DEVELOPMENT OF A STRUCTURED METHOD FOR KNOWLEDGE ELICITATION

Gail Swaffield

Thesis submitted to the Council for National Academic Awards in partial fulfilment of the requirements for the Degree of Doctor of Philosophy

Thames Polytechnic
Wellington Street
London SE18 6PF

In collaboration with

Logica Financial Systems Ltd.
64 Newman Street
London W1A 4SE

January 1990

DEVELOPMENT OF A STRUCTURED METHOD FOR KNOWLEDGE ELICITATION

Gail Swaffield

ABSTRACT

The subject of this thesis is, broadly, knowledge elicitation for knowledge-based, or expert systems. The aims of the research were to investigate the transference of techniques of systems analysis to the knowledge elicitation process, and in so doing, to develop a structured method for knowledge elicitation.

The main contributions to the area of knowledge elicitation made by the research are:

i) The development of a method which has as a central part of it, the definition of an explicitness boundary, across which and within which all data and processes must be explicit. It is argued that in order to be explicit, the data must be in the form of limited data sets as opposed to continuous data.

ii) The development of a method which forces the use of an intermediate representation, thus forcing a logical / physical design split, as in systems analysis for conventional data processing systems.

iii) The concern for user independence in the resulting systems. The ability to increase user independence is enhanced by the use of limited data sets, and also by the involvement of designated users of the expert system, and testing of the intermediate representation, during knowledge elicitation.

The starting point of the research is the lack of methods for knowledge elicitation, and the pitfalls of existing techniques. Many of the techniques to have emerged from other disciplines such as cognitive psychology are discussed with respect to the concerns of this thesis, and the proposed method.

The specific techniques from systems analysis which are applied to knowledge elicitation are data flow analysis, entity-relationship analysis, and entity life cycle modelling. These three techniques form the framework of the method, which starts with a high-level analysis of the domain, and results in an implementation independent representation of the expert domain, equivalent to a logical model in systems analysis and design.

The final part of the thesis shows the ease with which the resulting model is translated to two of the most commonly used knowledge representation schemes - production systems and frames.

ACKNOWLEDGEMENTS

Above all, I would like to thank my primary supervisor, Dr. Brian Knight for his support, encouragement and ideas throughout this research. In addition I would like to acknowledge the help of Dr. Mark Cross at many times throughout the project. Logica Financial Systems Ltd. and the ARIES Club provided me with a project on which to try out and expand my ideas; in particular I would like to thank Richard Lelliott for his involvement.

There are many other people who deserve thanks for their support over the last few years.

First, my family, who have continually supported me in every way throughout my career so far. Second, my friends and colleagues both at Thames Polytechnic, and elsewhere without whom life would have been much more difficult at times. Third, Ernst & Young, my current employers, who have provided me with the resources to produce this thesis and my colleagues there who have encouraged me to finish the research.

Last, but by no means least, I would like to thank Ian Graham, who has both kept me going and put up with my single-mindedness for many months.

CONTENTS

CHAPTER 1 INTRODUCTION

1.1 The Problem

The aim of the research undertaken for this thesis was to develop a method of knowledge elicitation. Elicitation in this context is defined as the extraction of knowledge directly from experts, and the formulation of this expertise into a machine independent representation. The motivation for the work falls into the following three categories.

1.11 Available Methods

Currently, the literature emerging from the expert systems 'world' suffers from a lack of techniques for knowledge elicitation from experts. Descriptions of systems developed rarely cover in any depth the approach taken to the extraction and formalisation of the expert knowledge, concentrating more on the implementation methods, and the final performance of the system.

Existing tools and techniques (as discussed in chapter 2) tend to cover only isolated aspects of the elicitation process as opposed to approaching the problem in a more structured, rigorous fashion. These approaches concentrate in particular on the elicitation of functions, which for certain domains may be suitable, but for those domains where the data is an important part of the knowledge, a more data-oriented approach to elicitation may be more appropriate.

1.12 User Involvement

A major difference between expert systems and conventional data processing systems is concerned with the type of knowledge encoded into the systems and therefore the level of ability required to use them with success. Techniques for knowledge elicitation have tended to ignore the presence of 'users', where these users may differ from the experts as regards their level of ability to carry out the expert task.

The production of user independent systems, i.e. systems which produce the same answer to a problem regardless of the user, must rely not only on the ability of the user, but also on the level at which the system is pitched. The involvement of the end users, particularly in the early stages of knowledge elicitation is therefore an important aspect of system development.

1.13 Use of Methods from Systems Analysis

Years of research and development have produced and improved methods of systems analysis in order to enable the development of robust, usable computer systems. Although expert systems differ from conventional data processing systems in many ways, e.g. the type of knowledge encoded, method of implementation etc., the need to carry out an analysis of existing knowledge is present in the development process of both types of system.

A strong motivation for this research is to investigate the possibilities of transferring structured methods from systems analysis to the knowledge elicitation process. The nature of structured methods would then suggest a data-oriented approach to knowledge elicitation as opposed to the functional approaches often taken and would also encourage the involvement of users in this early stage of system development.

In summary, the aims of this thesis can be outlined as follows:

* Investigate the suitability of structured systems analysis techniques for the knowledge elicitation process.

* Develop a method of knowledge elicitation which achieves the aims outlined above i.e. is structured, data oriented and ensures user involvement.

In the following sub-sections of this chapter, firstly the aims and problems associated with knowledge elicitation are discussed and secondly the need for structured techniques in systems analysis. The final section introduces the approach taken to developing a method of knowledge elicitation, and thus outlines the thesis.

1.2 Knowledge Elicitation

1.21 Aims

The overall aim of the knowledge elicitation stage of expert systems development is to extract expertise from experts in order to incorporate it into the knowledge base of the system. The main stages of knowledge based systems development have been outlined [Hayes-Roth, Waterman & Lenat, 1983] as :

* Identification - outlining the process to be automated, the boundaries of the system, finding the domain experts and setting the scene for future stages.

* Conceptualisation - defining the main concepts and relations within the domain of expertise with which to represent the knowledge of the expert, and specifying subtasks to be analysed.

* Formalisation - obtaining the structure of the knowledge in order to build a framework on which to build the knowledge base using a formal representation preferably suggested by the expert.

* Implementation - building on the framework developed by adding the functional knowledge i.e. the 'rules' used by the expert for problem solving.

* Testing - evaluation of the performance of the implemented prototype system, and subsequent revision.

From the above it can be seen that expert systems development, like conventional systems, is an iterative process, and therefore knowledge elicitation is a continuous task throughout the development cycle. The main concentration on elicitation is however in the three middle stages, i.e. conceptualisation, formalisation and implementation. For the physical implementation and testing stages further elicitation may be required in order to refine the system and improve performance.

Buchanan and Shortliffe [1985a] highlight the conceptualisation and formalisation stages as the most difficult aspects of the process, in order to provide the framework on which to build the knowledge base. This is discussed further in the next sub-section (1.22) in terms of the problems of selecting the method of formalisation and representation of the expert knowledge.

Expert systems development and in particular the knowledge elicitation process can be likened to systems analysis for conventional data processing applications. The latter has however been considered to be a simpler process [Hart, 1986] probably due to the complexity of the problems solved by experts and the type of knowledge to be uncovered. It is not clear however, how much of the difficulty in knowledge elicitation is due to a lack of suitable tools for the task. As discussed in 1.3, the task of developing data processing systems was made easier and the resulting systems were more robust when attention was paid to the analysis stage, and structured methods developed. In the last 10 - 15 years many tools and techniques have been developed which make the task of systems analysis more flexible, providing tools for conceptual modelling, analysis of data and functions independently etc. It is maybe the case that similar sets of tools need to be devised for the knowledge elicitation process.

