job to be processed on a given device.

The modification of the device weights, the change-over costs, and the priority also influence the scheduling, but the effects can be very different, depending on e.g. the load. For example, the modification of the change-over costs has no effect at all, as soon as there is no waiting job for the device.

This does not mean that it is better to replace this scheduling by another one. It would be necessary to write quantities of scheduling algorithms among which each customer would have to choose. In summary, the system administrator needs a very good feeling of the system, if he wants to conduct the spool scheduling.

3.1.5 Mechanisms which could be improved

In the present status, the management of the spool devices is completely manual. The spool administrator gives arbitrary selection operands when he starts the devices. It occurs that jobs can not be scheduled on any one, because they have no fit on any one.

An interesting feature would thus be an analysis of the jobs in queue, and the choice of the selection operands in order to process these jobs under optimal conditions. In this purpose, the program interface should be able to give information like :

- the lists of forms, classes, user-ids, ... of the remaining jobs eventually with the number of jobs;

- the lists of jobs which are not schedulable.

In any case, it is supposed that a scheduling program is active, which keeps information about the waiting and active jobs, and starts the devices or modifies their characteristics. It can be informed of each PRINT command, or it can get informations on statistics. It can be informed of each started job on a device, and each job end, and it can get informations about each active device.

Moreover, the program can get other kinds of informations, e.g. the quantity of loaded paper, better and dynamic estimations of paper changes, and so on. It seems that such a scheduler could constitute an interesting and performing expert system.

3.2 PARTICIPANT IDENTIFICATION

After identifying the problem, it is necessary to identify the participants to the development process of the expert system. We distinguish five classes of participants : the end-users, the operators, the system administrators, the experts and the knowledge engineers.

the end-users : the end-users are obviously key-participants to the development process because they are part of the finalities of the expert system. They are the ones who will use most of the functions of the system. The end-users should therefore be part of the development process, especially during the identification and conceptualization stages.

the operators : the operators are other key-participants to the development process because they are, like the end-users, part of the finalities of the project. In part 3.1, we have pointed out how the current spool scheduler badly lacks management functions dedicated to the operators (e.g. functions to know the type of paper that should be used on a particular printer, functions to know when a box of paper has to be changed, ...). The operators should therefore be an active part of the development process, especially during the identification and the conceptualization stages.

the system administrators : the system administrators are another kind of "user" of the spool scheduler. However, their functions are restricted to set several parameters that are necessary for a good processing of the spool scheduler (in other words, the system administrator operates for the tuning,

the parametring of the scheduler). To a certain extend, they should participate in the development process because they could contribute to a better tuning of the system.

the experts and the knowledge engineers : these two classes of participants are gathered into one because in our case, the expert and the knowledge engineer are one and the same person. As a matter of fact, in the spool scheduling domain, the experts are computer scientists who are working for many years in that domain and who have already developed several versions of conventional spool schedulers. In other words, no one knows the scheduling problems better than these computer scientists and, no doubt, no one else would be more capable of developing a spool scheduler expert system. Therefore, in our particular case, the expert is also the person who builds the scheduler. By learning the expert system techniques, this expert would become the knowledge engineer in charge of the development process (this is an enviable situation because, on one hand, this situation reduces the number of people involved in the building process, and on the other hand, this situation cancels all the knowledge acquisition problems described in section 1). This class of participants is of course the most important one because it will be responsible of the whole development process.

3.3 GOAL IDENTIFICATION

The initial goal of the project was double. First, the project had to study and to implement different scheduling algorithms using expert system techniques. We have already said in part 3.1 that many different scheduling algorithms exist and that some of them have been implemented by Siemens for a commercial use. These algorithms however have been implemented with conventional languages. So, the first goal of the project was to study the feasibility to implement these scheduling algorithms using some expert system techniques. According to the available time, one or several prototypes of spool scheduler expert system had to be built, each of these prototypes corresponding to a different scheduling algorithm. As the creation of an expert system prototype is a long-term task and as the number of resources was limited, this or these prototypes had to be simple but sufficiently complete to appreciate the feasibility of such an approach. This or these prototypes had to gather all the basic concepts of a conventional spool scheduler while implementing each time a different scheduling mechanism.

The second goal of the project was to study the potential of this new approach. The spool scheduler developed by Siemens is a performant product that is continuously improved. However, this product can not propose several interesting functions (e.g. display the list of non-schedulable jobs, display the list of the user-ids of the remaining jobs, make some dynamic estimations of the need in paper, ...) because it was developed with a conventional language. On the other hand, an approach that uses the expert system techniques can offer many of these functions. Being short of time and of resources, the

implementation of these functions was not possible. Therefore, the second goal of the project was to study the potential of this new approach.

CHAPTER 4 : CONCEPTUALIZATION

4.1 CONFIGURATIONS OF THE SYSTEMS

In order to get a better understanding of the processing of a spool expert system, we will illustrate the system configurations of the current version of the spool, the system configuration considered for the use of a spool expert system, and the system configuration used for the development of the prototype.

4.1.1 System configuration of the current spool system

Figure 3.3 shows the system configuration of the current version of the spool system developed by Siemens[4].

In this configuration, the spool system is integrated to the control system of a "central computer" (i.e. a mainframe or a minicomputer). This central computer is connected to :

- 0, one or several terminals
- 0, one or several terminal and/or computer networks
- 0, one or several printers
- 0, one or several printer and/or computer networks.

In this type of configuration, the jobs sent by the terminals or by other computers are processed within the central computer. The spool system receives the jobs, processes them (i.e. chooses the optimal printer for each incoming job) and sends the jobs to the selected printer.

[4] Network A and network B might be gathered in one.

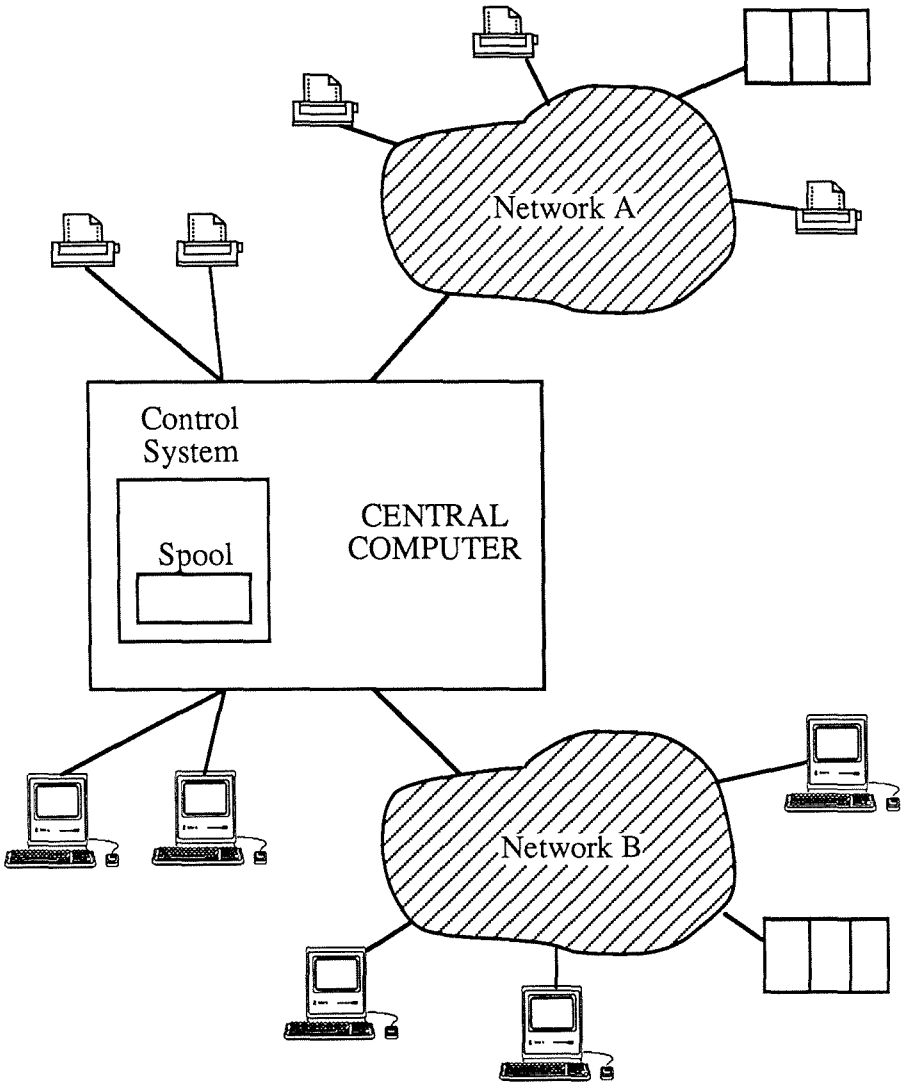

Figure 3.3 : system configuration of the current version of the spool

4.1.2 System configuration considered for the spool expert system

Figure 3.4 illustrates the system configuration envisaged for the spool expert system.

In this type of configuration, the spool system is not integrated to the central computer anymore. The spool expert system is installed on a micro-computer (e.g. a 80386-based computer) that is entirely dedicated to the spool scheduling activity. Thus, the central computer does not process any jobs; they are directly sent to the micro-computer. This micro-computer receives all the jobs, processes them and sends them to the printers.

The configuration shown in Figure 3.3 might be used for the spool expert system. But the Prolog language chosen to implement the system is very time- and space-consuming. In that case, the global performances of the system would be very low.

Therefore, the configuration shown in Figure 3.4 has been preferred to the previous one because all the power of the PC computer (i.e. CPU time and memory space) is entirely dedicated to the expert system.

