Chapter 4

Knowledge Engineering

4.1. Introduction

Human knowledge and reasoning are symbol-dependent. Their expression and use are possible only through symbols. Computers are now capable of representing, processing and using human knowledge for reasoning and problem-solving. Creation of an ensemble of knowledge handling capabilities for problem-solving through computers is referred to as Knowledge Engineering (KE). KE is the process of acquiring, representing, organising, encoding, storing, processing and applying knowledge through computers. Its purpose is knowledge-based problem-solving in specified domains. Expert Systems (ES) require the use of domain related 'expert' knowledge for their solution. The expert knowledge may be in the form of facts, relationships, procedures, heuristics, beliefs, experience and insights. It may variously be formal and categorical, quantitative and qualitative, incomplete and imprecise, uncertain and judgmental. Knowledge may be collected from many sources. A representative list of sources includes domain experts, books, computer databases, maps, flow diagrams, pictures, web-sites etc. These sources can be categorised into two types: documented and undocumented sources.

Domain experts are generally considered as the primary sources of knowledge for an expert system development. Experts should have developed domain expertise by task performance over a long period of time. One of the objectives of the knowledge acquisition is to find the experts' heuristics related to the task. Project experts should have enough experience to have been able to develop the domain insights that result in these heuristics.

Experts should be capable of communicating their knowledge, judgement, and experience and the methods they use to apply these to the particular task. Experts' temperament, co-operativeness, and working relation with the project team can have a major impact on the success and the speed of the knowledge acquisition.

After the knowledge acquisition, this knowledge has to be put into an objective form for the knowledge base. The proper selection and design of a suitable knowledge representation scheme should be in tune with the requirements of the application domain. In addition, the proper selection should also depend on certain important properties of a scheme like expressive power and adequacy in context to the

application domain. In this chapter, we have tried to analyse some of these issues from the viewpoint of an expert system designer.

Section 4.2 contains the description of levels of knowledge. Knowledge categories are presented in section 4.3. In section 4.4, we describe the knowledge acquisition. Methods of knowledge acquisition are presented in section 4.5. Knowledge acquisition problems and possible ways of overcoming them are discussed in section 4.6. Section 4.7 contains the knowledge acquisition in the context of present work. Section 4.8 will be devoted to describe some knowledge representation schemes from the literature. Section 4.9 contains the object-oriented presentation in expert systems. In section 4.10, we analysed the relative suitability of such schemes as described in section 4.8. Finally, we end up with some discussion.

4.2. Levels of knowledge

Knowledge can be represented at different levels, of which two extremes are - shallow knowledge and deep knowledge. Shallow knowledges are the surface level information, that can be used to deal with very specific situations. Deep knowledge refers to the internal and casual structure of a system and considers the interactions among the system components. Deep knowledge can be applied to different tasks and different situations. It is based on a completely integrated, cohesive body of human consciousness that includes emotions, common sense, intuition etc.

4.3. Knowledge categories

Knowledge can be differentiated into various categories - such as declarative knowledge, procedural knowledge, semantic knowledge, episodic knowledge and meta-knowledge.

4.3.1. Declarative knowledge

Descriptive representation of knowledge is a declarative knowledge. It is expressed in a factual statement. Declarative knowledge is especially important in the initial stage of knowledge acquisition.

4.3.2. Procedural knowledge

It includes step-by-step sequences and how-to types of instructions, it may also include explanations.

4.3.3. Semantic knowledge

Semantic knowledge reflects cognitive structure that involves the use of the long term memory.

4.3.4. Episodic knowledge

Episodic knowledge is autobiographical, experimental information organized as a case or an episode.

4.3.5. Meta-knowledge

Meta-knowledge means knowledge about knowledge. In AI, meta-knowledge refers to the knowledge about the operation of knowledge based systems i. e., about its reasoning capabilities.

4.4. Knowledge acquisition

Knowledge acquisition is the process by which expert system developers find the knowledge that domain experts use to perform the task of interest. This knowledge is then implemented to form the ES programs. Therefore, once the domain has been selected, knowledge acquisition is very likely the most important task in an ES development. Acquisition of domain knowledge is a crucial phase of any knowledge engineering application and its weakness or failure can decisively block any future progress toward building an ES. Knowledge acquisition has to be systematic and comprehensive. To be useful, the knowledge must be accurate, presented at the right level for encoding, complete in the sense that all essential facts and rules are included, free of inconsistencies and so on.

4.4.1. Sources of knowledge

From many sources knowledge can be collected. We may classify them into two broad categories: (i) classical sources, and (ii) more recently available web-based sources.

4.4.1.1. Classical sources

A representative list of classical sources includes (i) domain experts, (ii) books and literature, films, computer databases, pictures, maps, flow diagram, etc. and (iii) real field case studies. Furthermore, these sources can be divided into two types: documented and undocumented knowledge. Undocumented knowledge resides in people's mind. Worthwhile to mention that in agriculture domain there are scopes of accumulating undocumented knowledge as gathered by field experts during their visit to the field. In this respect, domain experts might be considered as a good source of knowledge. Although there is a need for better methods of knowledge acquisition, including techniques to automate the process, but for the foreseeable future, most of the knowledge for any practical expert system in a complex domain will be obtained

through the interaction of knowledge engineers and domain experts [1]. Recently Internet and WWW can be used as another source of knowledge.

4.4.1.2. Web-based sources

With the introduction of Internet and World-Wide-Web (WWW) knowledge acquisition has got a new dimension. WWW has not only revolutionized the dissemination process of information but also it has created novel opportunities for sharing literal knowledge via Internet. Peoples are now getting acquaintance with web-based computer technologies. Knowledge engineers, working in their fields of interest can search and access the relevant web documents as a source of knowledge.

4.5. Methods of knowledge acquisition

Once the problem domain has been selected, knowledge acquisition is very likely the most important task in an expert system development. The elicitation of knowledge from the expert can be done by various ways. We may classify the methods of knowledge acquisition in three categories: manual, semiautomatic and automatic.

4.5.1. Manual methods

Manual methods are basically structured around some kind of interview. The knowledge engineer elicits knowledge from the domain expert and / or other sources and then codes it in the knowledge base. The three major manual methods are interviewing (structured, unstructured, semi-structured), tracking the reasoning process, and observing.

4.5.1.1. Interviewing

4.5.1.1.1. Structured interview

Systematic goal-oriented interview is a structured interview. It establishes an organized communication between the expert and the knowledge engineer. It allows the knowledge engineer to prevent the distortion caused by subjectivity of the domain expert and structured interview reduces the interpretation problem inherent in unstructured interview. Attention to a number of procedural issues are necessary for structuring an interview. To identify major demarcation of the relevant knowledge, the knowledge engineer should study available material(s) on the domain. During the knowledge acquisition session he / she should identify target question to be asked. Sample question, questioning techniques, question(s) type and level should be written prior to structured interview. Knowledge engineer should follow the guide lines for conducting interviews. During the interview the knowledge engineer uses directional control to retain the structure of interview.