1.22 Problems

In addition to the problem of a lack of tools and techniques for knowledge elicitation, there are a number of inherent problems which can be defined in general terms as :

* form of the knowledge to be elicited,

* selection of formalisation and representation techniques,

* usability of the resulting system.

1.21.1 Form of the Knowledge

Knowledge or expertise is not always clear cut and objective and may also suffer from an element of subjectivity. If a scale of subjectivity is considered, those skills which lie at the completely subjective end will almost definitely be unsuitable for incorporation into an expert system. At this end of the scale, the expert reaches decisions almost entirely using intuition or 'automatic reaction' which will be almost impossible to capture in terms of 'rules'. At the opposite end of the scale problem solving can be considered to be almost completely 'algorithmic' in that a simple decision tree will be sufficient to model the decision making process. The majority of expert tasks, for which expert systems are developed lie somewhere between these two extremes of the scale and incorporate elements of both types of knowledge. The subjective element of knowledge therefore makes the elicitation process more complex than analysis of a non-expert task.

In addition, the expert may only reach decisions, or hold beliefs with a degree of certainty. This too is related to the subjectivity scale, as a purely objective decision as modelled by a flowchart or decision tree will involve only certain data, yes-no decisions etc. The problem with uncertain data or decisions is mainly how they should be modelled. Popular approaches are to use certainty factors and/or probabilities. Studies in cognitive psychology [Fhaner, 1977] have indicated strongly however, that human ability to express judgement in terms of probabilities is very limited. Not only will the expert have problems in articulating his uncertainty therefore, but the end-users of the system may be faced with the necessity to make such judgements while using the system. The opposite approach of using 'labels' or 'tokens' to represent uncertain values is one which has been considered and used but poses a different problem of the meaning attached to the labels i.e. different experts may attach a different meaning to the same label.

The problem of articulation of uncertain knowledge can be generalised to all aspects of the decision-making process i.e. the input data, reasoning process and the decision reached. Many experts find difficulties in expressing their knowledge and in particular in verbalising it.

Experiments which studied the relationship between task performance and the associated verbalisable knowledge [Berry & Broadbent, 1984] reached the conclusions that having verbalised knowledge about how a task was performed did not help the person to improve his performance of the same task. This would suggest that the task-related knowledge is for some reason not easily accessible (to the person concerned) in terms of verbal expression. Such findings would tally with the everyday occurrence of a person being able to complete successfully a (mental) task, without then being able to specify exactly how they did so, e.g. mental arithmetic or spelling previously unseen words.

The ability to communicate knowledge varies depending on whether the two participants in the exchange or transfer are both conversant in the same language. An expert may therefore have difficulty in communicating his knowledge to a knowledge engineer who has no knowledge or experience in the domain. On the other hand, a knowledge engineer who is also an expert could cause problems in trying to include his own expertise as part of that of the expert involved.

Lastly, expert knowledge like any other type of knowledge can consist of many different types of knowledge [Gammack & Young, 1984] which may require different methods of elicitation. Attention therefore needs to be paid to the kinds of knowledge e.g. concepts and relations, procedures, facts and heuristics, and classificatory knowledge which exist within the domain, before knowledge elicitation methods are applied.

1.21.2 Formalisation and Representation Techniques

In expert systems development there is a temptation to code the knowledge immediately into 'rules' and then to refine these rules as the resulting prototype is evaluated against test cases. If lessons are to be learnt from systems analysis however, there is much to be said for splitting the development into two design stages - logical and physical. Such an approach would aid the knowledge engineer in that the representation i.e. rules, frames etc. need not be decided at an early stage and the restriction of having to fit the knowledge into the structure provided by a particular toolkit or shell will not arise.

There is a still a need, however, to formalise the knowledge elicited from the expert, i.e. in the form of a logical model, thus requiring a logical modelling approach to be taken. The suggestion of Hayes-Roth et al. that the expert should select the method of formalisation may be applicable in some circumstances, but in others a more structured approach such as that taken in systems analysis may be more appropriate, where the formalisation technique is an integral part of the approach. Whichever method is used, the desirable result is a machine and implementation independent representation of the expert knowledge.

The task of selecting the representation of the knowledge for incorporation into the implemented physical system should then be a task of matching the type of knowledge to the most suitable representation, then selecting the most suitable shell or toolkit to achieve this aim.

1.21.3 Usability of the Resulting System

There is an obvious necessity that any system should contain the correct knowledge [Suwa, Scott & Shortliffe, 1985] in order for it to supply the users with the correct advice and solutions to their problems. This correctness should be in terms of both what is required of the system and how the problem is solved.

Problems occurred with the usability of MYCIN [Buchanan & Shortliffe, 1985b] due to the problems of integrating the system within the environment, and recognition of the need for the system. Even though steps were taken to make the system more usable in terms of explanation and HELP features, significant barriers still prevented widespread or routine use of the system. Had more attention been paid to the human factors element of the system, e.g. more control given to the user as opposed to the system controlling the dialogue and enabling only specific 1-2 word answers, these problems of usability may not have been so great.

Mumford [1985] advocates the participation of the users in expert system development as for data processing systems, suggesting that the users should be involved from the early stages

of the project. The assumption in this paper seems to be however that users are all experts, which may not always be the case. The extent and type of involvement of users must therefore be tailored to the type and level of end-user of the system.

1.3 Structured Systems Analysis

In the 1960's many data processing projects were found to be unsuccessful due to bad user-analyst relationships [de Marco, 1978]. The users were not involved in the early stages of system development and design but were forced to participate and comply with the analysts and further, were expected to use the resulting systems. The recognition of this problem was further highlighted in the 70's by Mumford [1978] who recommended that users should be involved almost to the degree of designing the systems themselves. Thus the idea of participative design came about whereby all users likely to be involved in or affected by the new system should be involved in its design.

In order to involve users in the design of computer systems however, a method of communication was found to be necessary i.e. a common language between computer-literate analysts and users. Existing methods of systems analysis relied on English narrative and flowcharts to describe the system to the users. The former had the tendency to be vague and long-winded and the same piece of narrative could be summarised in a fairly simple 'picture'. Gane and Sarson [1980] use the analogy of describing a house in terms of dimensions, doors to other rooms, positions of windows etc. instead of the use of a plan of the house and possibly visits to other similar houses. Flowcharts are no more successful however as they rely on knowledge of the physical design of the system i.e. how it is to be done rather than purely what needs to be done.

Such problems are the job of the system designer, not the analyst, and are not relevant to the user who wants to know whether the system will do what is required rather than how it will do it.

It became obvious that in order to overcome the above problems, a common language should take the form of graphical tools [de Marco 1978, Gane and Sarson 1977, 1980] which would aid

* the description of the problem by the user,
* definition and design by the analyst,
* communication to the programmer.

Also in the 70's, when database technology became more widely used, the need for the analyst's design to be logical was recognised i.e. there should be a definite logical/physical split between analysis/design, and design/implementation [Peters 1981, Gane and Sarson 1977, 1980]. The need for this split was due partly to the use of database technology in order to achieve independence from the physical implementation. The latter was desirable in order both to avoid problems caused by selection of the Database Management System (DBMS) early, and to lessen the problems caused by translating or converting to a different DBMS at a later stage.