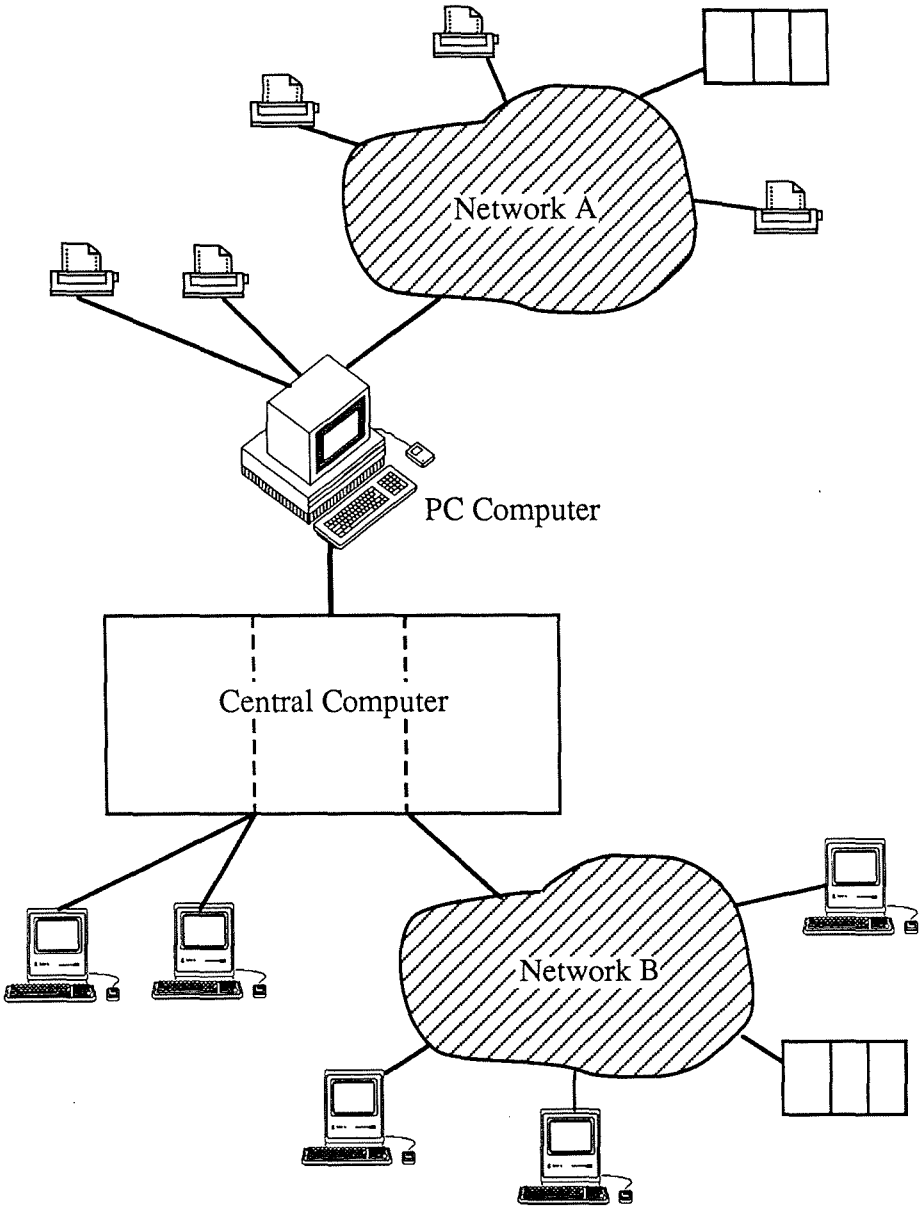

Figure 3.4 : system configuration for the spool expert system

4.1.3 System configuration used for the prototype

The system configuration used for the development of the prototype amounts to a simple 80386-PC computer (i.e. the type of computer that was chosen for a real implementation of the expert system). The main reason for such a bare configuration is, of course, the limitation of available resources. It is not conceivable to allocate one or several printers, as well as, one or several terminals exclusively for the development of a

prototype (even if this prototype is developed within the research centre of a big computer manufacturer !).

As there was only one computer without any real connection to any printer, we have developed a *simulation* prototype. We are talking about "simulation" because the prototype does not start any "real" printing at any time. The prime goal of the project was to study and to implement different kinds of scheduling. Therefore, the prototype confines itself to display on which device the job *could* be printed (i.e. the prototype displays the result of the execution of the scheduling algorithm). The reader might find this simulation approach relatively restrictive in regard to the initial goals. That is why, during all the development process, we took care that this simulation prototype could easily be transformed in a "real" printing prototype. As a matter of fact, only a couple of lines have to be changed to transform the prototype. After these changes, the prototype will not only display the result of the scheduling, but will also start a "real" printing of a file.

We also would like to point out that, during the construction of the prototype, we did not adopt a particular strategy concerning the use of the system. In other words, we did not develop a prototype that was exclusively reserved either to the end-users nor to the operators nor to the system administrators. On the contrary, we have adopted a pluralist strategy in order to determine if the prototype could satisfy all the participants. Therefore, the command language[5] gathers all the commands dedicated to the end-users, to the operators and to the system administrators without any distinction of classes. This approach is justified by the fact that it is easier to develop, to test and to evaluate a prototype without much usage restrictions. However, if this approach appears to be too soft, it is always possible to modify the prototype. As a matter of fact, only a couple of lines have to be changed to restrict the use of the system to a particular class of participants (e.g. we could restrict the command language to a certain kind of user-id or to a certain level of user priority). In that case, we would adopt a particular strategy.

4.2 BASIC CONCEPTS USED WITHIN THE CONTEXT OF THE PROTOTYPE

The development of an expert system is a long-term task which requires numerous resources. Being short of time, our prototype of a spool scheduler expert system only had to gather the basic concepts that are necessary for the correct execution of a spool. The important concepts used for our prototype are : the forms, the fonts, the devices, and the jobs.

[5] The command language is described in paragraph 4.4

4.2.1 The forms

By *form*, we mean the type of paper that can be used on a particular printer. Within the context of the development of our prototype, a form can be characterized by :

- a name = a value enabling to identify each form,
- a feeding type = the type of mechanism for dragging the paper,
- the paper size = the size of paper used on the printer,
- a job start sheet = the type of front-page to print,
- a job end sheet = the type of sheet ending the spoolout of a job,
- a job separator sheet = the type of separation between each page,
- the fob = the type of dia for laser printers.

A form is thus characterized by a set of values explicitly chosen by the operator and/or by the system administrator.

4.2.2 The fonts

By *font*, we mean the character set that can be used to print the job. Within the context of the development of our prototype, a font can be characterized by :

- a name = a value enabling to identify each font,
- a printing quality = a value representing the printing quality of the characters,
- a type = the type of characters (e.g. Courier, Times, Roman, ...),
- a style = the style of the characters (e.g. italic, bold, underlined, ...),
- a character size = the size of the character.

4.2.3 The devices

By *device*, we mean any printer likely to print a job (these devices can be local or remote printers). Within the context of our prototype, a device is characterized by :

- a name = a value enabling to identify each device,
- a printer type = the type and/or the brand of the device,
- an address = a destination name representing a pool of printers,
- a user priority = the lowest user priority allowed to send a job to the device,
- a user range = a restricting set of users allowed to use the device,
- a class range = a restricting set of spoolout classes associated to the account number of the user,
- a dia = the form-overlay with which the job must be executed,
- a form range = a restricting set of forms that can be used on the device,
- an active form = the current form installed on the device,
- a font range = a restricting set of fonts that can be used on the device,
- an active font = the current font,

- a rotation value = the orientation of the characters on the paper,
- a buffer size = the greatest allowed size of a job devoted to be printed on the device (it is a restricting value on the size of the job and not on the buffer size of the printer),
- a device state = the current state of the device.

A device is thus characterized by a set of values explicitly chosen by the operator and/or by the system administrator.

4.2.4 The jobs

By *job*, we mean any file or part of a file likely to be printed on a particular device. Within the context of our prototype, the characteristics of a job are :

- a job identifier = a value enabling to identify each job,
- a job state = the current state of the job,
- a device name = the device name on which the job should be printed,
- a first page number = a value representing the page number of the first page to be printed (i.e. the page number marking the beginning of the printing),
- a last page number = a value representing the page number of the last page to be printed (i.e. the page number indicating the end of the printing),
- a form = the form to be used,
- a font = the font to be used,
- a dia = the form-overlay with which the job must be executed,
- a rotation value = the orientation of the characters on the paper,
- a job priority = a priority assigned to the job,
- a number of copies,
- a remove flag = a value indicating if the file must be erased after printing or not,
- the file size = the size of the file to be printed,
- the user priority = the priority assigned to the user,
- the user identifier = the value identifying the user,
- the job class = the spoolout class associated to the account number of the user.

Among this set of characteristics, the first two are determined automatically by the spool scheduler, the following ten are explicitly chosen by the user, and the four last characteristics implicitly depend on the user.

4.3 TASKS TO CARRY OUT BY THE PROTOTYPE

We have already said that the main goal of the project was to implement a prototype that simulates the scheduling of a job. In this part, we will dissect this goal and give a more detailed analyse of the different tasks the program should carry out. We identified eight different types of tasks.

4.3.1 Device management tasks

At any time, the operator and/or system administrator should be able to :

- add a new device to the knowledge base[6],
- modify the characteristics of an existing device (e.g. modify the active form or the active font of a printer),
- delete an existing device (e.g. an old printer is replaced by a new one).

