CHAPTER 3

EXPERT SYSTEMS

Artificial Intelligence researchers blend cognitive psychology and computer science in their attempt to make computers smarter and more like human beings. Making an intelligent machine has always been a dream of humanity. As a science, Artificial Intelligence deals with any problem still not having a known algorithmic solution. Therefore, Artificial Intelligence has produced real achievements in many fields. They all deal with symbolic knowledge and require choices, a major component of intelligence. Among the various fields such as robotic, natural language comprehension, speech recognition, computer vision, learning processes, games, problem resolution and expert systems, expert systems are quite popular and fast developing. They attempt to simulate the behaviour of human experts.

3.1. ABOUT EXPERT SYSTEMS

3.1.1. Objectives and philosophy of expert systems

The interest of capturing expert knowledge is great in many fields since experts are scarce and busy. It takes time, experience and intelligence to become an expert and yet expert knowledge is likely to disappear with the expert. Therefore the achievement of an expert system would provide an everlasting, always at your disposal "expert". This would free the expert from routine time consuming appraisal and enable him to spend more time on updating and making his knowledge precise. During a lifetime people acquire experience unconsciously; with this experience they can behave efficiently, rapidly and arrive at decisions with little "calculation"; this is the experience that expert systems want to possess and utilise. Since this knowledge is fluctuating and ill defined, its application in traditional programming is very difficult.

When experts make an expert appraisal, they analyse the present situation and search for any relevant details symptomatic of a particular situation, making correspondences with analogic or opposite situations seen previously to draw proper conclusions. This requires intelligence and up to a certain point expert systems can perform this kind of intelligent action.

By their mimetic behavior, expert systems produce deceptive substitutes for reasoning which lead to unavoidable queries on the nature of human reasoning and what is amenable to modelling today by means of an expert system, and the achievements that may be contemplated. To find out how far expert systems could simulate human reasoning behavior,
investigations are going on in two directions:

- What are the fundamental mechanisms of generalisation, abstraction and inference used to simulate different patterns of human reasoning (cf. § 1.8.)
- What are the causal reasoning, classification and heuristic reasoning that are irreducible and can be formalised?

3.1.2. Learning process

Most experts have become so by making expert appraisals. Continuously, they learn from the appraisal that they make and thus validate the rule of thumb, intuitions and shortcuts that they use in their reasoning process. The incapacity of the expert systems to perform this learning process is an evident drawback to handing over expert appraisal to the machine. As far as the means used by the experts to learn from experience are not clear, generalisation from particular situations and elaboration of patterns cannot be automatically done. But analysing how experts always refer to their previous experience, or at least to some general lessons drawn from it, gives some enlightenment on their learning process.

The learning process contributes to:

- the updating of the expert knowledge and heuristics.
- apprenticeship, i.e. automatic acquisition techniques of symbolic knowledge based upon generalisation and classification.

Different types of learning process exist; some expert systems try to implement these learning methods:

- elaborating classes based upon the generalisation of concepts and specialisation.
- parameter adjustment: once the reasoning is completed, the programme tries to find out whether some other values of the parameters could have given a better result.
- progressive reconfiguration of an initial pattern, based upon a combined use of both deep (basic structure with all its components, their linkage, or behavioural model of the target object or a causal model between the object and its properties) and shallow knowledge (merely experience based) which gives quick access to answers and decisions. This aims at identifying the part of deep knowledge actually used to order research space so as to get quick access to the solution in every common case.
- analogy (Carbonell): identification of concepts between source and target objects, identification of the elements of information upon which the analogy can be based, creation of a link between analogous objects, validity of the structural link thus created.
3.1.3. Characteristics of the expert system softwares

Theoretically expert systems are capable of reasoning through a procedure comparable to the one adopted by the specialist solving a problem pertaining to his discipline (physician, geologist...). To begin with, the designer of an expert system identifies and collects from the human expert the knowledge used by the latter. Not just theoretical knowledge, but also empirical knowledge acquired from experience and used by the specialist for a successful application of the theory.