4.5.1.1.2. Unstructured interview

As a starting point, informal interviews are conducted for many knowledge acquisition requirements. It helps to get quickly to the basic structure of the domain and saves time. According to McGraw and Harboson-Briggs [2], unstructured interviewing seldom provides complete or well organized descriptions of cognitive processes. Firstly, they observed that the expert system domains are generally complex; thus, the knowledge engineer and the expert must actively prepare for interview situations. Unstructured interviews generally lack the organization that would allow this preparation to transfer effectively to the interview itself. Second, domain expert usually find it very difficult to express some of the more important elements to their knowledge. Third, domain expert may interpret the lack of structure as requiring little preparation on their part prior to the interview. Fourth, data acquired from an unstructured interview are often unrelated, exist at varying levels of complexity, and are difficult for the knowledge engineer to review, interpret, and integrate. A fifth problem cited by McGraw and Harbison-Briggs concerns training. Because of a lack of training and experience, few knowledge engineers can conduct an efficient unstructured interview. Thus, knowledge engineers appear disorganized and may unwittingly allow the expert to have low confidence in the knowledge engineer. This may decrease the rapport needed to work together on a large scale development effort. Finally and most importantly, unstructured situations generally do not facilitate the acquisition of specific information for experts.

4.5.1.1.3. Semi-structured interview

Semi-structured interviews are obviously a compromise between structured approach and unstructured approach. This approach is required when complete unfolding of the complexity of the problem domain is not possible.

4.5.1.2. Tracking the reasoning process

Tracking the reasoning process refers to a set of techniques that attempts to track the reasoning process of an expert. Cognitive psychologists use this technique in discovering the expert's "train of thought" while he / she reaches a conclusion.

4.5.1.3. Observations

Sometimes, it may be possible to observe the expert at work and thereby one can acquire knowledge.

4.5.2. Semi-automatic methods

Methods intended to help the knowledge engineers by allowing them to execute the necessary tasks in a more efficient and / or effective manner and also intended to support the experts by allowing them to build knowledge bases with little or no help from knowledge engineers are semiautomatic methods.

4.5.3. Automatic methods

In this method the role of the expert is minimal (limited to validation) and there is no need for a knowledge engineer. For example, the induction method can be administered by any builder (e.g., a system analyst).

4.6. Problems in knowledge acquisition

There are a number of problems with knowledge acquisition [3] mainly concentrating on two aspects : (i) availability of source(s) and (ii) transferring knowledge. To overcome these problems many efforts have been made [4]. For example, researches on knowledge acquisition tools are going on [5] to focus on ways to decrease the representation mismatch between the human expert and the program under development. Several expert system development software packages such as EXSYS, Level5 and VP expert greatly simplify the syntax of the rules (in a rule-based system) to make them easier for an expert system builder to create and understand without special training. Also, a natural language processor can be used to translate knowledge.

In case of web based acquisition, any search engine results a large number of directories. It is reported that the number of web sites is increasing 10% in every month. It is sometimes very difficult to have a comprehensive list of the proper web sites of domain interest. Although one may get some good starting places. Some problems associated with web based knowledge acquisition are:

1. Lack of quality control at stage of production, leading more easily to lack of reliability and ultimately leading to a wrong concept.

2. It is possible to read a web page without having seen context pages or the cover page containing disclaimers, warnings etc.

3. Authors of web pages, news articles, e-mails, etc., sometimes remain unidentified.

4.7. Knowledge acquisition in the context of present work

Books, manuals or other written materials discussing the domain can form the basis of an initial but secondary sources of knowledge base. In such written material, an expert has already extracted and organized some of the domain expertise. As the problem domain is concerned with the insect pests and diseases of tea, so at the very initial stage of this work, the knowledge of domain parameters was elicited from various books on the subject, workshop reports and published papers on pests and diseases of tea as mentioned in Chapter 2. As Tea Research Association of India (TRA) has long years of experience and contribution in providing the research and development in tea domain, so a major part of the documents were collected from TRA.

The domain terminology, categories and jargon were extracted from the above said literatures and a glossary of pertinent domain terms was prepared. It provided a benefit on the subsequent steps of knowledge engineering.

The insect pests and diseases causing major damages were identified on the basis of estimated crop loss and previous reports of incidence. The possible set of damaged symptoms were recorded against each insect pest and diseases which in later stage were verified by the domain experts. The morphological characteristics of pests were studied from text books and publications of entomology. Various relevant photographs were collected from TRA and as well as from domain experts.

Domain experts are the primary sources of knowledge for an ES. Knowledge acquisition primarily refers to the process of eliciting the needed knowledge from domain experts. Knowledge acquisition is concerned with eliciting the facts, rules, patterns, heuristics, and operations used by experts to solve problems in a particular domain. In the present work, as the primary source of knowledge, four domain experts having 15 to 25 years of experience had been consulted through structured interviews. Forms were prepared to record the knowledge extracted from those experts. The experts were requested to give their judgement for different set of possible observations. The representative sources of domain experts used in this work are:

(i) Mr. R. Sinha, Senior Executive, Radharani T. E., West Bengal, India.
(ii) Dr. S. Sanigrahi, Advisor, Tea Research Association, Nagrakata, West Bengal, India.
(iii) Mr. D. Guha, Manager, Madhu T. E., West Bengal, India.
(iv) Mr. P. Aditya, Asstt. Manager, Mornai T. E., Assam, India.

As a third source of knowledge, regular visits were conducted to the pest and disease affected sections of various tea gardens of North Bengal districts of West Bengal, India

in different seasons. All physical observations were incorporated in the knowledge base of the present system.

4.8. Knowledge representation methods

A knowledge representation method is the way a knowledge engineer models the facts and relationships of the domain knowledge. The two types of knowledge that need to be represented in a computer are declarative knowledge and procedural knowledge. Declarative knowledge signifies facts about objects, events, and about how they relate to each other and procedural knowledge signifies the way to use the declarative knowledge.

In the present state-of-the-art of the subject there is no best theory of knowledge representation and so there is no best way to represent knowledge. Each method has its own advantages and disadvantages.

Several common knowledge representation schemes have been discussed in the literature [3, 6-10] including logic, semantic networks, production rules, frames, scripts, and OAV-triplets as classical methods; and the relatively new paradigm: object-oriented (O-O) approach.

4.8.1. Knowledge representation using logic

The application of logic as a practical means of representing and manipulating knowledge in a computer was not demonstrated until the early 1960s [11]. Since that time, numerous systems have been implemented with varying degrees of success. Logic is a formal method for reasoning. Many concepts which can be verbalized, can be translated into symbolic representation which closely approximate the meaning of these concepts. These symbolic structures can then be manipulated in programs to deduce various facts to carry out a form of automated reasoning. The logic process takes in some information called premises and produces some outputs called conclusions. Logic is basically classified into two categories: propositional logic and predicate logic (first order predicate logic).

Propositional logic (PL) is the simplest form of logic. Here all statements made are called propositions. A proposition in propositional logic takes only two values i.e. either the proposition is 'TRUE' or it is 'FALSE'. Consider the following statements:

Statement 1 : Rubber is a good conductor of electricity

Statement 2 : Diamond is a hard material

Both these statements are propositions, with the former having a value of FALSE and the later having a value of TRUE.

Valid statements or sentences in PL are determined according to the rules of propositional syntax. This syntax governs the combination of basic building blocks such as propositions and logical connectives. There are two kinds of propositions: atomic propositions (simple propositions) and molecular propositions (compound propositions). Molecular propositions are formed by combining two or more atomic propositions using a set of logical connectives 'not', 'and', 'or', 'if', 'then', and 'if and only if'.

For example, the following are compound formulas:

It is raining and the wind is blowing
If you study hard you will be rewarded
The sum of 10 and 20 is not 50.

The syntax of PL is defined recursively as follows:

P and Q are formulas.

If P and Q are formulas, the following are formulas:

$(\sim P)$
$(P \ \& \ Q)$
$(P \ V \ Q)$
$(P \rightarrow Q)$
$(P \leftrightarrow Q)$

All formulas are generated from a finite number of the above operations.