4.3.2 Form management tasks

At any moment, the operator and/or system administrator should be able to :

- add a new form to the knowledge base (e.g. a new type of paper has just been released),
- modify the characteristics of an existing form (e.g. no more front-page for draft listing),
- delete an existing form (e.g. an old type of paper is no more available).

4.3.3 Font management tasks

At any instant, the operator of the system administrator should be able to :

- add a new font to the knowledge base,
- modify an existing font (e.g. the printing quality of a font has been improved),
- delete an existing font (e.g. an old printer using a very particular font has been removed; since the font was used only on that printer, it should be deleted from the knowledge base).

4.3.4 Tasks to control the printing of jobs

At any time, the user should be allowed to :

- send a new job to the spool,
- modify the characteristics of a job (e.g. if a job is rejected because of form mismatch, the user should be able to send the job to another device or to modify the form of the job),
- delete a job that has not already been executed (e.g. if the user has send the wrong file to the spool, the user should be able to recall the job).

4.3.5 Information tasks

At any moment, the user, the operator or the system administrator should be able to gain complete informations on :

---

[6] This corresponds to the connection of a new printer to the pool of printers.

- the set of devices available in the knowledge base,
- the set of forms available in the knowledge base,
- the set of fonts available in the knowledge base,
- the set of jobs in the waiting queues.

4.3.6 Interrogation tasks

At any instant, the user, the operator or the system administrator should be able to :

- know the state of any device or the state of any job (e.g. to know if a job has been printed or if a device is currently available),
- modify the state of a device or the state of a job.

4.3.7 Explanation tasks

The user should be allowed to understand the approach that has been used by the prototype to simulate the scheduling of a job. Therefore, the prototype has to give explanations concerning the different steps it took to resolve the scheduling. In other words, the prototype has to explain how and why it reached a certain solution[7].

4.3.8 Help tasks

The expert system prototype should give help to the user concerning :

- the different commands available,
- the data types allowed during the insertion or the modification of a device, of a form, of a font or of a job.

[7] Due to the lack of time, this function was not implemented. But a similar result can be obtained with the TRACE or the DEBUG command of the Prolog language.

CHAPTER 5 : FORMALIZATION

5.1 KNOWLEDGE REPRESENTATION TOOL

The language used to formalize the knowledge and to implement the spool prototype is the Prolog language version 1.1 developed by Siemens. In order to get a better understanding of the choices made to formalize and to implement the system, this paragraph reviews some basic mechanisms of Prolog.

Prolog is a programming language for symbolic, non-numeric computation. It is specially well suited for solving problems that involve objects and relations between objects. The fact that Tom is a parent of Bob can be written in Prolog as :

parent(tom,bob).

Here, *parent* was chosen as the name of a relation; *tom* and *bob* are its arguments. When this clause has been communicated to the Prolog system, Prolog can be asked a few questions about the *parent* relation. For example, is Tom a parent of Bob ?

?- parent(tom,bob).

Having found this to be an asserted fact in the program, Prolog will answer yes. More interesting questions can also be asked. For example, who is Bob's parent ?

?- parent(X,bob).

The system will answer *X=tom*.

Our program can be asked an even broader question : who is a parent of whom ?

?- parent(X,Y).

Prolog will display the solutions one at a time until all the solutions have been found. Here, the answer is :

X=tom, Y=bob

Our example has helped to illustrate some important points:

- it is easy in Prolog to define a relation, such as the *parent* relation.
- the user can easily query the Prolog system about relations.
- a Prolog program consists of *clauses*.
- the arguments of relations can (among other things) be : concrete objects, or constants (such as *tom*), or general objects (such as X). Objects of the first kind are called *atoms*. Objects of the second kind are called *variables*.

- questions to the system consist of one or more *goals*. The word "*goals*" is used because Prolog accepts questions as goals that are to be satisfied.

As an extension to the program, let us introduce the *offspring* relation as the inverse of the *parent* relation. The corresponding Prolog clause is :

```
offspring(Y,X) :- parent(X,Y).
```

This clause is also called a *rule*. There is an important difference between facts and rules. A fact is something that is always, unconditionally true. On the other hand, rules specify things that may be true if some condition is satisfied. Therefore, rules have a condition part (the *body*) - i.e. the right-hand side of the rule - and a conclusion part (the *head*) - i.e. the left-hand side of the rule. Facts are clauses that have the empty body. Questions only have the body. Rules have the head and the (non-empty) body.

A question to Prolog is always a sequence of one or more goals. To answer a question, Prolog tries to *satisfy* all the goals. To satisfy a goal means to demonstrate that the goal is true, assuming that the relations in the program are true. In other words, to satisfy a goal means to demonstrate that the goal *logically follows* from the facts and rules in the program. If the question contains variables, Prolog also has to find what are the particular objects (in place of variables) for which the goals are satisfied. The particular *instantiation* of variables to these objects is displayed to the user. If Prolog can not demonstrate for some instantiation of variables that the goals logically follows from the program, then Prolog's answer to the question will be "*no*".

For example, given the question

```
?- offspring(bob,tom).
```

Prolog will try to satisfy this goal. In order to do so, it will try to find a clause in the program from which the above goal could immediately follow. Obviously, the only clause relevant to this end is the rule about the *offspring* relation because the head of this rule *match*[8] the goal.

```
offspring(X,Y) :- parent(X,Y).
```

Since the goal is *offspring(bob,tom)*, the variables in the rule must be instanciated as follows : *X=tom, Y=bob*. The original goal *offspring(bob,tom)* is then replaced by a new goal:

---

[8] Given two terms, we say taht they *match* if :
(1) they are identical, or
(2) the variables in both terms can be instancied to objects in such a way that, after the substitution of variables by these objects, the terms become identical.

parent(tom,bob).

This goal is immediately satisfied because it appears in the program as a fact[9].

In the previous examples, it has always been possible to understand the result of the program without exactly knowing *how* the system actually found the results. It therefore makes sense to distinguish between two levels of meaning of Prolog programs: the *declarative meaning* and the *procedural meaning*.

The declarative meaning is concerned only with the relations defined by the program. The declarative meaning thus determines *what* will be the output of the program. On the other hand, the procedural meaning also determines *how* this output is obtained - i.e. how are the relations actually evaluated by the Prolog system.

The ability of Prolog to work out many procedural details on its own is considered to be one of its specific advantages. It encourages the programmer to consider the declarative meaning of programs, relatively independently of their procedural meaning. Since the results of the program are, in principle, determined by its declarative meaning, this should be (in principle) sufficient for writing programs[10].

5.2 KNOWLEDGE REPRESENTATION

In chapter 4, we have determined the important concepts used in the prototype. In this paragraph, we will concentrate on the formalization of these concepts in Prolog.

As each concept can be identified by a set of characteristics, these concepts will be represented by a list of attributes (each attribute corresponding to a characteristic). Within the context of our work, some attributes accept only two or three different types of value. This might be quite restrictive comparing to the large amount of values allowed by some attributes. However, for the development of a prototype, the restriction in the number of different values does not change the validity of a feasibility study. Therefore, we have opted for simplicity.

---

9 If there is no clause in the program whose head matches the goal *parent(tom,bob)*, then this goal fails. In that case, Prolog backtracks to the original goal to try an alternative way to derive to top goal *offspring(bob,tom)*.

10 Unfortunately, however, the declarative approach is not always sufficient, especially in large programs where the procedural aspects can not be completely ignored by the programmer for practical reasons of executional efficiency.

The formalization of the different concepts and the types of value allowed for the corresponding attributes are the following[11]:

5.2.1 The forms

A form will be formalized by the following list of attributes:

[name, feeding type, paper size, job-start-sheet, job-end-sheet, job-separator-sheet, fob]

The values allowed for the different attributes are :

- name :
  any string of characters starting with a lowercase letter

- feeding type :
  <single> for a sheet-by-sheet feeding
  <listing> for a continuous feeding

- paper size :
  <a3> for a paper corresponding to the international standard A3
  <a4> for a paper corresponding to the international standard A4
  <a5> for a paper corresponding to the international standard A5

- job-start-sheet :
  <brief> for a short front-page (a couple lines)
  <long> for a long front-page (a whole page)
  <none> for a printing without front-page

- job-end-sheet :
  <brief> for a short end-sheet (a couple lines)
  <long> for a long end-sheet (a whole page)
  <none> for a printing without end-sheet

- job-separator-sheet :
  <brief> for a short in-between sheet (1 or 2 lines)
  <long> for a long in-between sheet (5 or 10 lines)
  <none> for a printing without any in-between lines

- fob :
  either an element from the set {f1,f2,f3} to simulate a particular fob,
  or nothing if there should not be any fob.

5.2.2 The fonts

A font will be formalized by the following list of attributes:

[name, printing quality, style, type, size]

[11] Words between "<" and ">" should be written as such.

The values allowed for the different attributes are :

- name :
  any string of characters starting with a lowercase letter

- printing quality :
  <draft> to simulate a fast and draft printing quality
  <normal> to simulate a normal printing quality
  <high> to simulate a very high printing quality

- style :
  a non repetitive list of elements belonging to the set
  {italic,underlined,bold}

- type :
  <courier> for a character type "Courier"
  <times> for a character type "Times"

- size :
  <10> for 10 points size
  <12> for 12 points size
  <14> for 14 points size

5.2.3 The devices

A device will be formalized by the following list of attributes:

[name, printer type, address, lowest priority, user-id range, class range, dia, form range, active form, font range, active font, buffer size, device state]

The values allowed for the different attributes are :

- name :
  any string of characters starting with a lowercase letter

- printer type :
  any string of characters starting with a lowercase letter

- address :
  any string of characters starting with a lowercase letter

- lowest priority :
  any integer between 0 and 255 [12]

[12] 0 is the highest user priority and 255 is the lowest user priority.