After the knowledge acquisition phase, the designer of an expert system (i) selects or devises a suitable representation for the knowledge, and (ii) chooses or formulates the required control strategy that gives the power of reasoning to the expert systems. Thus one finds that in expert systems, unlike conventional computing systems, knowledge and control are made distinctly identifiable components.

The difference between algorithmic and declarative programming is a key concept of expert systems. In order to carry out a traditional programme, information scientists prepare a method of resolution (an algorithm) describing a step by step procedure of treating data, culminating in a certain result. (imperative programming). On the contrary in expert systems, a variety of knowledge is stored in the knowledge base. Any chunk of knowledge, thus stored makes sense by itself. These chunks are formulated in such a way that hypothesis is made not on how they are going to be used but on what they imply (declaratory programming). Therefore, this knowledge can be accessed any time according to the inference engine intimations. One of the essential characteristics of these softwares is their ability to deal with qualitative information and symbolic data besides the classical quantitative characteristics: Chunks of knowledge are based on the semantics of knowledge rather than on the means used to access the value of knowledge. While in conventional programming, the sequence of orders is based upon a rigid logical sequence of orders, expert systems draw some information (uncertain and partial conclusions) from fuzzy logical inferences. Nevertheless, formal logic is also used to prevent the coexistence of two contradictory elements.

Typically computer programmes have two components: the algorithm and the data, whereas the structure of an expert system includes three components: the knowledge base (facts and rules), the user interface, the inference engine.
3.1.4. Expert system Structure

![Diagram of expert system structure]

fig No.3.1: Organisation of an expert system

3.1.4.1. The knowledge base

The knowledge base contains all the information specific to the field of experience which is stored according to the knowledge representation conventions (academic notions as well as rules and heuristics used by the expert when he appraises). The knowledge base consists of the rule base and the fact base:

- At first, the fact base contains all the crude information initially given to the expert system by its modeller and the patterns that his knowledge and experience have led him to build up. These symbolic objects are condensed pictures of stereotyped situations.

- The rule base is the collection of all the links between the various elements of the fact base as well as the heuristics to be applied.

Example

```
if <tank_storage is silted up> then <reduction_of_water_availability>
& check <reduction_of_command_area>
& check <reduction_of_cropping_intensity>

if <sluice_leaking> then <low_system_water_efficiency>
suggest <management_problem>
suggest <maintenance_problem>
suggest <not_adequate_water_pricing>
```
After launching the execution, the fact base constitutes the working memory. Then it contains also the specific information concerning the problem to be processed and the elements of "demonstration" which have already been established by the expert system.

3.1.4.2. The inference engine

The inference engine is a programme which utilises the knowledge and the heuristics contained in the knowledge base to solve the problem specified by the user. The inference engine carries on the reasoning process from the information received through the user interface in the course of the execution and the data available in the fact base.

The process of solving a problem is generally diagrammed as a search tree, with each node in its branching structure representing a new state (a new configuration of knowledge) which can be the solution or a step toward the solution. The major difficulty in searching a complicated search tree, to solve a complex problem, is that examining all alternatives involves an unreasonable amount of time on even the fastest computer (NP complete problems); therefore heuristics and strategies have to be added to limit the search. Per contra, the strategies used to limit the combinatory trials (constraining the problem resolution) act against a systematic analysis and may decrease the quality of the solutions obtained (you can never be sure they are the best ones).

In § 2.6 we have discussed the different reasoning logics used by experts. Based on these considerations, different inference engines have been designed according to the type of logical chaining they use (Farenny, 75):
- Some are goal driven and work with backward chaining: the final goal to be verified is reduced to a set of subgoals that the inference engine tries to verify. If these subgoals cannot be verified, the engine searches for another set of subgoals that could lead to the same conclusions. If this is still not possible, the initial hypothesis is rejected and a new goal is chosen.
- Some follow a forward chaining logic and start from the available information and those available in the knowledge base to draw conclusions following the heuristics traditionally used by the expert. It goes on from the verified facts until the required goal is deduced.
- Some combine both the procedures.