The logical/physical split was also however found to be advantageous in terms of maintenance of the resulting system - the cost of error correction increases dramatically from analysis/design to coding and then implementation. At the stage of analysing requirements, the unit cost of error correction has been found by Boehm [Gane & Sarson 1977] to be 0.2, whereas the unit cost after implementation and testing is possibly greater than 10. More concentration on the design of the system was therefore thought to be the key to the reduction of the cost of error correction due to finding the errors early in system development.

From these problems in data processing in the 60's and 70's, and the recognition of what was required to prevent the resultingly unsuccessful projects, the definition of a structured method emerged :

* graphical tools should be used as a method of communication for logical design,

* there should be a logical/physical design split,

* there should be a high degree of user-analyst interaction at the logical design stage.

1.31 The Data Flow Method

The data flow method [de Marco 1978, Gane and Sarson 1977] is based on the principle of "transform analysis" [Yourdon and Constantine, 1979] in which the analysis concentrates primarily on the transformations which take place to the data within the environment under study. In this way, the technique bears similarity to the use of Petri nets for modelling resource scheduling problems in operating systems, where 'transitions' can only be fired by the arrival of the required number of 'tokens'.

Another major principle of the method is one more general to structured methods, i.e. the combination of the preliminary and detailed design stages into one phase, thus putting a different emphasis on the analysis. The method of structured analysis developed by de Marco stresses the following objectives of the structured analysis phase :

* move codification from design to analysis, in the form of the structured specification,

* use graphics of codification tools for user communication,

* remove redundancy from the structured specification,

* remove narrative text,

* remove physical information i.e. the structured specification should be purely logical.

The resulting structured specification consists of the following three parts :

* Data Flow Diagrams (DFD's) - show the logical flow of data through the processes which exist in the environment,

* Data Dictionary (DD) - documents the elements of the DFD's i.e. data flows, data stores and processes,

* Process descriptions (PD's) - also called mini-specs, describe the processes on the DFD's which transform the data from one form to another, using such methods as structured English, decision tables, decision trees etc.

1.32 Discussion of the Data Flow Method

The data flow method achieves all of the aims of a structured method in the following ways :

* the DFD's provide a graphical tool for user-analyst interaction, communication between analysts etc.,

* the whole process results in a structured specification which is a logical model of the system,

* the nature of the method ensures a high level of user-analyst interaction in order to produce the structured specification.

In addition, the methods of partitioning the system into sub-systems produces a highly modular approach to design giving all the advantages of hierarchical top-down design in addition to a straightforward method of identifying the automation boundary (i.e. the boundaries between those parts of the system which are to be automated by a computer system, and those which are not). The data dictionary provides rigorous documentation of the elements of the DFD and the PD's specify logically what is required of the system in terms of functions. The resulting structured specification is therefore complete in terms of the logical view of the system, and thus design and implementation of the physical system becomes a quicker and more straightforward process.

There are however certain disadvantages of the data flow method, related mostly to the functional approach taken to analysis.

Although the method, partly due to its name would appear to concentrate on the data within the 'system', the nature of the DFD's results in a functional approach. Data is only considered in terms of the transformations i.e. functions which occur i.e. how it flows within the system.

Consequently, the basis of the analysis is the set of functions within the system, which are not the most stable element of the environment - i.e. given the same data, different functions could be performed on it, but not vice versa. Concentrating on the functions rather than the data therefore results in little knowledge of the data itself, apart from its transformations, or how data elements relate to each other.

Although designed not to show control, the method inherently must show an element of control, as the flow of data within a system is part of the control knowledge of that system. The fact that process B requires data output from process A, whether the processes are decoupled by the insertion of a data store or not, means that within the same time cycle, process B must follow process A. The data flow method does not therefore remain independent of control knowledge.

Taking both of the above issues into account, the data flow approach seems to be neither wholly data-oriented or wholly function-oriented. The method of drawing DFD's is an attempt at data definition but the inclusion of functions destroys the purity of this approach. The attempt however to ignore the issues of control and sequencing of functions results in only partial treatment of the functions at the analysis stage.

Zimmerman [1983] describes the data flow method as a top-down functional approach to analysis. In this context it can be seen that the overall strength of the data flow method lies in the ability to identify the automation boundary and partition the system into sub-systems, allowing independent treatment of each of the resulting modules. As an initial tool for analysis the data flow method is therefore a valuable one but it does not allow definition of the structure and meaning of the data in the business environment.

1.33 The Data Structure Method

It is often stated that data is the fundamental building block of organisations and thus should be the basis of modelling the organisation. Further, the data within a system/organisation is

liable to be more stable than the functions, thus providing a more robust model of the organisation independent of its functions.

In general, the notion of a data model is one of an abstraction device i.e. it provides a method of concentrating on the general common properties of a set of objects within the business environment, whilst hiding the detail. A data model should thus convey the information content of the data as opposed to the individual values or occurrences of the data.

Within the data processing field, Data Structure Diagrams (DSD's) [Bachman, 1969] formed the first data models, introduced as a graphical technique of representing and communicating information structures at an abstracted level.

These DSD's concentrated on the following structures :

* entity class i.e. a group of similar entities (objects),

* entity set i.e. a group of entities in one class associated with one entity of a different class,

* set class i.e. a group of entity sets similar in terms of attributes to be considered collectively.

Two entity classes may be related therefore by means of a set class which indicates an owner-member relationship. As an example, 'department' and 'employee' may be entity classes for which a number of single entities exist. An entity set will exist between department and employee in that a group of employee entities will be members of one entity from the owner entity class department. A number of such relationships will however exist, thus forming a set class relationship between department and employee.

Such diagrams were intended to be used to represent two types of data or information structure i.e. hierarchical and network, or a combination of the two. In addition, relationships or set classes could be defined as 'dependent', 'dominant', 1-many, and 'sometime;member' i.e. optional.

Hierarchical and network data models both suffer, however, from the restriction of functional links [Tsichritzis & Lochovsky, 1982] which enforce the condition whereby a member entity class can have at most one owner entity class. Many-many relationships between entity classes cannot therefore be directly represented. In addition, recursive links i.e. relationships between entity classes where the owner and member are of the same type, cannot be represented. The latter restriction is due to the difficulties involved in navigating the resulting database due to the ambiguity caused by a record type being both an owner and a member.

The hierarchical model suffers from an extra restriction in that the structure must be that of an ordered tree where the functional links are always directed towards the leaves of the tree. The representation of many-many relationships therefore requires either duplication of records in the database or the existence of logical links and records as in the IMS DBMS.

Soon after Bachman, in the 1970's Chen [1976] had the idea of unifying existing data models (ie hierarchical, network and relational) into one, higher level conceptual model, based on 'real world' entities and relationships. Such a model would :

* contain semantic information about the real world,

* provide a high degree of data independence,

* provide a communication mechanism between analysts and users (clients).

The entity relationship (ER) model is a generalization of the network and hierarchical data models which does not have the restriction of functional links. Relationships between entity types can thus be multiple as opposed to binary, and can also be many-many and recursive. Chen's approach laid more emphasis on the relationships between the data than did Bachman's DSDs enabling more flexibility in the modelling phase of analysis.