- user-id range :
  either a list of user-ids to bound the use of the device to certain users[13],
  or nothing if the device can be used by everyone

- class range :
  either a non repetitive list of elements belonging to the set {cl1,cl2,cl3,cl4} to bound the use of the device to certain spoolout classes,
  or nothing if the device can accept any spoolout class

- dia :
  either an element of the set {dia1,dia2,dia3,dia4} if the device will print jobs with one of these dias,
  or nothing if the device does not use any particular dia

- form range :
  either a non repetitive list of existing form names to bound the use of the device to these forms,
  or nothing if the device can accept any type of form

- active form :
  the name of the current active form (i.e. a string of characters starting with a lowercase letter)

- font range :
  either a non repetitive list of existing font names to bound the use of the device to these fonts,
  or nothing if the device can accept any type of font

- active font :
  the name of the current active font (i.e. a string of characters starting with a lowercase letter)

- rotation :
  <portrait> for a printing with a vertical orientation
  <landscape> for a printing with an horizontal orientation

- buffer size :
  either an integer between 1 and 999.999 to bound the use of the device to jobs which size are smaller than this integer,
  or nothing if the device accepts jobs of any size

- device state :
  <active> if the device can be used
  <wait> if the device is momentarily out-of-order[14]

---

[13] A user-id is a string of characters starting with a lowercase letter.

[14] A more dynamic management of the devices (i.e. determining if the device state is "printing", "waiting for a job", "out of order", ...) couldn't be considered due to the lack of time.

5.2.4 The jobs

A job will be formalized by the following list of attributes:

[job-id, file name, device, page from, page to, form, font, dia, rotation, file size, user priority, user-id, job class, number of copies, remove, job priority, job state]

The values allowed for the different attributes are :

- job-id :
  an integer[15]

- file name :
  any string of characters starting with a lowercase letter

- device :
  either the name of an existing device (i.e. a string of characters starting with a lowercase letter) to print the job on a particular device,
  or nothing if the job can be printed on any device

- page from :
  either an integer to start the printing from a particular page number,
  or nothing to start the printing from the beginning of the file

- page to :
  either an integer to print the job until a particular page number,
  or nothing to print the job until the end of the file

- form :
  either the name of an existing form (i.e. a string of characters starting with a lowercase letter) to print the job with a particular form,
  or nothing to print the job with any form

- font :
  either the name of an existing font (i.e. a string of characters starting with a lowercase letter) to print the job with a particular font,
  or nothing to print the job with any font

- dia :
  either an element of the set {dia1,dia2,dia3,dia4} to print the job with a particular dia,
  or nothing to print the job without any dia

- rotation :
  <portrait> to print the job with a vertical orientation

[15] This integer is dynamically determined by the system and not by the user.

<landscape> to print the job with an horizontal orientation

- file size :
  an integer

- user priority :
  an integer between 0 and 255

- user-id :
  a string of characters starting with a lowercase letter

- job class :
  an element of the set {cl1,cl2,cl3,cl4}

- number of copies :
  either an integer greater than 1 to print several copies,
  or nothing to print one copy

- remove :
  <yes> to delete the file after the printing
  <no> either way

- job priority :
  either an integer between 0 and 30 to print the job with a particular priority (0 is the highest priority, 30 is the lowest priority),
  or nothing to print the job with the default priority (i.e. 15)

- job state :
  <active> to send the job to the spool scheduler
  <wait> to momentary set the job to a waiting state[16]

5.2.5 Restrictions

For a correct execution of the system, it is not allowed :

- to use uppercase letters as values for the attributes. Because the system is using Prolog, any uppercase letter or any string starting with an uppercase letter would be considered as an internal variable of the system. Any uppercase letter could therefore induce many problems during the system's execution.

- to use xxxxxxxx and xxxxxxx as file name, device name, form name or font name. These two strings are reserved for the execution of certain internal functions of the system.

[16] A more dynamic management of the jobs (i.e. determining if the job state is "printed", "waiting for a device", "printing", ...) couldn't be considered due to the lack of time.

5.3 FORMALIZATION OF THE USER INTERFACE

To be able to carry out the tasks described in paragraph 4.2, the user should be able to interact with the expert system. To enable this dialogue between the user and the spool scheduler, we have created a simple but complete interface. This interface consists of a command language, some dialogue screens and a set of help messages.

5.3.1 The command language

5.3.1.1 Commands intended to the operator

The commands intended to the operator (and to the system administrator) enable to carry out management tasks for the devices, the forms and the fonts. These commands are :

add dev *dev-name* :
    add a new device named *dev-name* to the working memory
modify dev *dev-name* :
    modify a device named *dev-name* to the working memory
delete dev *dev-name* :
    delete a device named *dev-name* to the working memory
add form *form-name* :
    add a new form named *form-name* to the working memory
modify form *form-name* :
    modify a form named *form-name* to the working memory
delete form *form-name* :
    delete a form named *form-name* to the working memory
add font *font-name* :
    add a new font named *font-name* to the working memory
modify font *font-name* :
    modify a font named *font-name* to the working memory
delete form *font-name* :
    delete a font named *font-name* to the working memory
set dev *dev-name* active :
    set a device named *dev-name* to the active state
set dev *dev-name* wait :
    set a device named *dev-name* to the wait state

5.3.1.2 Commands intended to the user

The commands intended to the user (and in a certain way to the operator) are :

show dev short :
    display the name, the type, the active form and the state of each device
show dev long :
    display all attributes of each device
show dev free :
    display all attributes of each free (unused) device
show dev form :
    display the name, the active form and the form range of each device

show dev font :
display the name, the active font and the font range of each device
show dev prio :
display the name, the lowest user priority and the user-id range of each device
show dev dia :
display the name, the dia and the rotation of each device
show form short :
display the name, the paper size, the fob, and the feeding type of each form
show form long :
display all attributes of each form
show font short :
display the name, the printing quality, the type and the style of each font
show font long :
display all attributes of each font
show job short :
display the job-id, the file name, the device and the state of every job
show job long :
display all attributes of every job
stat dev *dev-name* :
display the name, the state, the active form, the active font, the dia, the rotation, the active job and the number of waiting jobs of a device named *dev-name*
stat job *job-id* :
display the job-id, the file name, the user-id, the state, the affected device and the requested number of copies of a job identified by *job-id*
stat global system :
display the current status of the whole spool system[17]
print *file-name* :
simulate the printing of a file with default values
print *file-name* m :
simulate the printing of a file with specific attribute values chosen by the user
modify print :
modify the values of the last print command[18]
delete job *job-id* :
delete a job from the working memory
set job *job-id* active :
set the job in an *active* state
set job *job-id* wait :
set the job in a *wait* state
free *dev-name* :
simulate the liberation of a device named *dev-name* (i.e. the device has printed a file and is ready to print another one)

---

[17] This command works only if some printers are connected to the computer.

[18] This command is useful when a print request has just been refused because of a wrong filled attribute.

```
help :
     display the commands available
load :
     load the working memory from a file
save :
     save the working memory in a file
end :
     quit the spool scheduler prototype
```

5.3.2 Dialogue screens

Among the commands described above, some commands require many parameters to be executed. For example, a correct execution of the commands *add*, *modify* or *print* requires the user to enter a lot of different values.

To enable an easy way to enter these values, we have created some dialogue screens. This process has a double advantage :

- first, this process is easy to implement by the builder of the prototype. It is, in fact, easier and faster to create some dialogue screens than to write a full parser able to dissect any sophisticated command. This process being faster, it enables us to devote more time to other important tasks.

- secondly, this process is easier to tackle by the user. It is, in fact, easier for the user to follow the steps given by the system (dialogue screens, orientation messages, ...) than to memorize for each command the list of parameters, the right syntax for these parameters, the default values, ...

5.3.3 Helping messages

At any time (while displaying the result of the scheduling, while entering values, while writing down commands, ...) the user is helped with a large amount of helping messages. These short messages enable an easier and better use of the prototype. These messages indicate for instance :

- the types of values to enter
- the syntax errors and their correction
- the semantic errors
- the results of the scheduling and the tasks to carry out if a job has not been scheduled
- the reasons for the wrong execution of a command
- etc ...

5.4 ARCHITECTURE OF THE WORKING MEMORY

This working memory gathers all the important concepts identified in chapter 4 (i.e. the devices, the jobs, the forms and the fonts). These concepts are represented in the working

memory by lists of attributes. However, because of internal handling problems of these lists, some elements had to be added to the working memory.

On one hand, we had to frame these lists with an identifier. Because the working memory gathers lists of all types, the system must have a way to recognize which are the lists of devices, the lists of jobs, the lists of forms, and the lists of fonts. In other words, the system must have a way to identify each type of list. Therefore, each list has been framed with an identifier. The lists in the working memory look like :

for the devices :

dev([name, printer type, address, lowest priority, user-id range, class range, dia, form range, active form, font range, active font, rotation, buffer size, device state])

for the forms :

form([name, feeding type, paper size, job start sheet, job end sheet, job separator sheet, fob])

for the fonts :

font([name, printing quality, style, type, size])

for the jobs :

job([job-id, file name, device, page from, page to, form, font, dia, rotation, file size, user priority, user-id, job class, number of copies, remove, job priority, job state])

On the other hand, flag-lists had to be added for each concept. As a matter of fact, for some commands, the inference engine has to search through the working memory to gather all the lists belonging to the same concept (e.g. the set of devices for a *show dev long*). But, our inference engine searches through the working memory for a particular type of list from the first list of this type till a certain flag (and not till the last list of this type). This means that if the inference engine does not locate a certain flag, he is unable to give the set of lists of a common type. In a similar way, he is unable to search through the working memory to find a particular element. To solve this problem, flag-lists have been added for each concept. These flag-lists look like :

for the devices :

dev([xxxxxxx,1,1,1,1,1,1,1,1,1,1,1,1,1])

for the forms :

form([xxxxxxx,1,1,1,1,1,1])

for the fonts :

font([xxxxxxx,1,1,1,1])

for the jobs :

job([0,xxxxxxx,1,1,1,1,1,1,1,1,1,1,1,1,1,1])

The way these lists are sorted in the working memory does not matter (as long as the lists of a same concept are gathered together). An example of the architecture of the working memory is shown in Appendix.