Traditionally, it is advisable to use backward chaining when the number of goals is limited and the number of initial facts high and forward chaining when the number of initial facts is limited but the number of potential goals is high.

Another means of classifying expert systems accounts for the order of logic of the engines:
- Zero order logic: only deterministic, simple objects and the logical links between these objects can be represented.
-first order logic: more complex objects can be dealt with and unknown quantities and variables can be used as arguments of a relation.
-second order logic: deals with variables which can be relational symbols.

The inference motor is the heart of an expert system. The reasoning process appropriate to the particular case is treated repeating the basic inference cycle and its four sequential steps are restriction, filtration, conflict resolution and execution:
-'A priori' determination of the rules likely to be triggered in the present state of the problem is called the restriction step.
-Identification of the appropriate rules, all the premisses of which are verified, is called the filtration step. At this step, the software tries to match the present new situation with the patterns of its knowledge base.
-Choice of the rule to be triggered among the fireable ones. This step is the conflict resolution step. It is based on a criterion that is characteristic of the engine mode of conflict resolution and could be one of the following (some engines make it possible to associate different criteria to different rules):
  -the first rule is the one to be triggered
  -the more concise rule is the right one
  -the more recently fired rule
  -the more informative one or
  -the more constrained one
These are some criteria that human beings use alternatively in an adaptable strategy whereas the inference motor strategy is less flexible.
-Start of the chosen rule: the execution.

Engines can have two main strategies of exploration of the graph:
-when the exploration is said "in depth first", the priority is given to the new rules fireable,
-in the other strategy said "in width first", all the potential rules are tested before any one is executed.

Another distinction between the inference engines is the control type:
-the control may be irrevocable. When the set of the fireable rules is empty either the engine returns to the prior step and tries to fire another rule, or it stops.
-otherwise, the engine is said to be working by "successive attempts and backtracking"; prior conclusions are suppressed and the reasoning becomes non monotonous. All the intermediate possibilities exist. Therefore it is important to know which is the most adequate reasoning process according to the present subject.
The tracing facilities and the explication modules are also interesting properties of these softwares. The expert systems must be able to explicitly account for their behaviour and explain their procedure and the results obtained.

3.1.4.3. The user interface

The user interface enables people not aware of the knowledge representation used in the software to use the expert systems; it has to be as convenient as possible. De facto the knowledge base is a heap of facts and rules which by themselves do not lead anywhere. The expert system is designed to answer a problem, provided the problem is suggested to it and it is given some elements of answers to its questions. This is done through the user interface.

The user interface conveys to the user the questions raised by the software as well as the information presently required, and gives back the answers (or the non availability of the information) to the software. This is all the more interesting as it makes it possible to feed the system selectively with the relevant information for the present reasoning stage.

The facts within the knowledge base are given an order of source slot that specifies whether the corresponding information:
- has to be inferred, not to be asked for.
- has only to be asked for, cannot be inferred.
- has to be asked for before probing rules concluding with them.
- has to be asked for only after failure of the rules concluding with them.

3.1.5. Knowledge representation

In making a diagnosis of a patient or in trying to find out the cause of a motor breakdown, doctors and mechanics not merely call upon their reasoning capacity. They also call upon knowledge of all kinds acquired during their practice. Therefore, all existing types of knowledge have to be represented and stored so as to be easily accessible when necessary.

The elements of information could be classified as:

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>weir is a component of tank</td>
</tr>
<tr>
<td>classification</td>
<td>groundnut is a kind of crop</td>
</tr>
<tr>
<td>definition</td>
<td>privileged farmer is a farmer having a borewell whose land is in the head reaches</td>
</tr>
<tr>
<td>law, axiom</td>
<td>land tax equal to one tenth of the production</td>
</tr>
<tr>
<td>strategy</td>
<td>if sluice is leaking, suspect maintenance problem</td>
</tr>
</tbody>
</table>

...
Knowledge representation is the logical representation used by the software to formalise the elementary chunks of knowledge in adequate forms (Charniak E., 1985). Many different types of representation have been used depending on the type of software used, on the language structure, and on the type of problem. Some of them may be more convenient than others. It is rather a difference in tool than a basic conceptual difference.