An example of a compound formula is

$$((P \ \& \ (\sim Q \ V \ R) \rightarrow (Q \rightarrow S))$$

The inference rules of PL provide the means to perform logical proofs or deductions. A key inference rule in PL is modus ponens.

Modus ponens: From P and $P \rightarrow Q$ infer Q

or

$$\frac{P \qquad P \rightarrow Q}{Q}$$

For example,

given: (joe is a father)
and (joe is a father) → (joe has a child)
conclude: (joe has a child).

PL works well in situations where the result is either TRUE or FALSE, but not both. However, there are many real life situations that can not be treated this way.

First Order Predicate Logic (FOPL) or predicate calculus has assumed one of the most important roles in AI for the representation of knowledge. In FOPL, statements from a natural language like English are translated into symbolic structures comprised of predicates, functions, variables, constants, qualifiers, and logical connectives. The symbols form the basic building blocks for the knowledge and their combination into valid structures are accomplished using the syntax for FOPL. Once the structures have been created to represent basic facts or procedures or other types of knowledge, inference rules may then be applied to compare, combine, and transform these 'assumed' structures into new 'deduced' structures. This is how automated reasoning or inferencing is performed.

For example, the statement "All employees of the AI Software Company are programmers" can be written in FOPL as

$$(\forall x)\ ((\text{AI-SOFTWARE-CO-EMP}(x) \rightarrow \text{PROGRAMMER}(x))$$

where x is a variable which can assume a person's name.

Now if Jim is an employee of AI Software Company, i.e. AI-SOFTWARE-CO-EMP(jim), then the conclusion is "Jim is a programmer" i.e. PROGRAMMER(jim).

The above suggests how knowledge in the form of English sentences can be translated into FOPL sentences. Once translated, such statements can be typed into a knowledge base and subsequently used in a program to perform inferencing.

Like PL, a key inference rule in FOPL is modus ponens. From the assertion "Leo is a lion" and the implication "all lions are ferocious" we can conclude that "Leo is ferocious".

Assertion : LION(leo)
Implication: $\forall x$ LION(x) → FEROCIOUS(x)
Conclusion: FEROCIOUS(leo).

In general, if 'a' has property P and all objects that have property P also have property Q, then the conclusion is a has property Q.

Modus ponens:

$$\frac{P(a) \qquad \forall x\, P(x) \rightarrow Q(x)}{Q(a)}$$

4.8.2. Knowledge representation using semantic networks

A semantic networks or semantic net is a structure for representing knowledge as a pattern of interconnected nodes and arcs. It was introduced by Quillian [12] to model the semantics of English sentences and words. Semantic nets provide a more natural way to map to and from natural language than do other representation schemes. Network representations give a pictorial representation of objects, their attributes and the relationship that exist between them and other entities. Semantic nets are directed graphs with labeled modes and arcs or arrows. The language used in constructing a network is based on selected domain primitives for objects and relations as well as some general primitives. The following rules about nodes and arcs generally apply to most of the semantic nets:

1. Nodes in the net represent either entities or attributes or states or events.

2. Arcs in the net give the relationship between the nodes and labels on the arc specify what type of relationship actually exists.

For example, a class of objects known as 'Bird' is depicted in fig. 4.1. The class has some properties and a specific number of class named 'Tweety' is shown. The colour of 'Tweety' is seen to be yellow.

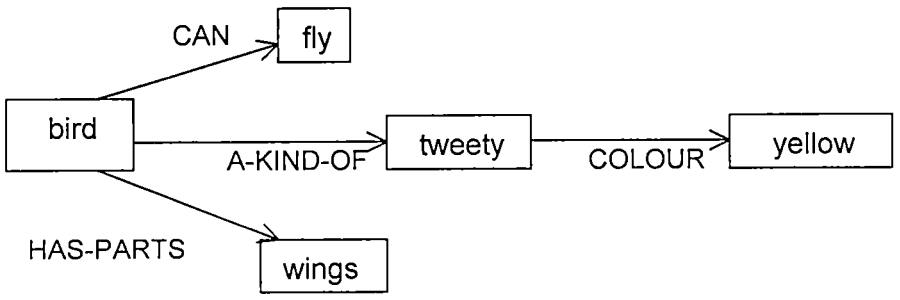

Fig. 4.1. Fragment of a semantic net.

The semantic nets have a certain intuitive appeal in that the related information is clustered and bound together through relational links. The task is typically contained within a narrow domain or semantic vicinity of the task. The graphical portrayal of knowledge can also be somewhat more expensive than other representation schemes.

4.8.3. Knowledge representation using rules

Rules provide a formal way of representing recommendations, directives, or strategies; they are often appropriate when the domain knowledge results from empirical associations developed through years of experience solving problems in an area. A rule has two parts. The first part is a premise of conditions connected by logical-AND or logical-OR relationships. The second part is a conclusion. When the premise of a rule is true, the conclusion of the rule will become true. In some systems rules may be implemented by semantic networks or OAV, as in MYCIN [13], the medical diagnostic system developed at Stanford University. Alternatively, rules may be represented by frames, as in IntelliCorp's knowledge engineering environment [14].

In expert systems jargon the term rule has a much narrower meaning than it does in ordinary language. It refers to the most popular type of knowledge representation technique, the rule-based representation. Rules are expressed as IF-THEN statements, as shown below [15]:

Rule 1. If a flammable liquid was spilled,
called the fire department.

Rule 2. If the pH of the spill is less than 6,
the spill material is an acid.

Rule 3. If the spill material is an acid,
and the spill smells like vinegar,
the spill material is acetic acid.

These are rules that might exist in a crisis management expert system for containing oil and chemical spills. Rules are sometimes written with arrows ($\rightarrow$) to indicate the IF and THEN portions of the rules.

Rule 2 in this notation would look like :

If the pH of the spill $\rightarrow$ the spill material
is less than 6 is an acid.

In a rule-based expert system, the domain knowledge is represented as sets of rules that are checked against a collection of facts or knowledge about the current situation.

When the IF portion of a rule is satisfied by the facts, the action specified by the THEN portion is performed. When this happens, the rule is said to fire or execute. A rule interpreter compares the IF portions of rules with facts and executes the rule whose IF portion matches the facts, as shown in fig. 4.2.

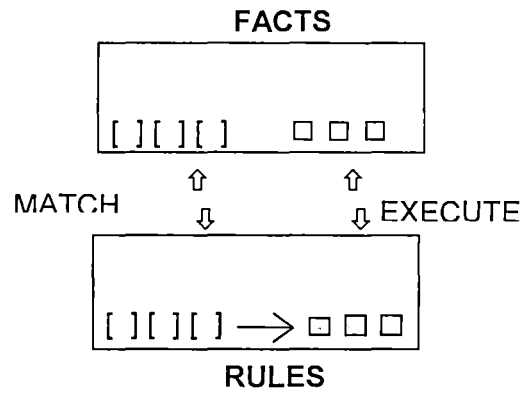

Fig. 4.2. The rule interpreter cycles through a match-execute sequence.

The rule's action may modify the set of facts in the knowledge base, for example, by adding a new fact, as shown in fig. 4.3. The new facts added to the knowledge base can themselves be used to form matches with the IF portion of rules as illustrated in fig. 4.4. The action taken when the rule fires may directly affect the real world, as shown in fig. 4.5.