5.5 ARCHITECTURE OF THE KNOWLEDGE BASE

The set of rules composing the knowledge base are sorted in the following order :

- rules for handling internal structures
- rules concerning the choice of the command to execute
- rules for adding devices, forms, or fonts to the working memory (the *add* command)
- rules to simulate the printing of a job (the *print* command)
- rules to modify devices, forms, fonts, or jobs in the working memory (the *modify* command)
- rules to display informations (the *show* command)
- rules to display device and job states (the *stat* command)
- rules to delete devices, forms, fonts, or jobs from the working memory (the *delete* command)
- rules to set a device or a job to a new state (the *set* command)
- rules to execute the scheduling of a job.

CHAPTER 6 : IMPLEMENTATION

In accordance with the prime goal of the project, three different versions of the prototype have been implemented, each one corresponding to a particular scheduling techniques. However, though being different, these three versions have common parts, namely :

- the user interface module,
- the knowledge base, except the scheduling rules (i.e. all the rules concerning the process of the *add*, *modify*, *show*, *stat*, *set*, *delete*, *load*, *save*, *print*, and *free* command).

Since these three versions are implemented in Prolog, the three prototypes use the same inference engine and the same architecture for the working memory.

Paragraphs 6.1 through 6.3 analyse in details the scheduling algorithms and the way they were implemented.

6.1 A PROTOTYPE OF A SPOOL SCHEDULER EXPERT SYSTEM, VERSION 1

In this first version of the prototype, the scheduling process starts when a user executes a *print* command. In this version, each device has a waiting queue. When the spool prototype receives a job, it starts the scheduling process of this job i.e. according to the characteristics of the devices and the characteristics of the job, the spool determines the "best", the "optimal" device to print this job. Once a device has been selected, the spool prototype puts the job in the waiting queue of that device. The printing of the job will take place when all the jobs previously placed in the waiting queue are printed[19].

This scheduling process is illustrated in Figure 3.5.

In this Figure 3.5, the different steps correspond to the following processings :

[19] Remember that in our case, the prototype just simulates the printing.

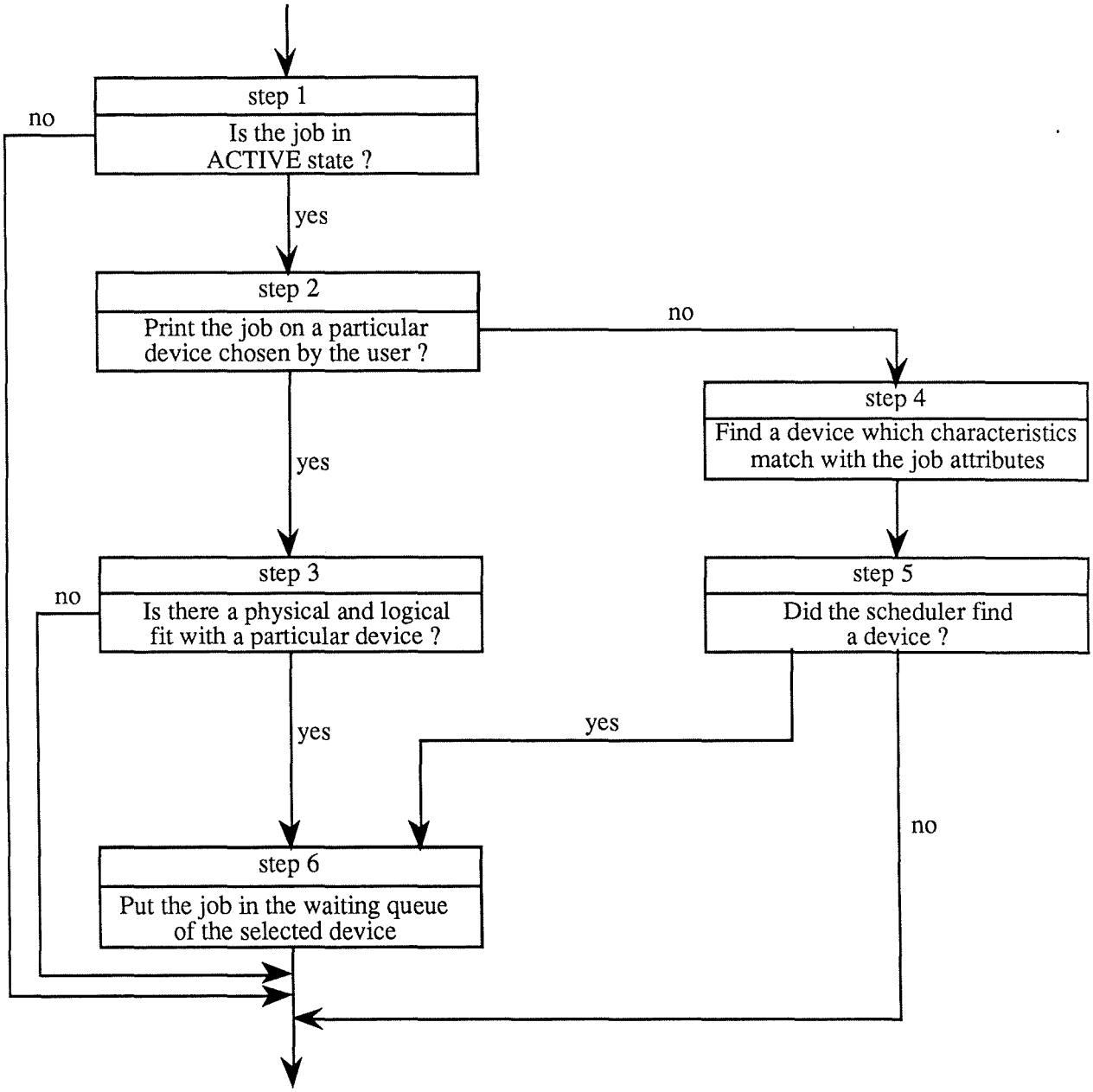

Figure 3.5 : scheduling process in version 1

step 1 : is the job in active state ?

If the *job state* attribute of the job is *active*, the scheduling process goes on. In the other case, the scheduling process is stopped and will restart with a *set job job-id active* command.

step 2 : print the job on a particular device ?

If the *device* attribute of the job is filled with a device name, the spool prototype will check if that particular device

can print the job. In the other case, the prototype will try to find any other device to print the job.

step 3 : is there a physical and logical fit with the particular device ?

At this stage, the prototype checks if the particular device can print the job, i.e. it successively checks if :

- the file size is smaller or equal to the device buffer size
- the job form belongs to the device form range
- the job font belongs to the device font range
- the job dia matches with the device dia
- the job rotation matches with the device rotation
- the job user priority is smaller or equal to the device user priority
- the job user-id belongs to the device user-id range
- the job class belongs to the device class range
- the job form matches with the device active form
- the job font matches with the device active font

If all these values match together (physical and logical fit) and are well-suited (the active form and the active font are the ones asked in the *print* command), then the job will be printed on that particular device. If, on the other hand, these values do not match together (this means that some characteristics of the job can not be accepted by the device), the scheduling process is stopped and a message is sent to the user. The user has then the opportunity to modify the attributes of his job with a *modify print* command.

step 4 : find any device whose characteristics match with the job attributes

If the user has not specified a particular device in his *print* command, the prototype tries to find a device with at least a logical and physical fit.

- first, the prototype searches the set of active devices having the same dia and rotation value as the ones found in the *print* command. This set of devices is called DEVLIST.

- if DEVLIST is empty, go to step 5

- if DEVLIST is not empty, the scheduler searches among the elements of DEVLIST for all the devices with a physical and logical fit, i.e. it successively checks for each device if:
  - the file size is smaller or equal to the device buffer size
  - the job form belongs to the device form range
  - the job font belongs to the device font range
  - the job user priority is smaller or equal to the device user priority

  - the job user-id belongs to the device user-id range
  - the job class belongs to the device class range
  - the job form matches with the device active form
  - the job font matches with the device active font

- the devices selected are shared out in two different lists :
  - the LOGLIST which gathers the devices with a physical and logical fit
  - the BESTLIST which gathers the "best" devices, i.e. the devices with a physical fit, with a logical fit, and which active form and active font are the ones requested in the *print* command.

step 5 : did the scheduler find a device ?

At this stage, the prototype checks the following situations:

- if DEVLIST is empty, the scheduling process is stopped, and a message is sent to the user to tell him that his job can not be printed on any device.

- if LOGLIST (and consequently BESTLIST) is empty, the scheduling process is stopped, and a message is sent to the user to ask him to change some attributes of his job.

- if BESTLIST is empty and LOGLIST is not empty, the job can be printed on at least one device but with another form and/or another font than the one requested. Therefore, a message is sent to the user to suggest him to change the attributes of his job or to change the form and/or the font of the device (i.e. this would be a "physical" change).