3.1.5.1. The production rules

The rules are generally formalised with production rules and is concerned with all the rigorous pieces of reasoning (Lauriere J.L., 1983):

IF <condition> then <consequence>
IF <premisses> then <conclusion>
IF <sluice_leaking> true then <suggest><problem in Operation&Maintenance>
and <low_water_efficiency> is confirmed

Rules may be written in zero order logic (no variables are admissible), or in first order logic (variables are admissible in the rules), or even in higher logical order, depending on the engine. The higher the order is, the more difficult and long the pattern matching. Though it might be convenient to define steps, subsystems, to structure the knowledge base and hasten the inferences, all the rules should be given without any order since in theory, the corresponding chunks of knowledge are assumed to be independent and of equal importance. The independence of the rules is essential to enable easy addition and withdrawal. The overall speed of the system and its adaptability vary in the reverse way.

3.1.5.2. Semantic networks

Semantic networks (Kochen M.) consist of nodes which are objects, concepts and events, and of the links or symbolic references between these nodes which represent their relationships.

3.1.5.3. Triplets and lists

Some expert systems shells (a shell is a development environment) like SNARK, OPS5 use triplets and lists:

triplets <object>, <relation>, <value>
examples:

field-A crop rice
tank name Kodur
tank location South Arcot
Lists are more convenient for objects having a lot of attributes.

3.1.5.4. The frames

Other softwares use frames. A frame is a structured object dominated by a central concept (event, person, notion) and includes a set of slots describing attributes of that concept. Each attribute is labelled optional or non-optional for each of the stereotyped situations created in the programme. It is thus possible to determine which attributes are more important for the present situation. The missing information can be filled in from contextual values or default attributes. (Minsky M. 1975) If all the non-optional attributes of a particular slot match with those of a particular frame, the frame itself is validated. Frames are a typical representation issued from the concepts of Object Oriented Language which defines structured objects and gives default values to 'not yet defined' or 'not defined' attributes by automatic inheritance from the parent object.

3.1.5.5. The scripts

The scripts (Shank R. and Abelson R.P.) summarise common human experiences. The stereotyped situation described in the scripts enables the programme to fill in the missing information and to make assumptions about the statement based on the context in which the statement is made.

3.2. EXISTING EXPERT SYSTEMS SHELLS AND TOOLS

In the 70's, expert systems were developed for the first time at Stanford University. The first attempt by Feigenbaum in 1965 aimed at determining the structure of an organic molecule based on input obtained from a mass spectrophotograph. Experts solved the determination problem by combining spectrophotographic data and heuristics developed from experience.
3.2.1. MYCIN

Later the same team developed MYCIN in the field of medical diagnosis (Shortlife B.E. and Buchanan B.G., 1975) which is one of the most famous expert systems. Therefore it has been much studied and often is considered as the archetype of expert systems. It combines various know-how and know-why regarding the diagnosis of bacterial blood infections and prescribes adequate antibiotic therapies. Asking questions concerning the patient's symptoms, the result of bacteriological analysis, etc., it concludes what infection is likely to be responsible for the present patient's state, and moderates its conclusion by a kind of probability. Mycin has a rule base and a backward chaining inference engine. It was the first expert system to incorporate an explanation module stating why a particular item of information was wanted and how conclusions were reached. It accounts for uncertainty with a likelihood coefficient which has often been considered by many as a dirty recipe.

Later on, separating the knowledge itself and the system that manages the knowledge base and controls the reasoning process, Mell W.J. developed an expert systems shell (inference engine and knowledge representation), EMYCIN which has been widely used to develop many other expert systems. On the same pattern PUFF has been built to comment on the results of a lung spirometric analyser and is presently working in hospitals with great success.