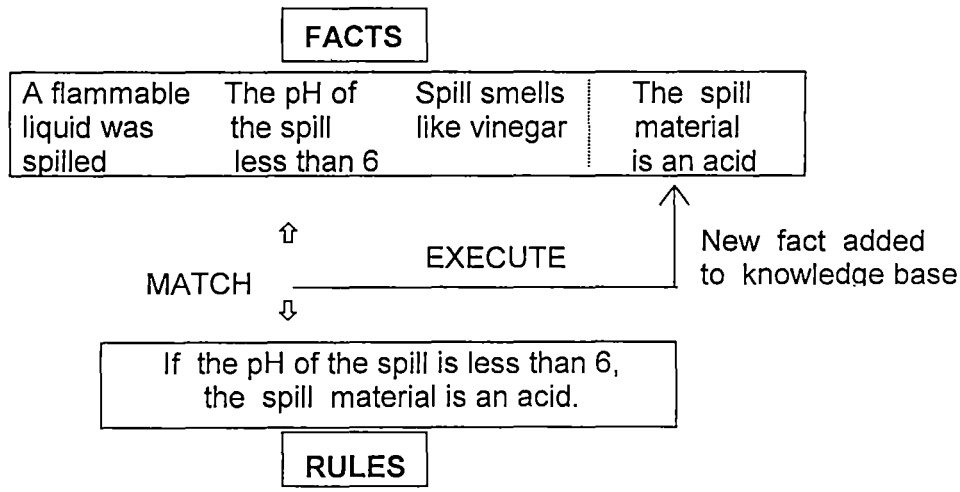

Fig. 4.3. Rule execution can modify the facts in the knowledge base.

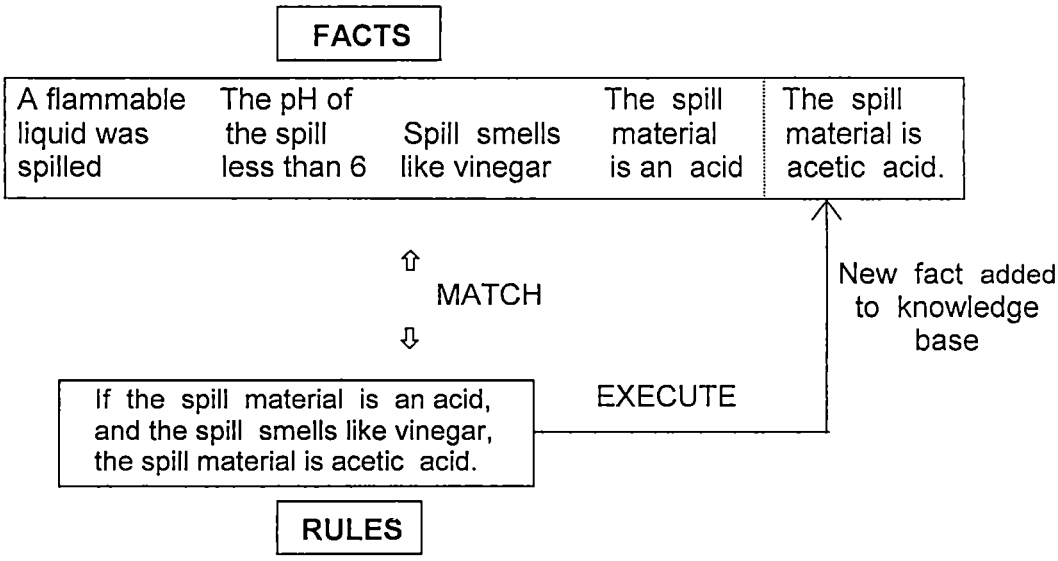

Fig. 4.4. Facts added by rules can match rules.

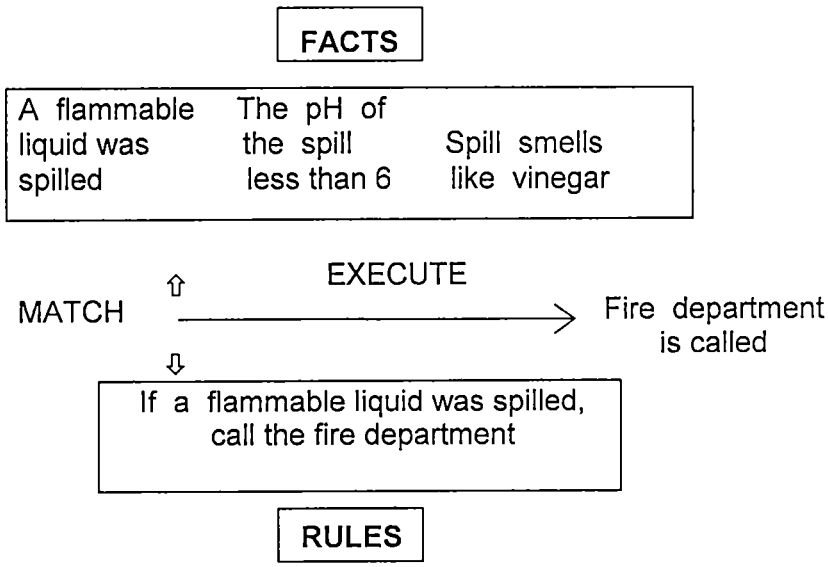

Fig. 4.5. Rule execution can affect the real world.

In this matching of rule, IF portions to the facts can produce what are called inference chains. The inference chain formed from successive execution of rules as shown in fig. 4.6. This inference chain indicates how the system used the rules to infer the identify of the spill material. An expert system's inference chains can be displayed to the user to help explain how the system reached its conclusions.

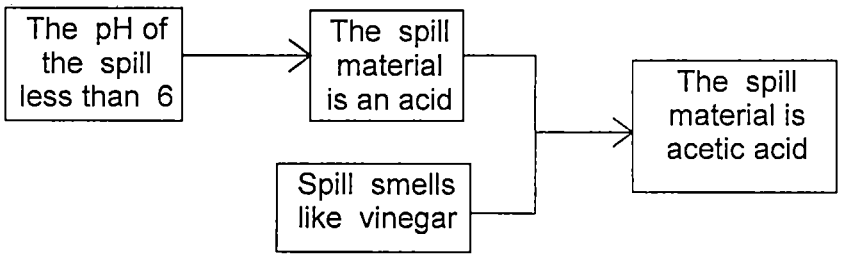

Fig. 4.6. Inference chain for inferring the spill material.

There are two important ways in which rules can be used in a rule-based expert system: forward chaining and backward chaining. The spill material example just presented used forward chaining.

Forward chaining is a 'data-driven' approach. In this approach one starts from available information as it comes in, or from a basic idea, then to draw conclusions. Backward chaining is a 'goal-driven' approach in which one starts from an expectation of what is to happen (hypothesis), then seek evidence that supports (or contradicts) his/her expectation. The inference chain created by backward chaining is identical to the one created by forward chaining; but however, the order and actual number of states searched can differ. The preferred strategy is determined by the properties of the problem itself. These include the complexity of rules, the shape of the state space, and the nature and availability of the problem data. All these vary for different problems.

4.8.4. Knowledge representation using frames

Minsky [16] introduced the concept of frames as a means of representing knowledge. He proposed that knowledge be organised into small packets called *frames*. Frames are general record-like structures which consist of a collection of slots and slot values. The slots may be of any size and type. Slots typically have names and values or subfields called facets. Facets may also have names and any number of values. A general frame template structure is shown in fig. 4.7.

(<frame name>
(<slot1> (<facet1><value1> . . . . .<valuek$_1$>)
(<facet2> <value1>. . . . .<valuek$_2$>)
.
.
(<slot2> (<facet1><value1> . . . . .<valuek$_m$>)
.
.)

Fig. 4.7. A general frame structure.