The objectives of entity analysis can be outlined as :

i) understand and document the nature of the information (data) which is fundamental to the business,

ii) produce understandable and communicable model(s) to describe the structure of this information.

The resulting logical data model can then be combined with a functional model resulting from functional analysis in order for the design stage to begin.

1.34 Discussion of the Data Structure Method

In the same way as the data flow method, the data structure method as described achieves all of the aims of a structured method i.e.

* the E-R Model and Entity Life Cycle (ELC) diagrams are graphical tools which aid communication between analysts and users,

* the set of documents resulting from entity-relationship, attribute and functional analysis make up a logical model of the 'system',

* the method by which the analysis is carried out requires a high level of analyst-user interaction.

In addition, unlike the data flow method, the data structure method is more data-oriented [Zimmerman, 1983] in that the data is analysed first, independently of the functions on it. In this way, the structure and nature of the data is fully understood so that if the functions should change, the analysis of the data would still apply.

The functions and the data are however linked in both the ELC diagrams, and in the functional analysis stage. The ELC diagrams logically show the different states in which an entity can be with regards to a single attribute, and the corresponding events which cause these changes in state. The functional analysis breaks down the tasks within the environment into function hierarchies, thus making them equivalent to a set of levelled DFD's in the data flow method.

The issues of control and procedural knowledge are left out of the data structure method entirely, as these are physical issues to be considered at the physical design stage. At the end of the analysis stage, the logical specification defines :

* the data structures required i.e. the types of objects to be defined and their corresponding attributes or properties,

* the 'legal' states in which an entity can be i.e. the values a single attribute can take and the corresponding events which cause the changes in attribute value/status,

* the functions required by the users and the entities effected by these functions,

which should be a sufficient definition of the user-analyst view of the system for design to begin.

1.4 Approach Taken

The main issue addressed by this research concerns the application of a logical modelling method to the knowledge elicitation stage of expert systems development. The method concentrates first on defining a boundary between explicit and non-explicit processes, then uses data structure modelling techniques to formulate the explicit knowledge which will constitute the knowledge base. In this section the explicitness boundary is discussed with respect to a) the meaning of explicitness and b) how such a boundary could be defined. The concept of logical data structure modelling of the knowledge within the boundary is then introduced as is the practical use of a data oriented methodology.

1.41 The Explicitness Boundary

i) Importance of the Boundary

The importance of the boundary is related to the question of exactly what should be incorporated into the knowledge base of the expert system. Addis [1985] defines expert

systems as those computer systems where the knowledge is organised in an explicit form e.g. into facts, inferential knowledge and deduction methods. Further, a knowledge base of an expert system has been described by Black [1986] as "... an explicit structured representation of the underlying rules of some area of expertise ...", which is structured by the use of generalisations making the storage of large quantities of facts redundant. It is the presence of these generalisations or organising principles which makes the distinction between knowledge as in a knowledge base, and data items as in a database.

ii) What is Explicitness ?

Such definitions concerning expert systems suggest that the knowledge base should only contain knowledge in an explicit form and thus the boundary can be called an 'explicitness boundary'. It is not clear however, what exactly is meant by explicitness in this context. By definition, an explicit statement is one which is expressed clearly and unambiguously and in which there is no hidden or latent meaning. The opposite term, implicitness applies to statements which are implied or can be inferred from others but are not clearly expressed.

In the context of knowledge bases however, the use of the term 'explicit' must mean more than this. Typically, knowledge bases have been constructed using representations based on rules, frames etc. all of which are considered to be more explicit than e.g. Fortran or 'C' code. The former types of representation certainly possess the attribute of clarity, though not necessarily unambiguity whereas the latter type i.e. conventional programming languages are often clear only to those who have produced the code and are therefore familiar with the thought processes which helped to produce it.

A human being's knowledge or expertise is not considered to be explicit if it cannot be expressed easily to others. The task of knowledge elicitation is however to express expert knowledge in a form which is comprehensible to others. The notion of explicitness thus becomes more than just clear, unambiguous expression. The idea of 'comprehensibility' of the resulting formulation becomes an important attribute. What, however, do we mean exactly by comprehensibility ?

In terms of the goals of knowledge elicitation, we want the knowledge base, or at least a representation of the expert knowledge, to be understandable to the experts and possibly the users involved in system development. Whilst not attempting to discuss in detail the nature of human learning or understanding, it seems reasonable that to a certain degree, the power to understand something may require the ability to learn. In some situations, e.g. understanding a language, it is easy to learn without understanding e.g. the alphabet is first learnt 'parrot fashion', words can be used in terms of an individual having learnt the spelling and pronunciation, and even sentences can be memorised and repeated without understanding. In the same situation, it may be possible to have some understanding without learning, but full understanding of a language may only exist after a considerable amount of learning.

So, if comprehensibility is concerned, at least to some degree, with the ability to learn, in addition to the knowledge base being clearly and unambiguously expressed, it will benefit from being in a learnable form. The organising principles which are a distinctive feature of a knowledge base also are a product of learning as part of learning is concerned with general principles as opposed to memorising large quantities of facts. In addition, it follows that the expressed knowledge, if learnable, must be communicable i.e. explainable to others.

Michie's [Michie, 1982a,b] idea of making knowledge comprehensible, and therefore explicit, is that the representation of the knowledge should be within the 'human window'. Techniques for the representation of any problem can be viewed on a combined scale of memory usage and processing ability. At one end of the scale, solving the problem requires the use of a large amount of processing ability and only a small amount of memory. At the other end of the scale the converse is true i.e. a small amount of processing ability is required but a large amount of memory. As an example, apart from a few exceptional individuals, there is a limit to the complexity of a mathemetical process e.g. addition which can be carried out in the head without breaking that process down into sub-processes. If a human being is asked to add e.g. 4567 + 1278, he is most likely to break this process down by adding pairs of digits in turn starting either from the left or the right. Each process then consists of the addition of two (or three where a carry results) single digit numbers. The alternatives to the latter method

are either to carry out the addition as one process, or to memorise a look-up table comprising all combinations of numbers and their sums. Neither of these representations of problem solving lie within what Michie terms the 'human window' as the former is too complex for the brain's calculation powers, and the latter requires too much memory.

It follows therefore that in order to make the problem representation comprehensible to a human, and therefore explicit, some method of representing the problem must be found which lies between the two extremes above i.e. within the human window. It is then this explicit representation which should be used in order to make the knowledge base or a representation of it, comprehensible to the experts and users.

An additional problem with expert systems however is posed by the different levels of ability of the experts supplying the knowledge and the users of the system. It is reasonable to expect that the 'human window' for each of these groups will be at a different point on the memory/processing ability scale. The notion of 'learnability' will also vary with this variation in the human window, and thus the two can be considered as linked. A novice in a particular subject may only be able to solve problems from memory whereas the expert would be able to solve more difficult problems by using more processing power to work out the solution from first principles. The converse may also be true i.e. where an expert may not need to work out the solution to a problem from first principles but may be able to recognise the solution instantly, whilst the novice needs to work out the solution from first principles. Thus the notion of an explicit knowledge base is dependent on both the experts and users of the potential system.

Explicitness in the context of knowledge bases can now be summarised as follows :

* clear, unambiguous expression,

* comprehensible to experts and users i.e.
  - explainable,
  - communicable,
  - within the human window,
  - learnable,
  - organised by general principles.

iii) How is the Boundary Defined ?