3.2.2. Other expert systems

3.2.2.1. Expert systems in the agricultural field

The diagnosis of phytopathological infections is one of the most advanced fields of agricultural expert applications. In such domains knowledge is quite sharp, the number of parameters limited and the problem to be solved quite precise. The knowledge concerned with the diagnosis is not evolutive and the consultation of the expert occurs at a defined time. (no need to account for time evolution parameters which are specially ticklish regarding pattern matching).

Tom (tomato diseases diagnosis INRA France) is another expert system. Its main advantages besides the diagnostic are:

- that it preserves the experience and knowledge of the one sharp expert of a field,
- that it makes the knowledge of all the top level experts available for every technician at the same time,
- that it leaves the expert free to investigate, (where he is found more productive) freeing him of expertise, and
- that it constrains the expert to cut to pieces and specify his reasoning behaviour.
The success that Tom obtained is responsible for the growing interest for expert systems in the agronomic field.

A pilot project testing the applicability of using a dynamic crop simulation model of cotton as the basis for a crop management expert system has been developed in California (McKinion & Al, 1980); COMAX-GOSSYM can be used to make preplanting decisions for planting date, row spacing, nitrogen application at planting, and plant population. You can also use the programme to make decisions on nitrogen applications and irrigation scheduling during the season. This preplant decision uses simulation: a full run of the programme is made with various values of the factor being looked at. By comparing the yields for the various runs Comax can determine which is the most beneficial. Comax can also determine the effects of weather. Any recommendation is based on not less than thirty simulations minimising the water and nutrient stress. Gossym is a process level model which simulates the plant physiological, micrometeorological and soil physical processes. It is considered a state of the art model of the cotton plant. Comax-Gossym is written in Lisp for the expert system part and the procedural attachment and the algorithms are in Fortran 77.

Other expert systems that have been developed are:
- Cow mastitis diagnostic with SNARK (rules base, forward chaining development tool), and
- Fodder planning with OPS5 (object, backward chaining development tool).

3.2.2.2. PROSPECTOR

There is a good deal of argument about what topics are relevant for building expert systems and what type of machine should be used for such applications. Some practitioners argue that the only problem that merits an expert system are those that will yield a high economic pay off. Such people disdain microcomputer based tools and set the price of developing expert systems in millions of dollars. Prospector, a mineral exploration expert system is one of them; the heart of the expert system is the KAS inference engine. In the first step this expert system is fed with the available data, then the inference engine computes the five probabilities of finding the 5 mineral ores which Prospector is skilled to look for; this gives the user the possibility of giving up the investigation of some side minerals and focusing on the most probable ones. From this point, Prospector starts backward chaining to finalise the investigation.

Others share the view that anyone can develop his own expert system in a useful, though not in a spectacular manner using expert system development tools now available on microcomputers. Therefore you do not have to write the expert system from scratch in any programming language.
3.2.2.3. Expert systems in the social sciences

Some applications have also been made in the field of social science; quite early in 1966, Eliza was proposed to the public: Eliza used to mimic the answers of a psychiatrist. Many patients were lured by Eliza to tell all about their lives. They had more confidence in the computer than in human beings and felt that they were understood by the computer better than they had ever been by human beings. Yet Eliza was only building sentences picking the structure of one of the readymade patterns available in its knowledge base and in using the vocabulary of the patient's sentence.

3.3. LANGUAGES ORIENTED TOWARD ARTIFICIAL INTELLIGENCE

3.3.1. LISP

We have already emphasized that one of the main characteristics of Artificial Intelligence is dealing with symbolic knowledge. To develop expert systems, new programming languages specifically conceived for symbol manipulation, have been created. Among them, Lisp is one of the most used languages. (Charniak E. & Al, 1985)

Symbol manipulation is like working with words and sentences (concepts); therefore, in LISP:
- the fundamental objects are wordlike objects called atoms.
- groups of atom form sentence-like objects called lists, a concept which has been used with profit to create object oriented expert systems and frames.
- lists themselves can be grouped into higher level lists. This gives the essential ability to form hierarchical groups.
- atoms and lists collectively are called symbolic expressions which constitute Lisp's fundamental way of working with symbols.