A frame structure provides facilities for describing objects, facts about situations and procedures on what to do when a situation is encountered. The frames are used to represent two types of knowledge viz. declarative and procedural. A frame that merely contains description about objects is called a declarative or factual or situation frame. On the other hand, it is possible to have procedural knowledge represented in a frame. If a frame holds knowledge of a procedure, it is termed as a procedural frame.

For example, the Ford frame illustrated in figure 4.8 has attribute value slots (COLOUR: silver, MODEL: 4-door), a slot which takes default values far GAS-MILEAGE and a slot with an attached if-needed procedure.

```
(ford  ( AKO (VALUE car))
       (COLOUR ( VALUE silver))
       (MODEL  (VALUE 4-door))
       (GAS-MILEAGE  (DEFAULT  fget))
       (RANGE  (VALUE  if-needed))
       (WEIGHT  (VALUE  2600))
       (FUEL-CAPACITY  (VALUE 18))
              .
              .
              .
```

Fig. 4.8. A Ford frame with various slot types.

The value fget in the GAS-MILEAGE slot is a function call to fetch a default value from another frame such as the general car frame for which Ford is a-kind-of (AKO). When the value of this slot is evaluated, the fget function is activated. When fget finds no value for gas mileage, it recursively looks for a value from ancestor frames until a value is found.

The if-needed value in the RANGE slot is a procedure name that, when called, computes the driving range of the Ford as a function of gas-mileage and fuel-capacity. Slots with attached procedures such as fget and if-needed are called procedural attachments or demons. They are done automatically when a value is needed but not provided for in a slot. Other types of demons include if-added and if-removed procedures. Like semantic nets, frames are usually linked together in a network through the use of special pointers such as the AKO.

Frame representations have become popular enough that special high level frame-based representation languages have been developed. Most of the languages use LISP as the host language. Several frame languages have now been developed to aid in building frame based systems. They include the Frame Representation Language (FRL) [17], Knowledge Representation Language (KRL) [18], and KLONE [19].

4.8.5. Knowledge representation using scripts

In the real world, one encounters typical stereotypic situations such as going to movies, shopping in a supermarket, visiting a dentist, or eating in a restaurant. A script is a frame-like knowledge representation structure which is intensively used for describing such stereotypic sequences of actions. It is a special case of frame structure and is intended for capturing situations in which behaviour is very stylised.

Scripts were proposed by Roger Schank [20] at Yale University. Scripts are composed of a series of slots that describe, in sequence, the events that we expect to take place in familiar situations. Just as the concept of frames is based on the assumption that we have a set of expectations about objects, the use of scripts assumes that we also expect certain sequences of events to occur in particular times and places. Visit to a restaurant is composed of a series of scenes, for example, an entering scene, a sitting scene, an ordering scene, an eating scene, a bill payment scene etc. Fig. 4.9 shows a script for entering scene.

```
SCRIPT     : RESTAURANT
SCENE      : ENTERING
        GO INTO THE RESTAURANT
        LOOK AT THE TABLES
        DECIDE WHERE TO SIT
        GO TO TABLE
        SIT.
```

Fig. 4.9. A script for entering in a restaurant.

Reasoning in a script begins with the creation of a partially filled script situation to meet the current situation. Next a known script which matches the current situation is recalled from memory. The script name, pre-conditions or other key words provide index values with which to search for the appropriate script. Inference is accomplished by filling in slots with inherited and default values that satisfy certain conditions.

Scripts have now been used in a number of language understanding systems, such as SAM, PAM, POLITICS, FRUMP, IPP, BORIS etc. at Yale University by Shank and his colleagues. They all used some form of script representation schemes.

4.8.6. Object-attribute-value triplets as KR scheme

The object-attribute-value triplets (OAV-triplets) method represent knowledge using three tuples : (O, A, V). O is the set of objects, which may be physical or conceptual entities. A represents the set of attributes characterizing the general properties associated with objects. V (values) specify the nature of the attributes.

The OAV method is actually a special form of semantic network. The relation between an object and an attribute is a has-a link, and the relation between an attribute and a value is an is-a link. The objects, attributes, and values of OAV are equivalent to the nodes in semantic networks. Knowledge can be divided into dynamic and static portions. The triplet values are the dynamic portion. These values may change, but the static portion (usually facts and rules) remains unchanged for different consultations.

An expert system stores data about real-world entities. In knowledge representation theory, the real-world entities are objects. Each object has one or more attributes or properties, and the attributes have values; for example,

Object	Attribute	Value
Car	Colour	Black
Smith	Fever	High

The object is Car, the attribute is Colour, and the value is Black. Another example, the object is Smith, the attribute is fever, and the value is High.

OAV is more structured than a semantic network. However, when the number of objects increases, an OAV system becomes difficult to manage.

4.8.7. Object-Oriented approach

The world consists of different objects. An object is an independent entity represented by some data and a set of operations (methods and capabilities) [21]. Therefore, an object can be used to represent a variety of knowledge. Knowledge (K) can be formally represented by three tuples, K = (C, I, A). C is a set of classes represented by class objects. I is a set of instances represented by instance objects. A is a set of attributes possessed by the classes and instances.

4.8.7.1. Classes

A class is a description of a group of similar instance objects [21]. It is a mold that determines the behavior of its instances. Each class has a unique name and a set of attributes that define the properties of the class. A class may be a sub-class of another class and may inherit properties from its parent class as discussed in sub-section 4.8.7.4.

4.8.7.2. Instance objects

Instance objects are members of classes. Their properties are defined by their parent classes. Each instance object consists of three sets of attributes :

(a) Name - the name of the object, which is unique in the system. It is used to identify the object.

(b) Class - the class that contains the object.

(c) Instance attributes - attributes belonging to the instance object. Some operations may be performed on these attributes. The behaviour of the object is determined by the values of these attributes.

4.8.7.3. Attributes and methods / operations

The property of an attribute is determined by its type and value. The type of an attribute is defined by its class, while the value may be defined in its class or its instance object. A set of generic attributes can be associated with the every object in a class, say furniture, for example. All furniture have a cost, dimensions, weight, location, and colour among many possible attributes.

Methods are a kind of attribute belonging to objects. They are used to represent capabilities, not to store data, and are defined in classes. Methods cannot be modified during consultations.

4.8.7.4. Inheritance

Properties of a class can be inherited from its parent's class. This feature permits factoring knowledge into a class hierarchy. Thus, it encourages modular design of knowledge. The system adopts the inheritance rules, which are similar to those in smalltalk [22], as follows :

(a) If class A inherits from class B, then the objects of class A support all operations supported by objects of class B.

(b) If class A inherits from class B, then class A's attributes are a superset of class B's attributes.

For example, Chair is a member of the class furniture. Chair inherits all attributes defined for the class. This concept is illustrated schematically in fig. 4.10.

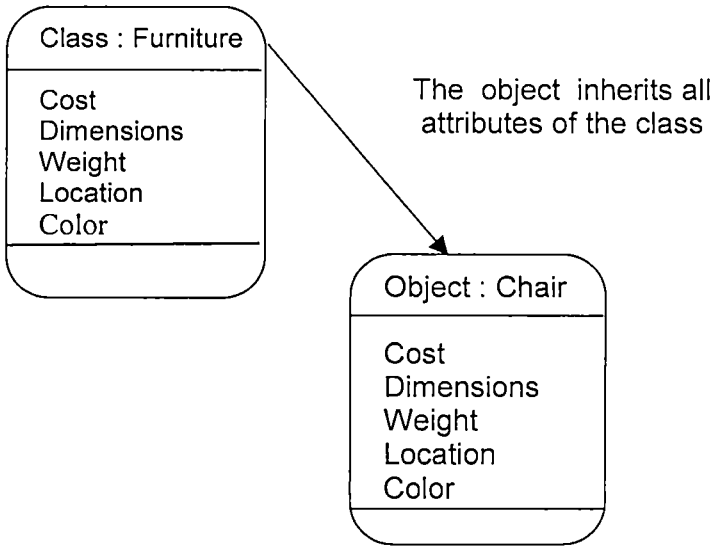

Fig. 4.10. Inheritance from class to object.