In order to make the knowledge in a knowledge base explicit, we need to firstly define the correct boundary which will satisfy the explicitness constraint with respect to both the experts and users. It is the importance and definition of this boundary which forms one of the main proposals of this thesis.

A structured approach can be taken to the problem using the data flow technique of systems analysis, in order to identify the different processes or types of processes within the expert task. An 'explicitness' boundary, akin to the automation boundary in data processing systems can then be drawn which will form the boundary between the knowledge base and other parts of the system. The use of this technique is investigated further in chapter 3.

A useful tool for defining the boundary is the requirement that the input and stored data is confined to limited data sets. In chapter 3, the concept of limited data is described, and this requirement justified by the arguments outlined below.

a There is a class of problems for which limited data exists i.e. where the data input to and output from the problem is naturally in a limited form and thus for which such a method of elicitation will work.

b If we look at the definition of explicitness as defined above, the practicality, if not necessity of using limited data sets becomes clear. In order to be comprehensible to the experts and users, one of the requirements is that the expressed knowledge should lie within the human window of the experts and users involved. By definition, this means that the amount of memory and processing ability required to represent the knowledge is small which in turn suggests only a limited input data set.

A small memory, as required for explicitness, can still deal with a large input data set but a small one would be sufficient as only the significant values of that data set will be stored in memory and therefore recognised by the processes concerned. As an example, humans can detect all temperatures i.e. this is a continuous process with a large data set but, for decision making as regards eg. controlling central heating, only a small number of data elements or values need to be remembered and thus used in the decision-making process. Take the following set of rules :

IF t >70 THEN 'heating off'

IF t >60 and t <70 THEN 'heating on, low'

IF t <60 THEN 'heating on, high'

the important temperatures to remember are 60, and 70. There is therefore no need to use a large input data set which represents the continuous range of temperatures. This argument is amplified in section 3.22.

c As in any computer system, the quality of the user input is vital to the overall correctness of the system i.e. taking the processes as a 'black box', if the input is not correct, then the output cannot possibly be correct. Due to i) the character of the input data i.e. subjective, open to interpretation etc., and ii) the differing levels of ability to discriminate, of users and experts, the importance attributed to the definition of the input data is even higher for expert systems than for traditional data processing systems. In order to reduce the level of user variability and thus preserve user independence therefore, it becomes important to pay attention to data definition in

any knowledge elicitation method. Section 3.23 of chapter 3 investigates the importance of, and methods of preserving, user independence.

d As discussed previously, one of the requirements for explicitness is communicability. For the purposes of knowledge elicitation, non numerical data, even about continuous processes is communicable (to other humans) only by means of words, and terms representing a finite data set.

The result of having defined the explicitness boundary can be summarised as :

* identification of exactly which processes lie inside and which outside, the knowledge base,

* definition of the limited data sets which both affect the knowledge base and are produced by it.

1.42 Data Structure Modelling of Explicit Knowledge

Most knowledge elicitation methods, as discussed in chapter 2, take a functional approach, with definition of the input data for the system being defined as a secondary process. It is clear however that data is necessary in order for functions to work, in any type of system, and there seems to be no evidence to suggest that an approach which defines the data first, before the functions would not be a viable one.

Data structure modelling techniques have been investigated for their ability to model both the intensional and extensional, static knowledge of the expert. It is shown in chapter 4 that entity-relationship diagrams can be used to represent an abstraction of the intensional knowledge within the expert's domain, and entity life cycle diagrams provide the ability to model changes in status of the entity types defined, thus representing the more detailed intensional knowledge.

As a logical model, the resulting formulation of the knowledge is independent of the physical implementation. It is important however that translation to a physical representation is

possible from this model. By use of an example, implementation of the logical model using two of the major knowledge representation techniques i.e. frames and production systems is illustrated in chapter 5. Further, general relationships between these techniques and the logical model resulting from the entity-relationship approach are outlined.

1.43 Utility of a Data-Oriented Method in Practice

During the early stages of research for this thesis it was felt that a data oriented approach to knowledge elicitation was a viable one, but the actual method and techniques to use were not established. An 'ad hoc' data oriented knowledge elicitation method was applied to two problems as part of the ARIES (Alvey Research into Insurance Expert Systems) project undertaken by Logica (UK) Ltd. The emphasis of the approach taken, as outlined in chapter 6, was

i) to define all data within the problem-solving task i.e. factors and the limited set of values they could take before defining any of the functions on the data,

ii) to test the suitability for users of the resulting input data sets defined in i) and thus the levels of accuracy and consistency which could be expected of the system.

The aims of the approach taken were to firstly assess the suitability of a data oriented approach to knowledge elicitation and the effect on the ease or difficulty of the process, secondly to stress the importance of involving the end users in the knowledge elicitation phase of system development, and thirdly to highlight the importance of concentrating on data in order to improve the robustness of the system.

The conclusions reached from this practical work were

i) that a data oriented approach was the right approach, but that a more rigorous, structured approach should be taken rather than the 'ad hoc' methods applied,

ii) that user involvement was a valuable part of knowledge elicitation and the tests for accuracy and consistency of the system vital in order to assess the performance of the system in the hands of the users.

CHAPTER 2 KNOWLEDGE ELICITATION

2.1 Tools and Techniques for Knowledge Elicitation

Methods used for knowledge elicitation vary widely and originate from different fields such as cognitive psychology, instructional technology, systems analysis etc. In this section, the methods currently developed and used will be discussed under the following broad categories :

* interview techniques,
* observation,
* automated techniques,

Each technique is discussed where relevant, with respect to both

i) the specific issues of logical modelling, explicitness of the formulation, limited data sets and user independence,

and

ii) the degree to which it supports or contests the ideas in this thesis.

2.11 Interview Techniques

Interview techniques can be divided into two main types - informal and formal. The former can be used with some success in the earliest stages of knowledge elicitation, in order for the knowledge engineers to become familiar with the domain, and with the experts, and *vice versa*. Techniques used include discussion and short lectures or tutorials given by the experts, which can be useful for initially defining the boundaries of the domain to be studied, and the main features of the problem.

Formal interview techniques make use of structuring methods which are generally borrowed from the field of psychology.

Apart from these however, other methods of structuring include the use of examples to lead the interviews, giving a focus to the discussion.

Information gained from the study or 'walkthrough' of examples can then be generalised and tested against different examples. Psychological techniques as used for knowledge elicitation can be loosely grouped into the following three categories of techniques :

* memory probing,
* concept sorting,
* multidimensional scaling.

2.11.1 Memory Probing

Memory probing techniques such as Flanagan's critical incident method [Gammack & Young 1984, Welbank 1983] take the 'example walkthrough' method a stage further by asking the expert to recall particularly 'memorable' events. These could be e.g. difficult problems which were hard to solve, or unusual cases. Such methods would be more likely however, to result in the elicitation of exception handling knowledge, rather than knowledge of how to deal with the larger proportion of 'normal' cases.

These techniques provide a method of focussing on problem specific knowledge in interviews, but are functional and thus do not provide a means of structuring the knowledge elicited, as do the following techniques.

2.11.2 Concept Sorting

Concept sorting [Gammack & Young, 1984] concentrates on defining the concepts within the problem-solving domain which are of interest to the expert, in order to see how the expert structures his domain knowledge. The expert is asked to sort the set of concepts which broadly describe the domain into groups according to common features amongst the members of these groups. The task is an iterative one, which eventually forms a hierarchy

with basic components at the bottom, and higher order, or more abstracted concepts at the higher levels.