- if BESTLIST is not empty, the scheduler has to choose the optimal device among the elements of BESTLIST. This choice is based on the user-id range, the class range and the user priority. The optimal device is the one that offers the strongest restrictions on the user-id range and the class range, and that offers the lowest user priority, i.e.
  - if the use of a device is restricted to a small range of users, this device is preferred to the one that accepts all users,
  - if the use of a device is restricted to a small range of job classes, this device is preferred to the one that accepts all kinds of job classes,
  - a device is preferred to another one if its use is restricted to users with a higher priority level.

step 6 : put the job in the waiting queue of the selected device

If a device has been selected, the prototype sends a message to the user to tell him that his job has been placed in the waiting queue of that device.

6.2 A PROTOTYPE OF A SPOOL SCHEDULER EXPERT SYSTEM, VERSION 2

In the second version of the prototype, the scheduling process starts when a device is freed, i.e. when a device has finished to print a job and is ready to print a new job. In order to simulate this freeing, the user executes a *free dev-name* command.

In this version, each job sent to the spool scheduler is placed in the spool waiting queue. There, the jobs wait to be processed. And when a device is freed, the system starts the scheduling process of that device, i.e. according to the characteristics of the device and the characteristics of all the jobs in the spool waiting queue, the spool determines the "best", the "optimal" job to be printed on that device. Once a job has been selected, it is deleted from the waiting queue and sent to the device for printing.

The scheduling process of this second version is illustrated in Figure 3.6.

In this Figure 3.6, the different steps correspond to the following processings :

step 1 : the user sends a *print* command

step 2 : print the job on a particular device ?

If the *device* attribute of the job is filled with a device name, the spool prototype will check if that particular device can print the job. On the other hand, the scheduler places the job in the waiting queue.

step 3 : is there a physical and logical fit with the particular device ?

In this stage, the prototype checks if the particular device can print the job, i.e. it successively checks if :

- the file size is smaller or equal to the device buffer size
- the job form belongs to the device form range
- the job font belongs to the device font range
- the job dia matches with the device dia
- the job rotation matches with the device rotation
- the job user priority is smaller or equal to the device user priority
- the job user-id belongs to the device user-id range
- the job class belongs to the device class range

If all these values match together (physical and logical fit), then the job will be placed in the spool waiting queue. If, on the other hand, these values do not match together (this means that some characteristics of the job can not be accepted by the device), a message is sent to the user. The user has then

the opportunity to modify the attributes of his job with a *modify print* command.

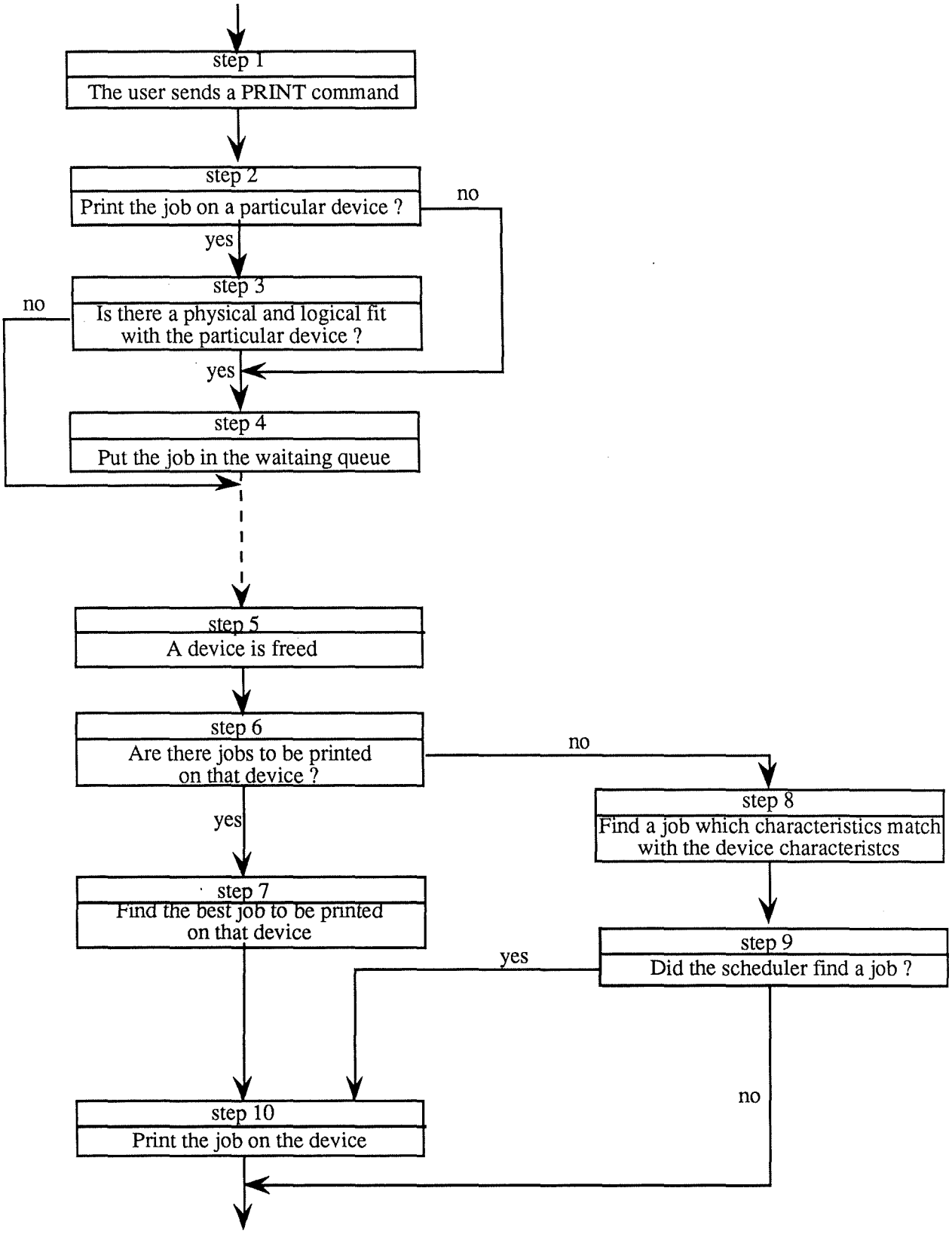

Figure 3.6 : scheduling process in version 2

step 4 : put the job in the spool waiting queue

step 5 : a device is freed

A user simulates the freeing of a device with a *free dev-name* command.

step 6 : are there jobs to be printed on that device ?

The scheduler searches in the spool waiting queue the set of active jobs to be printed on that device. This set of jobs is called LOGLIST (because the physical and logical fit of these jobs have already been checked in step 3).

If LOGLIST is not empty, go to step 7; otherwise, go to step 8.

step 7 : find the "best" job to be printed on that device

Among the elements of LOGLIST, the system searches the jobs with a better fit, i.e. the jobs of which both the form and font match with the active form and active font of the device. The set of jobs with a better fit is called BESTLIST.

- if BESTLIST is not empty, the system selects the job with the highest job priority (if there are several jobs with the same priority level, the system chooses the oldest job).

- if BESTLIST is empty, the jobs can be printed on the device but with another form and/or another font than the ones requested. Therefore, the system selects the job with the highest priority and sends a message to the correspondent user to suggest him to change the attributes of his job or to change the form and/or the font of the device (this would be a "physical" change).

step 8 : find any job whose characteristics match with the device characteristics

If the user has not specified a particular device in his *print* command, the system tries to find a job with at least a physical and logical fit.

- first, the scheduler searches the set of active jobs having the same dia and rotation value as the one of the freed device. This set of jobs is called JOBLIST.

- if JOBLIST is empty, go to step 9.

- if JOBLIST is not empty, the scheduler searches among the elements of JOBLIST for all the jobs with a physical and logical fit, i.e. it successively checks for each job if :
  - the file size is smaller or equal to the device buffer size
  - the job form belongs to the device form range
  - the job font belongs to the device font range

  - the job user priority is smaller or equal to the device user priority
  - the job user-id belongs to the device user-id range
  - the job class belongs to the device class range
  - the job form matches with the device active form
  - the job font matches with the device active font

- the jobs selected are shared out in two different lists :
  - the LOGLIST which gathers the devices with a physical and logical fit,
  - the BESTLIST which gathers the "best" jobs, i.e. the jobs with a physical fit, with a logical fit, and which form and font correspond to the active form and active font of the device.

step 9 : did the scheduler find a job ?

In this stage, the scheduler checks the following situations:

- if JOBLIST is empty, the scheduling process is stopped, and a message is sent to the user to tell him that the device can not print any job.

- if LOGLIST (and consequently BESTLIST) is empty, the scheduling process is stopped, and a message is sent to the user to tell him that the device can not print any job.

- if BESTLIST is empty and LOGLIST is not empty, the device can print at least one job but with another form and/or another font than the ones requested. Therefore, a message is sent to the user to suggest him to change the attributes of his job or to change the form and/or the font of the device (i.e. this would be a "physical" change).

- if BESTLIST is not empty, the scheduler has to choose the optimal job among the elements of BESTLIST. This choice is based on the job priority and the age of the job. The optimal job is the one with the highest job priority. If several jobs have the same priority level, the system chooses the oldest one.

step 10 : print the job on the device

The system sends a message to the user to tell him which job will be printed next on the device.

6.3 A PROTOTYPE OF A SPOOL SCHEDULER EXPERT SYSTEM, VERSION 3

This third version of the prototype is a mixed version of the two first ones. It corresponds to the second version (in the way that the scheduling process starts when a device is freed) to which a module has been added. This additional module starts

a pre-scheduling process (similar to the scheduling process of version 1) on the set of free devices.