Once the class has been defined, the attributes can be reused when new instances of the class are created. For example, assume that we were to define a new object called table that is a member of the class furniture. Table inherits all of the attributes of furniture [22].

Every object in the class furniture can be manipulated in a variety of ways. It can be bought and sold, physically modified, or moved from one place to another. Each of these operations will modify one or more attributes of the object. For example, if the attribute location is actually a composite data item defined as:

Location = building + floor + room

Then an operation named move would modify one or more of the data items (building, floor, or room) that comprise the attribute location. To do this, move must have "knowledge" of these data items. The operation move could be used for a chair or a table, as long as both are instances of the class furniture. All valid operations (e.g. buy, sell, weigh) for the class furniture are "connected" to the object definition as shown in fig. 4.11 and are inherited by all instances of the class.

The object chair (and all objects in general) encapsulates data (the attribute values that define the chair), operations (the actions that are applied to change the attributes of chair), other objects (composite objects can be defined [23]), constants (set values),

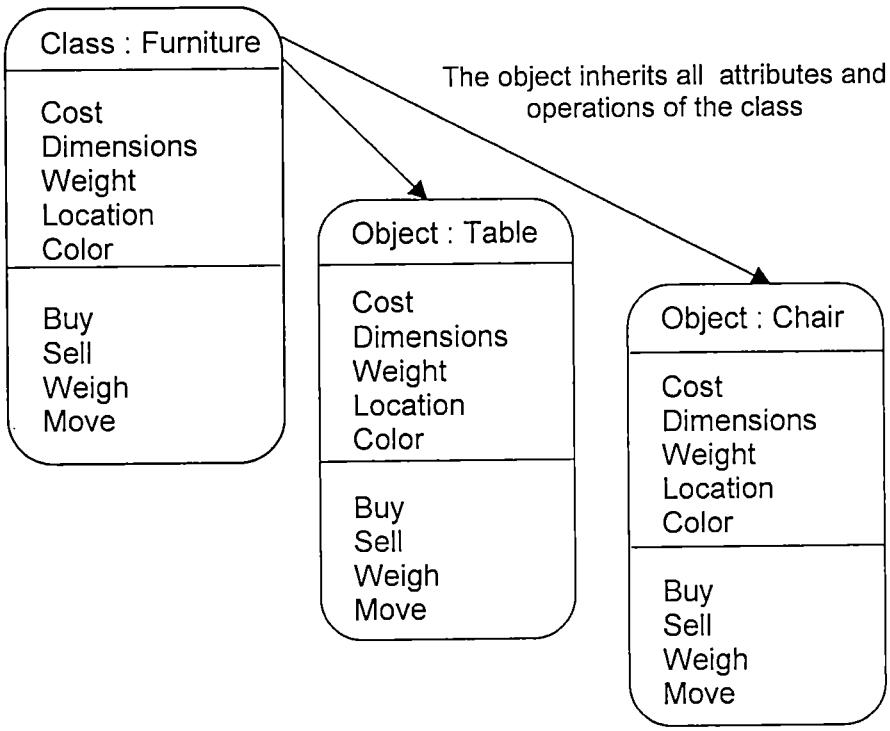

Fig. 4.11. Inheritance of operations from class to object.

and other related information. Encapsulation means that all of this information is packaged under one name and can be reused as one specification or program component.

4.9. AI, expert systems and O-O technology

In recent years, the term or adjective 'object-oriented' has become a popular slogan of any kind of systems regardless of its actual qualities. But however, we must admit that behind the slogan there must be some interesting ideas and concepts people suggest for developing large, integrated systems. Although there is the lack of standard definition of what is the O-O approach, but however, the properties like encapsulation, inheritance, polymorphism are considered useful for O-O approach. O-O is now being exploited for analysis and design, and people are sharing their experiences.

The ideas behind object oriented programming (OOP) and object oriented technology (OOT) date back to the forties [24]. These ideas, however, were not put into practice until the introduction of the Simula_67 programming language [25]. Simula, a superset of Algol, was designed for describing a wide class of discrete event simulations and implementing them for simulations. Simula objects represent data and an operation on the data. These objects communicate with each other through messages to determine

their next action. Although primitive by today's standards, Simula provided the first insight into the value of OOP.

The form of OOP we are accustomed to seeing took shape in the seventies with the development of Smalltalk at the Xerox Palo Alto Research Centre. Although Smalltalk is used to develop expert systems, its real value is that it offers a user-friendly programming environment.

What made Smalltalk easy to use and conceptually appealing was the extensive use of techniques commonly found today in OOP languages : class / object representations, inheritance, message-passing and encapsulation, to name a few. Researchers at Xerox Palo Alto Research Centre found that these techniques enabled a programmer to easily perceive an object system's structure and operation and to use this understanding to efficiently develop an interface, or for that matter, an entire functioning program. OOP's intuitive approach was the key to Smalltalk's success. Programming solutions frequently followed the methods that humans use to address everyday problems.

Given Smalltalk's intuitive programming environment, coupled with AI researchers' interests in computers representing and reasoning with knowledge similarly to humans, it was only natural for these researchers to adopt object-oriented techniques. This trend was most noticeable during the eighties.

One of the most important events during the eighties that spurred the interest in AI was the marketing of expert system development shells. Most of the early shells were rule-based. However, given the appeal of object-oriented systems, as demonstrated by Smalltalk's success, the demand pushed vendors to offer tools with object-oriented techniques. These tools, commonly called frame-based development programs (but sometimes called hybrid tools), usually combine object-oriented techniques with rule-based programming. New procedural languages with object-oriented techniques also surfaced, such as Objective C, C++, Pascal Object, Modula-2 and Lisp extensions such as Scoops, Flavors, Loops and the Common Lisp Object System (CLOS).

Armed with powerful object-oriented shells and languages, expert system developers took aim at problems that were often out of the reach of rule-based approaches. A review of systems developed during the later eighties and early nineties clearly shows a swing toward object-oriented techniques [26]. This trend was due partly to the availability of relatively inexpensive frame-based shells that ran on a variety of platforms. Two of the earliest frame-based shells, the knowledge engineering environment from IntelliCorp and the automated reasoning tool from inference, offered AI researchers powerful tools, but were costly and ran on mainframes or workstations, preventing their widespread use. In the mid-eighties, vendors began marketing cheaper

object tools, many of which ran on a PC. This situation led to the accelerated development of frame-based expert systems. Most important, it opened the door at most universities for teaching object-oriented technology (OOT) techniques to the next generation of AI researchers. Flourishing development of object-oriented knowledge-based systems continues. Vigorous development of object-oriented knowledge-based systems continues. Most corporations - including many in the Fortune 500 - are focusing on client-server and object-oriented problems. These organizations have come to recognize AI in general and OOT in specific, as a standard way of doing business. Whereas many of these companies first ventured into AI by forming a dedicated group of AI specialists, most of these specialists now work in the more traditional programming departments, where they routinely carry on their trade of knowledge-based programming.