This technique has been used to discover the different ways in which experts and novices view the same problem of categorising and representing physics problems [Chi, Feltovich & Glaser 1981]. Both groups were asked to describe categories of physics problems in terms of specific problems associated with each category, and the methods of solving these problems. The resulting protocols were converted to networks of node-link structures, which highlighted the differences in the two perceptions of the same problem. The experts networks contained principles of physics, and the knowledge of how to apply these principles, whereas the novice networks contained superficial features of the problems, but little knowledge of the laws or principles of physics.

The latter application of the concept sorting technique recognises the expert-novice distinction and could therefore be very useful in situations where the users of the expert system are of a different level of ability to the experts. If the differences in perception of the problem between the two groups of people can be identified before implementation of the system, then this can be taken into account in system design. Such a technique could therefore be complementary to any knowledge elicitation method, as a means of identifying the concepts common to both sets of 'users'.

2.11.3 Multidimensional Scaling - Repertory Grid Analysis

The method of structuring the domain knowledge provided by repertory grid analysis, a form of multidimensional scaling, is concerned with the elicitation of implications or relationships which may exist between concepts within the domain. It is therefore useful as an elicitation tool in domains where expertise consists of making discriminations between closely related concepts.

The idea of repertory grid analysis stems from the theory of Personal Construct Psychology, as developed by George Kelly in the 1950's. It was used as a method of discovering how

individuals categorise experiences and classify their environment in order to improve their ability to predict future events and so act more effectively in that environment.

Much of the work in this area has resulted in the development of automated systems either for part of the elicitation process [Shaw 1980,1981, Gaines & Shaw 1986], or for the bulk of the process, including prototype development [Boose 1985, 1986]. These systems will be discussed in the automated techniques section, whilst in this section, the method itself and its use as an elicitation tool will be discussed.

Elements i.e. concepts or objects within the domain, on which discriminations can be made, are the subject of the analysis. These may be e.g. living creatures, books, software packages etc, and form one dimension of the rating grid. The expert is asked to compare three specific elements at a time, and to define a characteristic that distinguishes one element from the other two. This characteristic or trait, together with its opposite trait forms a bipolar construct, and the list of such constructs which results from analysing all combinations of the elements in this way, forms the second dimension of the grid.

So far, the grid consists of elements and constructs, but no ratings. The expert is then asked to rate each element according to each construct, on a scale of e.g. 1 - n where n is equivalent to the number of elements, resulting in a grid as in fig. 2.1. (This is an extension to the original rating process, where each element corresponded either to the left hand pole (trait) or to the right-hand pole. A further extension, used within Boose's Expertise Transfer System (ETS) allows the use of 'B' for 'both poles' or 'N' for 'neither pole'.).

Once the rating grid has been elicited it must then be analysed in order to define patterns and relationships between elements and constructs. Manual analysis techniques include laddering, matching scores and verification of relationships. The laddering technique results in a hierarchical structure of constructs produced by the identification of sub-ordinate and super-ordinate relationships between constructs. The former are identified by asking 'How ?' questions, and the latter by asking 'Why?' questions about the constructs.

Constructs	Elements					Reverse Constructs
	A	B	C	D	E	
Happy	5	1	2	3	4	Sad
Cooperative	3	2	1	4	5	Uncooperative
Motivated	2	3	1	4	5	Demotivated
Strong-minded	5	3	4	1	2	Weak
Complex	3	1	5	2	4	Straight-forward

Fig. 2.1 Simple rating grid where elements are people and constructs are characteristics.

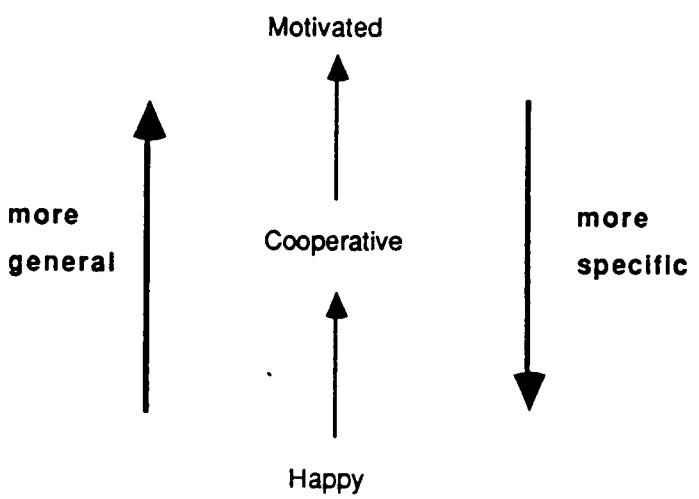


Fig. 2.2 Laddering constructs

For the grid in fig. 2.1, the laddering technique would involve asking questions such as 'Why is someone co-operative ?' in order to elicit a super-ordinate i.e. more general construct. The answer may be e.g. 'Because they are motivated', resulting in the relationship below. A further question 'How do you know that someone is co-operative ?', i.e. 'what is the evidence for that ?', should elicit the sub-ordinate construct, e.g. happy. The set of relationships in fig. 2.2 would then result.

Where a relationship appears to exist in the grid i.e. where there is a correlation between the ratings of two traits for all or a majority of elements, this relationship must be verified. The result of verification will be a series of implications between constructs which may be e.g. parallel i.e. for two bipolar constructs A-B and X-Y, A implies X, and B implies Y. Other implications which may exist in the grid can be classified as orthogonal, reciprocal or ambiguous.

In order to elicit such implications, direct questions about the pair of constructs must be asked, e.g. 'If someone is co-operative are they always motivated ?' and 'If someone is un-cooperative are they always de-motivated ? '. Positive answers to both of these question would imply a parallel implication between co-operative/unco-operative and motivated/de-motivated. A positive answer to the first and negative to the second would however imply an orthogonal relationship between the constructs.

Calculating matching scores between elements or constructs indicates the degree to which the expert links these concepts, and is a more statistical method of the above form of verification of relationships. A still more complex method is to use cluster analysis, where initially, distance measurements are calculated over the grid, then these are turned into similarity measures on a scale of 0-100 %. For constructs, this process is more complicated than for elements, as the effect of reversing the poles of each construct have to be taken into account. The result is known as a 'focussed' grid in which elements and constructs are rearranged according to similarity, reversing constructs where necessary. An automated

version of grid focussing called FOCUS has been developed [Shaw 1980] which uses the technique of cluster analysis.

The advantages of this technique for elicitation have been outlined [Hart, 1986] as :

* making the expert think about the problem and gaining his perception of it in terms of concepts,

* useful in the early stages of elicitation for selecting attributes (i.e. the constructs) and examples (i.e. the elements) for automatic rule induction,

* highlighting patterns and associations in the domain which may not be obvious even to the expert.

In addition, the technique has been extended [Shaw 1981] to compare and contrast different views of the same domain, as discussed in the automated techniques section.

Although repertory grid analysis results in a set of implications or rules, it is basically data-oriented. The initial analysis involves the identification of elements on which the expert discriminates and constructs which enable this discrimination. Further, the method of grid construction forces discreteness although the constructs are only bi-polar. An idea of continuity within a trait is simulated to some degree by the ranking process, but continuous data is not explicitly represented.