As a matter of fact, a close analyse of the second version of the prototype reveals a lack of efficiency in the management of the free devices : when the scheduler does not allocate a job to a free device, this device remains unused as long as a user re-executes a *free dev-name* command. To solve this problem, a third version of the prototype has been implemented. In this third version, if a device is freed and if no job has been allocated to that device, then the device is placed in a list of free devices called FREELIST. In this way, when a user executes a *print* command, the system starts a pre-scheduling process on the elements of FREELIST, i.e. instead of placing the job in the spool waiting queue, the system will try to print the job on one of the currently unused devices. If a free device can print the job, then it is deleted from the FREELIST and the job is directly printed on that device. If on the other hand, any free device can not print the job, then the FREELIST remains the same and the job is placed in the spool waiting queue.

The pre-scheduling process corresponds to the scheduling process of version 1, except that the system considers only the few devices of the FREELIST instead of considering all the devices of the working memory. All other functions (scheduling of the spool waiting queue, ...) correspond to the ones implemented in version 2.

CHAPTER 7 : TESTING

The final stage in building an expert system is the testing stage. This involves evaluating the prototype system and the representation forms used to implement it.

In our case, the spool scheduler prototypes have been tested with a variety of examples to determine weaknesses in the knowledge base and inference structure. The different test problem examples have been organized so that they cover the subproblems, probe the boundaries of expected "hard" cases, and deal with the "classical" cases of a problem.

Several weaknesses have been found leading to a regular revision of the prototypes. Some concepts had to be reformulated. Some representations had to be redesigned. But most of the weaknesses were eliminated through small refinements of the system, i.e. recycling through the implementation and testing stages in order to tune or adjust the rules and their control structures until the expected behaviour is obtained.

The result of these regular revisions led to a higher level of performances. Although the general performances of the system have never been tested with any statistical tool, the response time of the system (i.e. the time between the user executes a command and the system displays a response message) has always been very short (0.2 or 0.3 second).

CONCLUSION

CONCLUSION

To develop a real spool scheduler expert system is a complex and long-term task that would go beyond the context of this dissertation. That is the reason why, right from the beginning, the orientation of the project was to develop an experimental prototype. This means that, even if the system is not dedicated to a commercial use, its development is nevertheless based on a realistic study of the existing.

In this order of ideas have been defined the two main goals of the project : on one hand, to tackle the spool scheduling problem with a new computer approach (i.e. the expert system approach), and, on the other hand, to define the possibilities offered by this new approach to build a commercial product.

To answer the first goal, we have applied the general methodology for building expert systems to our spool scheduling problem. We have started with an identification stage to study the numerous advantages and disadvantages of the traditional spool scheduler (i.e. the spool written with conventional languages), and to identify the different mechanisms (algorithmic or others) that are to be used for the building of a prototype. We have carried on with a conceptualization stage to define the system configuration, the basic concepts, their truth values and the relations between them. This second stage has also enabled us to define the tasks the prototype has to carry on. The formalization stage has then enabled us to choose a pertinent representation mode for the concepts and their relations. Finally, the implementation stage has enabled us to illustrate the modelisation of the scheduling process with a particular algorithm (and since there are many scheduling processes, we have implemented three algorithms, leading to three different versions of our prototype).

When one tries to evaluate these prototypes, one has to adopt an imprecise procedure. As a matter of fact, evaluation of expert systems, unlike knowledge acquisition or inferencing mechanisms, is a topic that has been minimally addressed. Expert system evaluation is often nebulous, since there are no universally accepted or unbiased formal specifications against which the system can be judged. We will then evaluate our prototypes through the criterions identified by [WOL-87], that is : the user acceptance, the performances of the system, the utility or value-added benefits, the liability risk and the feedback to the knowledge engineers.

From the user's perspective, the prototypes have a smooth and quite efficient interface. Of course, this interface is not perfect but it provides the user with a simple command language, many dialogue screens, many helping messages, ... Due to the

lack of time, only the explanation module could not be developed. However, we could compensate this lack through the use of the TRACE or DEBUG command of the Prolog debugger. These two commands allow the user to follow, step by step, the progression of the inference engine. This way thus enables us to understand how the prototype reached a particular result.

The quality of the decisions taken by the prototypes has been fully tested with a wide range of test cases. And the answers derivated from these test cases match with the criterions given by the experts. However, the tested systems are mere prototypes which gather only the basic concepts of a spool. The improvement of the reasoning techniques and the broadening of the knowledge base could thus be an extension to our work.

From the point of view of the value-added benefits, our prototypes bring out some of the numerous advantages of the expert systems. As a matter of fact, our prototypes provide the user with a lot of informations that are not available with the traditional spool systems. Thus, at any time, the user or the operator may get informations on the devices (e.g. the list of free devices; the list of devices with a particular form; the list of priority range allowed on each device; the list of devices reserved to a certain range of user-ids; ...) or informations on the jobs (e.g. the list of the forms, the classes, of the user-ids of the remaining jobs; the list of jobs which are not schedulable; ...). These rapid prototypes thus let us foresee already the numerous possibilities that could be offered by a real spool expert system.

The liability risk question is, in our case, irrelevant because our systems are just bare simulation prototypes that can not cause any damage to a particular organization.

Finally, evaluation enables a feedback process to take place whereby the comments serve as basis for iterative refinements. In that connection, we could review some of our initial choices such as the scheduling algorithms, the system configuration or the choice of language. As said before, there are a lot of different scheduling algorithms, each one of them validating some particular criterions. In our project, we have implemented three different algorithms. But this does not mean that these algorithms are the most pertinent ones, or the ones that offer the best performances. It is so advisable to consider other scheduling forms and, possibly, to compare the performances (qualitative and quantitative) of these systems with the performances of our prototypes. This comparison could then determine "the" scheduling process to adopt for a real expert system. The choice of the Prolog language could also be reconsidered. Even if this language is well suited for the development of expert systems, it still needs large computer resources (it particularly needs large stacks which have been a critical resource in the development of our prototypes). The use of another language or, possibly, the use of an expert system shell could therefore be considered. At last, the configuration of the expert system could also be reviewed. This configuration was defined as such because of the large computer resources necessary for the execution of a Prolog program. Its advantages

are to reserve all the power (in CPU and in memory) of a PC computer and to offer a specific console to the operator. However, this configuration is quite cumbersome (it needs an additional computer) and might be more costly than a standard configuration. It could so be redefined, especially if we plan to use another language.

Through the evaluation of our prototypes, we have partially answered to the second goal of the project (because we have shown how our prototypes already offer new functions that are not available with the conventional spools). But a real spool expert system could provide a lot more possibilities. In fact, a spool expert system could supply the user with a better quality of service (additional informations, better user interface, ...). But it could also offer a better management of the printers. Through its "experience" (i.e. the analysis of the previously solved problems and the quality of its knowledge base), a spool expert system could dynamically analyse the jobs in queue in order to process them under optimal conditions. It could make a dynamic estimation of paper changes to relieve the operator of his work. It could also help the system administrator in the choice of selection operands. In summary, many important new functions could thus be implemented through the use of expert system technology, leading to a new generation of spool systems.

Beyond the cliches, the realization of this work has enabled us to discover a promising approach in the development of spool scheduler : the expert system approach. It is likely that spool scheduler expert systems will develop in the near future, and it is to be hoped that, eventually, the quality of the services offered to the user will be highly improved.