A look at the recent marketing approach of vendors of AI object-oriented tools is also revealing. As any good business would do, these vendors have kept a finger in the air to sense the direction of their clients interests. They found that although terms such as "AI" and "expert systems" might have fallen out of favour in some circles, their clients' still wanted the object-oriented capability of their products. To go with the flow, these vendors began to advertise their products as "intelligent applications tools". AI capability was still there, but the idea of AI faded into the background. This presents an interesting situation : companies using AI but not promoting it, and vendors marketing products with AI capabilities but not advertising it. Although abandoning the AI label, both have created a new infrastructure on which to build the knowledge-based technology that should flourish in the latter part of the nineties. The irony : even if the spotlight is no longer on AI, AI's contributions will continue to positively affect future information processing, only under other labels [27].

4.10. Analysing relative suitability

The major advantage of semantic networks is flexibility, since new nodes and links can be defined as required without restriction. This flexibility also exists in object-oriented (O-O) knowledge representations where, by storing the names of their objects as the attributes of an instance object, relations between instance objects can be established dynamically. These relations have the same power as links in semantic networks; in fact, this object-oriented (O-O) construct can be viewed as dynamic semantic network. The is-a links of semantic networks can be implemented in object-oriented (O-O) representations by relationships between classes and sub-classes or between classes and instances. Has-a links can be implemented by the relationships between classes and attributes. Therefore, object-oriented knowledge representation has the same power as a semantic network but is much more structured.

A common disadvantage in semantic networks, rules, and OAV representations is that they are not structured enough. A significant increase in the number of objects or rules makes the system difficult to manage. This is because the knowledge cannot be modularized and interactions among rules and objects become too complex. When the value of an object or an attribute is modified, it is difficult to pinpoint the effects on the whole system. Therefore, such knowledge representations are difficult to develop and maintain, especially for a large knowledge base. The encapsulation property and structuredness of object-oriented (O-O) knowledge representations give them a distinct edge over these three representations.

Frames are more structured than semantic networks, rules, and OAV representations, since related attributes and rules can be grouped into frames hierarchically. However, modularity of knowledge represented in frames cannot be clearly defined, and frame representation lacks flexibility. In a frame system, relationships between frames may be member or subclass links and thus are not unique. Moreover, in some systems, a rule is represented by a frame linked to another frame with a special relationship. These factors greatly reduce the structure in a frame system. In object-oriented (O-O) knowledge representation, which is quite similar to frames, knowledge can be arranged in a hierarchical form using classes. However, a subclass link is the only possible relationship between two classes, an is-a link is the only possible relationship between a class and an instance object, and rules are defined as methods in classes - clear cut distinctions that reduce ambiguity and improve understandability.

In tune with our identified key requirements the domain lays on an expert system, we now analyse the relative suitability of different KR schemes discussed in section 4.8. When the domain knowledge is vast and varied, the knowledge can become unmanageable. To handle a large knowledge base it is suggested [28] that the structuredness and modularity is necessary where knowledge is varied. A common disadvantage in Semantic Networks, Rules and OAV representations is that they are not structured enough [10]. It is very difficult to manage a system with these representations when the number of objects and rules increases significantly. According to some researchers [29], some applications such as engineering processes, manufacturing and communications are expected to contain 100,000 rules or more. It is then very difficult to pinpoint the effects on the whole system if a value of an object or an attribute is modified.

The major advantage with semantic networks is its flexibility in defining new nodes and links as when required. The type of flexibility is also with the O-O approach which may be viewed as a dynamic semantic networks. The O-O knowledge representation has the same power as of semantic networks but is much more structured. Frames are more structured than Semantic nets, rules and OAV representations, since related attributes and rules can be grouped into frames hierarchically. This is a passive

data structure which lacks flexibility and the relationships within this system are not unique. The active data structure, the O-O representation of knowledge where declarative as well as procedural knowledge can be mixed, is structured and is much more meaningful semantically. The O-O form of KR encourages modular designs supporting the improvement of the efficiency of knowledge acquisition and management. The properties like encapsulation and inheritance of O-O approach are really attractive for large, integrated information systems. The encapsulation property prevents object manipulation except by defined operations. Inheritance is a valuable mechanism which enhances reusability and maintainability of software. Because this approach minimizes object interdependency [30] the knowledge can be structured.

A common disadvantage in OAV triplets and rules is that there may be some redundancy in information which may lead to some inconsistency. There is no such redundancy problem with Semantic Networks, frames and O-O forms. Moreover, O-O approach to KR supports high level of knowledge abstraction, an important advantage over other classical approaches.

Considering all these factors, we advocate O-O representation to improve consistency, understandability, maintainability and modifiability of knowledge base. Last, but not least, in the evolution of an expert system [15], prototyping may have an adverse impact on modifiability and maintainability of knowledge bases since these may be patched and modified several times. This may, however, be overcome by the use of O-O approach. As the system grows, the major changes will be with the addition of new objects or deletion of old objects rather than modifying the old objects. In this respect O-O approach is considered very useful for rapid prototyping, an added advantage.

4.11. Some ES and ES development tools using different KR-schemes

The following table 4.1 represents some expert systems and ES-developmental tools [15, 29,31,32] with the kind of knowledge representation scheme(s) and control for knowledge based scanning.

Table 4.1. Some ES/ES-tools using different KR-schemes.

ES/ES-tools	Representation	Control
ADVISOR II	Rule-based	Backward chaining
AI/RHEM	Rule-based	Forward chaining
ARBY	Rule-based	Backward chaining
ART	Rule-based, Frame-based	Forward and backward chaining
BABY	Rule-based	Forward chaining
CLOT	Rule-based	Backward chaining
CODES	Rule-based	Backward chaining

DELTA	Rule-based	Forward, backward chaining
DIPMETER ADVISOR	Rule-based	Forward chaining
DSCAS	Rule-based	Forward chaining
DUCK	Logic-based, Rule-based	Forward, backward chaining
EMYCIN	Rule-based	Restrictive backward chaining
ESE	Rule-based	Backward chaining
EXPERT	Rule-based	Forward chaining
EXSYS	Rule-based	Backward chaining
FAITH	Frame-based	Backward and forward chaining
FOLIO	Rule-based	Forward chaining
GOLDWORKS	Rules, Frames, Objects	Control over direction
GURU	Rule-based	Backward chaining, Limited forward chaining
GUSS/1	Rule-based	Backward and forward chaining
IMACS	Rule-based	Forward chaining
KES	Rule-based, Frame-based	Backward chaining
KES II	Rule-based, classes	Backward chaining, Limited forward chaining
LEONARDO	Rules, Frames, Procedures	Backward and forward chaining
LEVEL5 (OBJECT)	Large-hybrid-object-oriented, Rule-based	Forward and backward chaining
LISP	Procedure-oriented, functional, symbolic expressions	Forward, backward chaining
M.1	Rule-based	Backward chaining
MEDICO	Rule-based	Forward chaining
MES	Rule-based	Forward chaining
MI	Rule-based	Forward chaining
MUD	Rule-based	Forward chaining
MYCIN	Rule-based	Backward chaining
NEURAX	Rule-based	Forward, backward chaining
NEXPERT	Rule-based	Forward, backward chaining
ONCOCIN	Rule-based	Forward, backward chaining
OPS5	Rule-based	Forward chaining
PDS	Rule-based	Forward chaining
Plant/cd	Rule-based	Backward chaining
PROJCON	Rule-based	Backward chaining
PROLOG	Logic-based	Backward chaining
PTRANS	Rule-based	Forward chaining
PUFF	Rule-based	Backward chaining
RITA	Rule-based	Forward, backward chaining
S.1	Rule-based, Frame-based	Backward chaining