The construction of the grid and subsequent analysis for implications between constructs could be used as a method of identifying entity and relationship types in order to construct an entity-relationship (ER) model. This would suggest that the repertory grid is in fact a form of logical model, although the relationships are not explicitly represented. The translation of a grid to an ER model is straightforward involving the use of entity sub-types to represent the two poles of each construct, as in fig. 2.3.

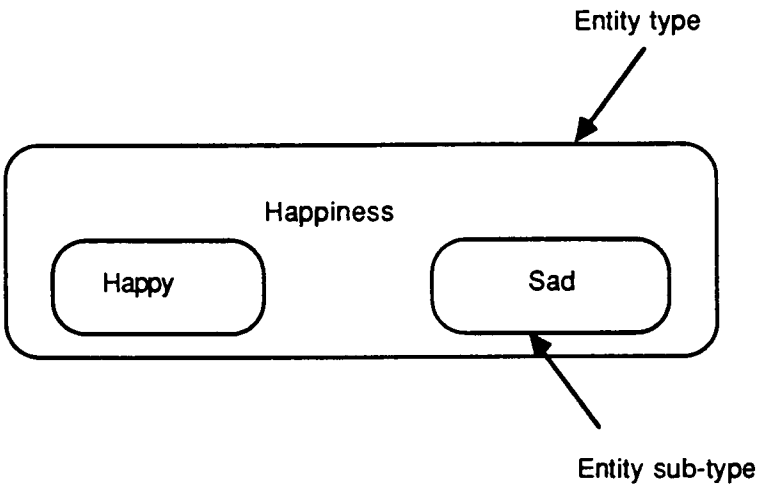


Fig. 2.3 Representing construct poles using ER conventions

The relationships or implications mentioned previously would be represented as in fig. 2.4 using ER modelling conventions. The examples which represent implications from a grid are taken from Boose [1985].

Note that the latter relationship of fig. 2.4 is not equivalent to the first ie. parallel relationship which exists only between specific entity sub-types.

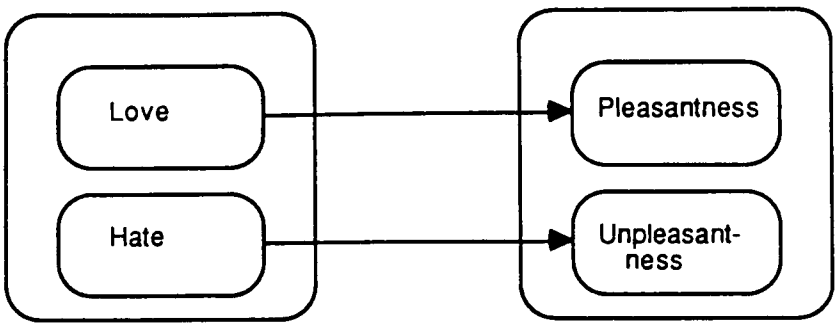
Additional methods which may be classified under the heading of interview techniques include teachback interviewing, task analysis, domain and knowledge analysis, and extended relational analysis.

2.11.4 Teachback Interviewing

Teachback interviewing [Johnson & Johnson, 1985] is based on the idea of Conversation Theory, attributable to Pask, and was developed as an attempt to capture a 'competence model' [Johnson, 1985] of the expert. Such a model should capture the conceptual structure of the expert knowledge in addition to the procedural skills involved in the problem solving task, and should model the relationship between these two aspects of the knowledge. The aim is to enable the system to justify its conclusions in a similar manner to the expert due to a good cognitive match between the system and the expert.

Conversation theory concentrates on the idea of making concepts and understanding public knowledge by means of interaction between participants. The teachback principle involves the expert describing a procedure to the analyst, who then teaches the same procedure back to the expert until it can be said that a shared concept of the procedure exists between the expert and analyst. Understanding at this stage can only be said to be on a conceptual level however, as the expert and analyst may not necessarily have the same thought processes even though they can describe the procedure in the same way. The second part of the teachback exercise therefore attempts to capture these thought processes by means of the expert explaining how he remembers and reconstructs the concept. The teachback exercise

Parallel



Orthogonal

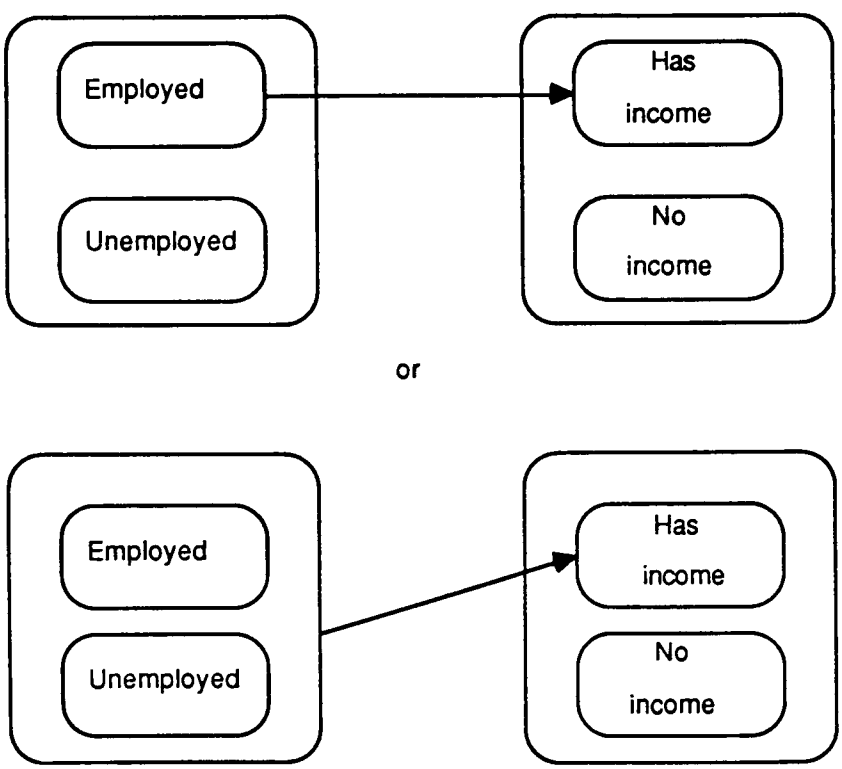
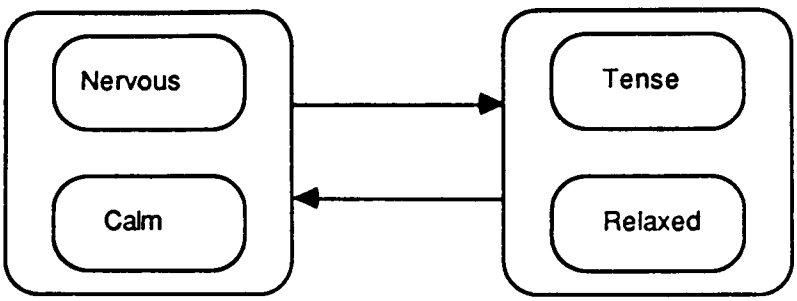