APPENDIX

This appendix illustrates the architecture of the working memory in the second version of our spool scheduler prototype

```
dev([dev15,bull,addr15,200,[usr1,usr2,usr3],'','',[form4,form5,f
orm6],form5,[font1,font2,font3],font3,portrait,0,active]).
dev([dev14,siemens,addr14,240,'','',dia1,'',form5,[font3,font4,f
ont5,font6],font6,portrait,65000,active]).
dev([dev13,philips,addr13,215,[usr3,usr4,usr5,usr6],'','',[form1
,form2,form3,form4],form4,[font1,font5],font1,portrait,90000,act
ive]).
dev([dev12,ibm,addr12,225,'',[cl1,cl2,cl4],dia1,[form1,form3],fo
rm3,'',font5,portrait,0,active]).
dev([dev11,olivetti,addr11,245,'','',dia2,[form2,form5,form6],fo
rm2,'',font6,portrait,120000,active]).
dev([dev10,amstrad,addr10,235,[usr4,usr8,usr10],'',dia1,'',form1
,[font1,font2,font6],font2,landscape,0,active]).
dev([dev9,compaq,addr9,200,'',[cl1,cl2,cl3],'','',form5,'',font4
,portrait,80000,active]).
dev([dev8,hewlett,addr8,235,[usr1,usr7,usr8],[cl1,cl2],dia1,[for
m3,form5,form6],form3,'',font3,portrait,0,active]).
dev([dev7,xerox,addr7,200,'','','',[form2,form4],form4,[font1,fo
nt2,font4,font6],font2,portrait,0,active]).
dev([dev6,digital,addr6,250,[usr4,usr6,usr7,usr8],'',dia3,'',for
m4,'',font3,portrait,95000,active]).
dev([dev5,toshiba,addr5,200,'','','',[form3,form5,form6],form6,[
font2,font3],font3,landscape,0,active]).
dev([dev4,bull,addr4,240,'','',dia3,[form3,form4,form6],form4,''
,font3,portrait,0,active]).
dev([dev3,siemens,addr3,255,'',[cl2,cl3],'','',form5,[font1,font
2,font6],font6,portrait,120000,active]).
dev([dev2,oki,addr2,230,[usr7,usr8],'',dia1,[form5,form6],form5,
'',font1,portrait,0,active]).
dev([dev1,ibm,addr1,200,[usr1,usr2,usr3],'',dia3,[form2,form5,fo
rm6],form2,[font1,font4],font4,portrait,0,active]).
dev([xxxxxxx,1,1,1,1,1,1,1,1,1,1,1,1,1]).


form([form6,single,a4,none,none,none,'']).
form([form5,single,a4,long,none,none,f2]).
form([form4,listing,a4,long,long,none,'']).
form([form3,listing,a4,brief,brief,none,'']).
form([form2,listing,a4,brief,brief,brief,f1]).
form([form1,listing,a4,none,none,none,'']).
form([xxxxxxx,1,1,1,1,1,1]).
```

```
font([font6,high,[bold,italic],courier,12]).
font([font5,high,'',courier,12]).
font([font4,normal,[bold],times,12]).
font([font3,draft,[underlined],courier,12]).
font([font2,draft,'',courier,14]).
font([font1,high,[underlined,bold],times,14]).
font([xxxxxxx,1,1,1,1]).


job([15,file15,'',0,0,form3,'','',portrait,80000,240,usr2,cl1,0,
n,15,active]).
job([14,file14,'',20,30,form5,'',dia2,portrait,50000,255,usr10,c
l3,0,n,15,active]).
job([13,file13,'',0,0,'',font4,'',portrait,110000,210,usr7,cl2,0
,n,15,active]).
job([12,file12,'',10,10,form1,font6,dia1,portrait,2000,210,usr11
,cl1,0,n,15,active]).
job([11,file11,'',0,0,'','','',landscape,12000,255,usr1,cl2,0,y,
25,active]).
job([10,file10,'',0,0,'','',dia3,portrait,56000,220,usr13,cl2,0,
n,15,active]).
job([9,file9,'',0,0,form6,font1,'',portrait,34200,235,usr9,cl1,3
,n,5,active]).
job([8,file8,'',0,0,form2,'',dia1,landscape,25500,215,usr3,cl3,0
,y,10,active]).
job([7,file7,'',10,70,'',font5,'',portrait,120000,240,usr20,cl1,
0,n,15,active]).
job([6,file6,'',0,0,'','',dia3,portrait,7000,230,usr11,cl1,0,n,1
5,active]).
job([5,file5,'',0,0,'','','',portrait,4500,245,usr7,cl2,0,y,15,a
ctive]).
job([4,file4,'',0,0,form5,font2,dia2,portrait,34800,200,usr4,cl1
,0,n,15,active]).
job([3,file3,'',0,0,'','','',landscape,500,150,usr8,cl3,2,n,1,ac
tive]).
job([2,file2,'',0,0,form6,'',dia1,portrait,3500,255,usr6,cl1,0,n
,15,active]).
job([1,file1,'',0,0,'',font3,'',portrait,5500,244,usr14,cl2,0,n,
15,active]).
job([0,xxxxxxx,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]).
```

BIBLIOGRAPHY

[ALT-84] : ALTY, J.L., *Expert systems - concepts and examples*, The National Computing Centre Ltd, Manchester, 1984.

[BIL-84] : BILLMERS, M.A., SWARTWOUT, M.W., *AI-SPEAR : a computer system failure analysis tool*, in *Proceedings of the Sixth European Conference on Artificial Intelligence*, September 1984, pp. 61-69.

[BON-85] : BONNET, A., *Artificial intelligence - Promise and performance*, Prentice-Hall International Ltd., London, 1985, pp. 142-216.

[BOO-84] : BOOSE, J.H., *Personal construct theory and the transfer of human expertise*, in *Proceedings of the National Conference on Artificial Intelligence*, August 1984, pp. 27-33.

[BRA-86] : BRATKO, I., *Prolog - programming for artificial intelligence*, International Computer Science Series, 1986.

[CHO-87] : CHORAFAS, D.N., *Applying Expert Systems in Business*, McGraw-Hill Book Company, New York, 1987.

[CLA-83] : CLANCEY, W.J., *The advantages of abstract control knowledge in expert system design*, in *Proceedings of the National Conference on Artificial Intelligence*, August 1983, pp. 74-78.

[FOR-87] : FORD, N., *How machines think*, John Wiley & Sons Ltd., Chichester, 1987, pp. 171-185.

[GEV-84] : GEVARTER, W.B., *Artificial intelligence - expert systems - computer vision and natural language processing*, Noyes Publications, Park Ridge, 1984, pp. 71-86.

[GEV-85] : GEVARTER, W.B., *Intelligent machines - an introductory perspective of artificial intelligence and robotics*, Prentice-Hall Inc., Englewood Cliffs, 1985, pp. 47-66.

[GOO-85] : GOODALL, A., *The guide to expert systems*, Learned Information Ltd, 1985.

[GRI-84] : GRIESMER, J.H., HONG, S.J., KARNAUGH, M., KASTNER, J.K., SCHOR, M.I., *YES/MVS : a continuous real time expert system*, in *Proceedings of the National Conference on Artificial Intelligence*, August 1984, pp. 130-136.

[HAY-83] : HAYES-ROTH, F., WATERMANN, D.A., LENAT, D.B., *Building expert systems*, Addison-Wesley Publishing Company Inc., Reading, 1983.

[HEN-85] : HENNINGS, R.D., *Artificial intelligence - 1. Expertsysteme*, Mathware-Verlag GmbH, Berlin, 1985, pp. 271-310.

[HU-87] : HU, S.D., *Expert systems for software engineers and managers*, Chapman and Hall, New York, 1987.

[JAC-86] : JACKSON, P., *Introduction to expert systems*, Addision-Wesley Publishing Company, Reading, 1986.

[JAN-85] : JANSON, P.A., *Operating systems - structures and mechanisms*, Academic Press Ltd., London, 1985, pp. 71-100.

[KEL-87] : KELLER, R., *Expert system technology - Development & application*, Prentice-Hall Inc., Englewood Cliffs, 1987.

[KLA-86] : KLAHR, P., WATERMAN, D.A., *Expert systems - techniques, tools, and applications*, Addison-Wesley Publishing Company Inc., Reading, 1986.

[KRI-87] : KRIZ, J., *Knowledge-based expert systems in industry*, Ellis Horwood Ltd., Chichester, 1987, pp. 17-23.

[LEE-88] : LEE, SUH, *PAMS : a domain-specific knowledge-based parallel machine scheduling system*, in Expert Systems, Vol. 5, No. 3, August 1988.

[LIE-86] : LIEBOWITZ, J., *Useful approach for evaluating expert systems*, in Expert Systems, Vol. 3, No. 2, April 1986.

[MAR-88] : MARTIN, J., OXMAN, S., *Building expert systems - a tutorial*, Prentice-Hall Inc., Englewood Cliffs, 1988.

[MIL-87] : MILENKOVIC, M., *Operating systems - concepts and design*, McGraw-Hill Book Company, New York, 1987, pp. 99-121.

[NAY-84] : NAYLOR, C., *Build your own expert system*, Sigma Technical Press, Wilmslow, 1984, pp. 1-12.

[NEG-85] : NEGOITA, C.V., *Expert systems and fuzzy systems*, The Benjamin/Cummings Publishing Company, Menlo Park, 1985, pp. 1-47.

[PIN-89] : PINKERT, J.R., WEAR, L.L., *Operating systems - concepts, policies, and mechanisms*, Prentice-Hall Inc., Englewood Cliffs, 1989, pp. 232-248.

[RAU-88] : RAUCH-HINDIN, W.B., *A guide to commercial artificial intelligence*, Prentice-Hall Inc., Englewood Cliffs, 1988, pp. 236-282.

[ROL-87] : ROLSTON, D.W., *A multiparadigm knowledge-based system for diagnosis of large mainframe peripherals*, in *Proceedings the Third Conference on Artificial Intelligence Applications*, February 1987, pp. 150-155.

[SAV-88] : SAVORY, S.E., *Artificial intelligence and expert systems*, Ellis Horwood Ltd., Chichester, 1988, pp. 15-35.

[SHI-84] : SHIRAI, Y., TSUJII, J-I., *Artificial intelligence - concepts, techniques, and applications*, John Wiley & Sons Ltd., Chichester, 1984, pp. 103-131.

[SIE-86] : SIEGEL, P., *Expert systems - a non-programmer's guide to development and applications*, Tab Books Inc., Blue Ridge Summit, 1986, pp. 1-159.

[SIL-87] : SILVERMAN, B.G., *Expert systems for business*, Addison-Wesley Publishing Company Inc., Reading, 1987, pp. 5-51.

[SIM-84] : SIMONS, G.L., *Introducing artificial intelligence*, The National Computing Centre Ltd., Manchester, 1984, pp. 175-207.

[SIM-85] : SIMONS, G.L., *Expert systems and micros*, The National Computing Centre Ltd., Manchester, 1985.

[STE-86] : STERLING, L., SHAPIRO, E., *The art of Prolog : advanced programming techniques*, The MIT Press, Cambridge, 1986.

[TOW-86] : TOWNSEND, C., FEUCHT, D., *Designing and programming personnal expert systems*, Tab Books Inc., Blue Ridge Summit, 1986, pp. 1-47.

[WAT-86] : WATERMAN, D.A., *A guide to expert systems*, Addison-Wesley Publishing Company Inc., Reading, 1986.

[WEI-84] : WEISS, S.M., KULIKOWSKI, C.A., *A practical guide to designing expert systems*, Rowman & Allanheld Publishers, Totowa, 1984.

[WIN-86] : WINSTON, P., PREDERGAST, K., *The AI business - commercial uses of artificial intelligence*, The MIT Press, Cambridge, 1986, pp. 15-51.

[WOL-87] : WOLFGRAM, D.D., DEAR, T.J., GALBRAITH, C.S., *Expert systems for the technical professional*, John Wiley & Sons Ltd., Chichester, 1987.