Moreover Lisp has some specificities that make it a convenient language for Artificial Intelligence (Winston P.H., 1984):
- interactivity: Lisp is oriented toward programming at a terminal with rapid response. All the procedures and data can be displayed or altered at will.
- Lisp environment is handy for quick and easy debugging of programmes.
- Lisp procedures and Lisp data have the same form. One Lisp programme can use another as data; it can even create and use it within the same programme.
- all programmes are functions with arguments and only one returned value. This is compatible with Horn's clause. (Horn's clause i.e. a restriction for IF/THEN rules compulsory for backward chaining logic which specifies that the rules may have any number of premisses but only one conclusion).
3.3.2. PROLOG

Prolog is a programming language used to solve problems of "objects" and "links" between the objects. Therefore to build a programme in Prolog one must:
- declare the facts or the assertions on the objects and their interactions,
- set the rules on the objects and their interactions,
- ask questions on the objects and their interactions.

In fact Prolog has been conceived as an inference engine (forward chaining, first order logic) and the whole language and programme writing is like expert systems. It is quite a slow language even though it is a very powerful one. It has been chosen by the Japanese as the language of their 5th generation computers. Yet the facilities and the flexibility offered by rough Prolog used as an inference engine are not as good as those offered by other expert system development tools. Programming in Prolog requires high level skills for programmers whereas programming with some expert system development tools can be done without having to get into the inference engine strategy and algorithm.

3.3.3. OOL: Object Oriented Language

In object oriented programming (Pascoe G.A., 1986), objects are considered as instantiation of classes; the class provides all necessary information to construct and use objects of a particular kind and its instances. A class provides storage for methods which are procedures invoked in sending a message to a class's instances (stored as local variable) whereas the instance state is stored as instance variables; information hiding, data abstraction, dynamic binding and inheritance; four characteristics of object oriented programming.
Information hiding: the state of a software module is contained in private variables, visible from only within the scope of the module. Only a localised set of procedures directly manipulates the data. It gives flexibility since modules can be modified without affecting the remaining parts of the programme.
Data abstraction: abstract data types are defined as consisting in an internal representation plus a set of procedures used to access and manipulate data.
Dynamic binding: because of data abstraction many different types of variables can be stored on the stack and a procedure acting on the data stored on the stack must get information regarding the type of variable it operates on; this is done through dynamic binding.
Inheritance: Inheritance enables programmers to create parent structures that propagate their methods and values to their sons. Thus subclasses can be defined by specialisation of super class without the need of an extracode.
<table>
<thead>
<tr>
<th>MANUFACTURER</th>
<th>EXSYS</th>
<th>Insight</th>
<th>EES</th>
<th>P.C. envy</th>
<th>Mr. EXPERT</th>
<th>Expert Kit</th>
<th>MP LEO</th>
<th>GET</th>
<th>GETS</th>
<th>GETEX</th>
<th>MI</th>
<th>SNARK</th>
<th>S1</th>
<th>SNOPS</th>
<th>NEXPERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNOWLEDGE REPRESENTATION AND REASONING STRATEGIES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF/THEN-rule</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>frames</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>decision tree</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>forward chaining</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>backward chaining</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>multiple answers</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>certainty factor</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>combined certainty</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>maximum nb. of rules</td>
<td>300</td>
<td>1000</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td>dep. on config.</td>
<td></td>
</tr>
<tr>
<td>DEVELOPER AND USER INTERFACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>skill level</td>
<td>novice</td>
<td>intermediate</td>
<td>expert</td>
<td>intermediate</td>
<td>novice</td>
<td>intermediate</td>
<td>expert</td>
<td>intermediate</td>
<td>expert</td>
<td>intermediate</td>
<td>expert</td>
<td>intermediate</td>
<td>expert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ease</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>exploration mode</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>INTEGRATION, DELIVERY AND SUPPORT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>internal data</td>
<td>ascii files</td>
<td>ascii files</td>
<td>ascii files</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td>ascii</td>
<td></td>
</tr>
<tr>
<td>external programs</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td>ascii, DBASE</td>
<td></td>
</tr>
<tr>
<td>compiled system support</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td>Mac, PC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig No.3.2: COMPARATIVE TABLE OF SOME EXPERT SYSTEM SHELLS
3.4. THE EXPERT SYSTEMS SHELLS AVAILABLE