SAL	Rule-based	Forward chaining
SAVOIR	Rule-based	Backward and forward chaining
SPE	Rule-based	Forward chaining
SPERIL-I	Rule-based	Forward chaining
SPERIL-II	Rule-based	Forward and backward chaining
TALIB	Rule-based	Forward chaining
TATR	Rule-based	Forward chaining
TAXADVISOR	Rule-based	Backward chaining
THYROID MODEL	Rule-based	Forward chaining
WHEEZE	Frame-based	Backward and forward chaining
XCON	Rule-based	Forward chaining
XI PLUS	Rules, Induction	Control over direction
XSEL	Rule-based	Forward chaining
YES/MVS	Rule-based	Forward chaining

4.12. Object-oriented knowledge structure for tea pests and diseases

This work emphasizes the usefulness of O-O knowledge structure to improve consistency, understandability, maintainability, and modifiability of knowledge base. As an implementation tool, we have chosen object-oriented tool- Level5 Object. One can find the knowledge structure of "*TEAPEST*", - a rule based object-oriented expert system for insect pest management in tea (chapter 7) in Appendix A. In Appendix B, one can find the knowledge structure of "*TEADISEASE*", - a rule based object-oriented expert system for disease management in tea (chapter 8).

4.13. Discussions

In this chapter, we have considered the vital issues of knowledge engineering. Here we have tried explore the difficulties associated with knowledge acquisition. Potential sources used in this research have been pointed out. We have tried also to analyse the relative suitability of different KR schemes from the viewpoint of an expert system designer for the tea insect pest and disease domain. Our analysis finds object-oriented approach more suitable for the problem domain.

While above analysis and consequent results might lead one to believe that the object-oriented paradigm is a panacea for all the woes of knowledge engineering/ abstraction/representation, the paradigm does have some drawbacks [33]:

- One of object-oriented technology's disadvantage is its long learning curve. The classical developers have to devote several months before they are skilled enough to start a project.

- Second problem may be that is expected in the initial stages of any relatively new technology is the unavailability of robust and reliable tools such as AI-languages or a shell. However, at present, there are some ES-shells using O-O technology (e.g. Level5 Object) are coming into the market.

- The third problem may come from the very nature of abstraction. The reliability of the abstraction layer(s) should be sufficiently high so that there should not be any bug with these layer(s). These bugs are rarely trapped by the application layer due to the shielding property of abstraction. A careful design is, obviously, required to overcome this problem.

At this stage, there is no doubt that O-O technology should certainly assist us : (i) in developing a complex system; (ii) in maintaining the system; and (iii) in modifying the knowledge base of a system.

References

1. David S. Prerau. Developing and Managing Expert Systems. Addison-Wesley Publishing Company, Inc. 1990. pp. 200.

2. K. L . McGraw and B. K. Harbison-Briggs. Knowledge Acquisition, Principles and Guidelines. Englewood cliffs, N. J. Prentice-Hall, 1989.

3. Efraim Turban. Expert Systems and Applied Artificial Intelligence. Macmillan Publishing Company; New York. 1992.

4. J. H. Boose. A survey of Knowledge Acquisition Techniques and Tools. Knowledge Acquisition, 1. March 1989.

5. K. Bogel. Design and Implementations of a Web-based knowledge acquisition toolkit for medical expert consultation systems. Doctoral Thesis, Technical University of Vienna. 1997.

6. N. J. Nilsson. Principles of Artificial Intelligence. Narosa Publishing House, New Delhi. 1990.

7. Elaine Rich and Kevin Knight. Artificial Intelligence (2nd ed.). Tata McGraw-Hill, New Delhi. 1991.

8. Henry C. Mishkoff. Understanding Artificial Intelligence. H. W. Sams and Co. 1985. pp. 9-10.

9. Eugene Charniak and Drew McDermott. Introduction to Artificial Intelligence. Addison-Wesley. 1985.

10. K. S. Leung and M. H. Wong. An expert system shell using structured knowledge: An object-oriented approach. IEEE Computer; vol. 23, no.3. March 1990. pp. 38-47.

11. D. W. Patterson. Introduction to Artificial Intelligence and Expert Systems. Prentice-Hall of India, New Delhi. 2000. pp. 47-79.

12. M Quillian. Semantic Memory. In Semantic Information Processing. Ed. M. Minsky. Cambridge: MIT Press. 1968.

13. M. Van et al. Emycin manual. Tech. report. Heuristic programming project. Stanford University. 1981

14. F. R. Kehler. The role of frame-based representation in reasoning. Comm. ACM; vol.28, no.9. Sept. 1985. pp. 904-920.

15. D. A. Waterman. A Guide to expert systems. Addison-Wesley, Reading, Mass. 1986.

16. Marvin Minsky. A framework for representing knowledge. The Psychology of Computer Vision, (ed.), P. H. Winston. McGraw Hill, New York. 1975.

17. D. G. Babrow and T. Winograd. An Overview of KRL, a Knowledge Representation Language. Cognitive Science, 1(3). 1977.

18. I. P. Goldstein and R. B. Roberts. Nudge, a Knowledge Based Scheduling Program. In Proc. of the Fifth Int. Joint Conf. on Artificial Intelligence. 1977. pp. 257-263.

19. R. J. Brachman. A Structural Paradigm for Representing Knowledge. Report No. 3605. Bolt Beranek and Newman Inc. Cambridge, Mass. 1978.

20. Roger C. Schank and Peter G. Childers. The Cognitive Computer. Addison-Wesley. 1984.

21. A. Goldberg and D. Robson. Smalltalk-80 : The language and its Implementation. Addison-Wesley; NY. 1983.

22. R. S. Pressman. Software Engineering : A practitioner's approach. 3rd edn., McGraw-Hill; Inc. 1992.

23. Object-Oriented Requirements Analysis (course notebook). EVB Software Engineering. 1989.

24. K. Nygaard. Basic concepts in object-oriented programming. Sigplan Notices; vol.21, no. 10, October 1986. pp. 117.

25. O. Dahl and K. Nygaard. Simula - A language for programming and description of discrete event systems. Introduction and Users Manual. Norwegian Computing Centre, Oslo, Norway. 1967.

26. J. Durkin. Expert Systems : Catalog of Applications. Intelligent Computer Systems; Inc., Akron, Ohio, 1993.

27. J. Durkin. Expert Systems : A view of the field. IEEE Expert. April 1996. pp. 56-63.

28. W. B. Gevarter, The nature and evaluation of commercial expert system building tools. IEEE Computer; vol.20 No.5. 1987. pp. 24-41.

29. F. Hayes-Roth. Invited Talk. IEEE Compcon; San Francisco, CA. February 1987.

30. A. Synder. Encapsulation and inheritance in object-oriented programming languages. Proc. OOPSLA, ACM; New York. 1986. pp. 38-45.

31. W. B. Gevatter. The nature and evaluation of commercial expert system building tools. IEEE Computer, vol. 20, No. 5, 1987. pp. 24-41.

32. James L. Alty. Expert System Building Tools. Topics in Expert System Design. G. Giuda and C. Tasso,(ed), Elsevier Science Publishers B. V., North Holand, 1989. pp. 193.

33. R. Gupta, W. H. Cheng, R. Gupta, I. Hardonag and M. A. Breuer. An object-oriented VLSI CAD Framework: A case study in rapid prototyping. IEEE Computer, May 1989, pp. 28-37.