Expert systems shells cover a wide range of tools; some of them are quite simple, while others are quite elaborate. All of them have a knowledge representation scheme, an inference or search mechanism, a means of describing a problem and a way to determine the status of a problem while it is being solved. Some other system functions may also be available and will aid in selecting an adequate development tool, once the most desirable features to look for in an expert system development environment have been identified.

Citrenbaum,(1988) has given a list of relevant tools or features that may be useful:
- consistency and completeness checking,
- high quality explanation
- optimised query sequences in rule base expert systems
- different knowledge representation forms
- dynamic user influence of hypothesis generation
- uncertain dealing with less than certain conclusions
- audit trail
- links with widely used products
- compatibility with other shells and tools
- interface with algorithmic and procedural language
- guidance
- dealing with graphics
- default values
- inheritance mechanisms...

Fig No.3.2 gives a comparative table of some expert systems with a subset of the above criteria.

3.5. BUILDING AN EXPERT SYSTEM

Knowledge acquisition as well as building an operational expert system needs a well defined and structured problem as well as identified and available experts.

3.5.1. Methodology for building expert systems

Traditionally the expert system development is considered under four stages: problem selection, initial prototype, expanded prototype and delivery system.(fig No. 3.3):
- Problem identification, determination and specification correspond to the requirement stage in a conventional development software. The major objective of this stage is to ensure that the problem will satisfy a real need and be technically feasible.
- The initial prototype development includes the selection of an inference mechanism, an adequate knowledge representation and a development tool to demonstrate the technical and
economical feasibility of the target expert system on an initial subset problem.

- The next phase of development called an expanded prototype, will consist of devising a suitable expert system architecture, knowledge representation and strategy (according to project complexity). This phase should lead to the prompt completion of the project with a limited budget and still enough knowledge to ensure that all essential parameters are included. The initial problem subset is expanded to full complexity and interaction with the related systems (data base, programme, measuring equipments, video interaction and so forth). Validation of the expert tool will occur at this level.

- The fourth phase leads to the final delivery product, with integration of appropriate hardware constraints and optimisation.

3.5.2. Knowledge acquisition

Knowledge acquisition is a long, difficult and ill formalised process. It requires collection of the experts' knowledge and the real tactics of their thinking process. However, it may be mentioned that the extraction of their knowledge is not an easy task. Here are some of the standard methods used to extract knowledge:

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method of &quot;familiar&quot; tasks</td>
<td>Analysis of the tasks that the expert usually performs</td>
<td></td>
</tr>
<tr>
<td>Structured and unstructured interviews</td>
<td>The expert is queried with regard to knowledge of facts and procedures</td>
<td></td>
</tr>
<tr>
<td>Limited information tasks</td>
<td>A familiar task is performed but the expert is not given certain informations that is typically available</td>
<td></td>
</tr>
<tr>
<td>Constrained processing type</td>
<td>A familiar task is performed but the expert must do so under time or other constraints</td>
<td></td>
</tr>
<tr>
<td>Method of tough cases</td>
<td>Analysis of a familiar task that is conducted for a set of data that presents a tough case for the expert</td>
<td></td>
</tr>
</tbody>
</table>

fig No.3.4: Knowledge acquisition methods (Hoffman, 1987)

All the foregoing methods cannot always be applied, and they do not have the same advantages. Choice of a method depends on the type of tasks which they are used for (simple tasks (brevity of the instructions needed), short tasks (time), flexible tasks), the type of data they provide (usable rules, data validity) and on their efficiency (amount of informative proposition they enable you to obtain per minute).


A method is not necessarily a key to success:
- There often are semantic problems that hamper communication: current words are used in such a way that their inaccuracies and ambiguities mask the real reasoning process.
- Experts often have a natural tendency to present their knowledge in a logical and didactic way which does not highlight their real reasoning mechanisms.
- The expert is often unable to formulate the knowledge which appears obvious to his mind and translating common sense into rules remains a critical point.
- Uncertain inferences are still another problem which is not satisfactorily tackled in spite of the likelihood coefficients or because of these coefficients; these coefficients are far less flexible than experts' real methods.
- In handling difficult problems, people proceed by stages; to cover all difficulties their reasoning is often inexact and general and then, as contradictions appear they gradually reorganise their thoughts until all contradictions disappear. It is quite difficult to allow temporary inconsistency in the base and to establish qualitative inconsistency thresholds above which the patterns have to be modified.
- Consensus among experts should be a must to validate knowledge in a multi-experts base and preserve its consistency.

3.5.3. Formalisation

The selection of a defined type of language, engine and environment is a difficult task:
- There is no absolute and rational way to prove the superiority of one engine type compared to others
- On the other hand some knowledge representation is more adapted to the type of information that one is going to work on.
- The environment of an expert system is of the highest importance, for the access to data banks, for the ability to use classical algorithms and for passing information to and from them as well as for the interface quality

3.5.4. The validation, precision and consistency

The verification of knowledge consistency is still done on an experimental scale; the definition of methods formalising the process of knowledge acquisition must significantly contribute to the betterment of expert system reliability, performance and development costs.

Precision of knowledge is quite difficult to measure. Yet, we would like to know the reasons for this imprecision (almost impossible to collect), the factors according to which this precision is modified, and the trend of the modification.
§ 3.5.4 Validation

The validation according to the methodology fig No.3.3. occurs at the end of the prototype phase. There is no theoretical fool-proof basis to this validation and not many methods exist to control and validate systematically the knowledge base obtained. It is all the more so that many applications did not get out of the research field or did not overcome yet the feasibility level. Yet validation of expert systems is an imperative prerequisite to the operational use of the tool.

Some techniques do exist to check the logical coherence of a text in linguistics but there is not enough communication between this field and the artificial intelligence one to allow the finalisation of efficient methods; The existing methods are too heavy in the case of an important knowledge base and anyhow logical coherence is no substitute for the exactness and the accuracy of the knowledge imbedded in it. To palliate this lack of precise validation procedure, different ways have been experimented.

The validation procedure should start with the comparison of the expert's conclusions and those driven from the expert system first on studied cases and then on different cases (if the expert is still available or if a set of examples has been left apart for this purpose; the knowledge imbedded in the knowledge base resulting from an initial and more limited set of examples). If disagreements appear, the level and the origin of the divergences should be identified (hypothesis, implicit considerations etc.) and necessary corrections made.

Second the confrontation of the expert with the rules evolved from his doing may be tried. This assumes that at first one has made sure that the expert (source of knowledge) agrees with all the conclusions given by the expert system. But even with this preliminary caution, when confronted with explicit rules, the expert may deny their relevancy or that they adjust to his thinking process. This does not imply that they should necessarily be altered or discarded; it may just be that the expert is not aware of his reasoning or that he feels that his reasoning is more shaded or complex that what has been shown in the rules. In the latter case, the discrepancies between the reasonings may have escaped detection because no observed cases have lead to the identification of these shades and more complex ramifications of the actual expert's reasoning.

Third confrontations of results and rules may be tried with other experts. Any discrepancy in reasonings or conclusions ought to be made clear but it will not necessarily invalidate the former reasonings.

A lot more research work should be done to work out proper methodologies to validate knowledge bases. Until then it has to be done on an experimental basis and a lot is left to the appraising and the common sense of those who develop expert systems.