Fig. 2.4 Implications identified from repertory grid analysis

Reciprocal



Ambiguos

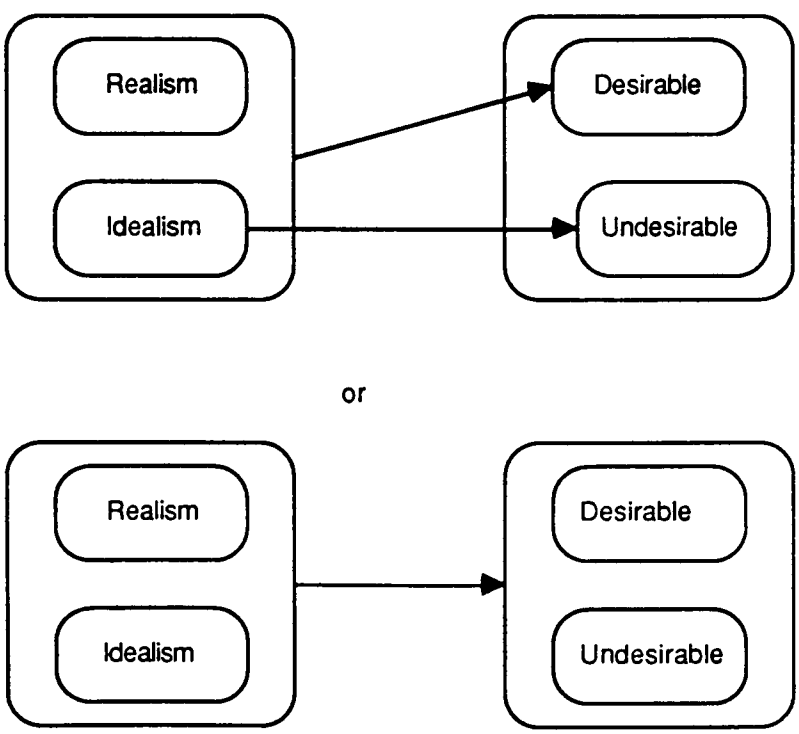


Fig. 2.4 (cont.) Implications identified from reperetory grid analysis

is carried out as before, continuing until the expert is satisfied with the analysts version. At this point the expert and analyst are said to have a shared understanding of the task.

The knowledge contained in the transcript resulting from this exercise is transformed into a systemic grammar network (SGN) consisting of nodes and links. The nodes describe concepts which may be objects or procedures, and the links describe the methods of reaching one node from another. Fig. 2.5 consists of a straightforward example of a systemic grammar network, indicating the kinds of links allowed.

Advantages of the teachback method are i) that the technique elicits the experts conception of the domain rather than that of the knowledge engineer, and ii) that the use of SGN's provides a communicable description of the knowledge before implementation.

Although the SGN is a communicable description of the knowledge and is therefore useful during elicitation and design, it cannot be described as and therefore does not offer the advantages of a logical model. Firstly, the SGN contains an element of 'how' the problem should be solved in addition to 'what' is to be solved, and secondly the knowledge represented is not intensional knowledge but extensional i.e. details of the knowledge are included rather than the main facts and principles.

Problems associated with this method as identified by the authors are that the teachback method of interviewing is not strongly structured, and poses a difficult task for the knowledge engineer (analyst). In addition, the idea of obtaining a model of competence of the expert is in itself a good one, but users of the resulting system, who may be other experts or users of lesser ability, will not necessarily have the same conceptual structure of the problem. There is no mention of how this problem is resolved and at what level the system is presented to the users, in particular, the issue of explanations of reasoning. There are obvious implications therefore concerning the user independence of the resulting system, and also how usable and widely applicable the system would be.

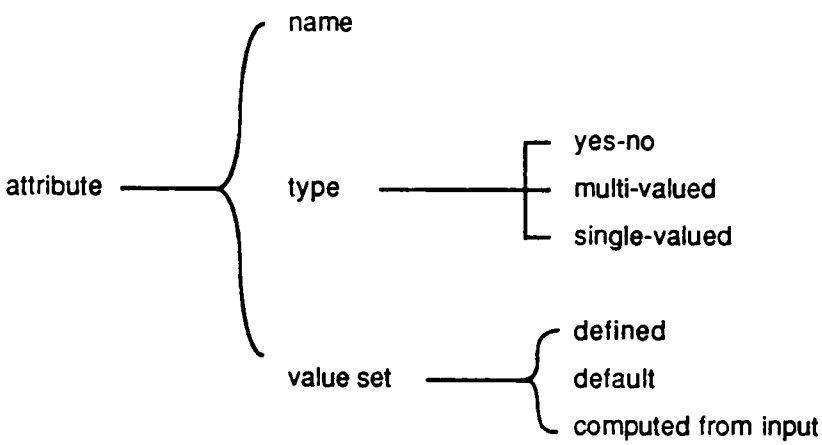


Fig. 2.5 Systemic Grammar Network to define an attribute

2.11.5 Task Analysis

Broadly, task analysis is a functional approach to elicitation which involves breaking down the main problem into a hierarchy of sub-tasks which must be performed. The objectives of task analysis in general can be outlined [Johnson, Diaper & Long 1984] as, the definition of :

- the objectives of the task,
- the procedures used,
- actions and objects involved,
- time taken to accomplish the task,
- frequency of operations,
- occurrence of errors,
- involvement of sub-ordinate and super-ordinate tasks.

The result is a task description which must be formalised in some way e.g. by flowcharts, logic trees or formal grammar. The process does not however describe the knowledge of the expert i.e. the underlying knowledge structure, but how the task is performed and what is needed to achieve it's aim - the former is only elicited incidentally.

As a means of breaking down the problem area into its constituent sub-problems, task analysis would be useful in a similar way to data flow analysis. Although the method does incorporate the analysis of the data associated with each task, it is lacking in graphical techniques for representation of this data, and therefore remains mostly useful for functional elicitation.

The approach to cognitive task analysis recommended by Braune & Foshay [1983], based on human information processing theory [Newell & Simon, 1972] is less functional than the basic approach to task analysis as outlined above, concentrating on the analysis of concepts. The second stage of the 3-step strategy is to define the relations between concepts by analysing examples, then to build on the resulting schema by analysing larger problem sets.

The schema which results from the above analysis is a model of the knowledge structure of the expert, similar to that achieved by the concept sorting method, describing the 'chunking' of knowledge by the expert. This chunking is controlled by the idea of expectancy according to the theory of human information processing ie. the selection of the correct stimuli for solving the problem, and the knowledge of how to deal with them.

This approach is akin to the idea of ER modelling due to the concentration on the analysis of concepts and relations before further analysis of functions/tasks.

Task (and topic) analysis is also used by instructional technologists who a large amount of the time are attempting to carry out the same kind of task as knowledge engineers. Instructional technology is concerned with the analysis of the learning process, and of specific knowledge and skills, in order to improve the instructional process. Two of the approaches used within this field are Gilbert's Mathetics and Romiszowski's knowledge and skills analysis [Romiszowski, 1981].

Mathetics is an approach to defining the knowledge required to make a decision related to a physical task, concentrating on what the subject must know before carrying out that task. The analysis consists of first identifying the stimuli the subject will meet and then analysing the desired responses to these stimuli. The next stage is to explore and map out alternative courses of action, then to identify important discriminations and generalisations in order to specify the concepts the operator should master. The emphasis of this approach is clearly on identifying what the learner should know in order to perform a task. This approach, like that of Braune and Foshay seems less functional than the conventional approach as the analyst concentrates firstly on the 'data' to which the subject responds and only then on the functions which act on this data.

Romiszowski's method developed as an approach to the analysis of learning difficulties, is based on the distinction between knowledge and skill. Knowledge can be defined in this context as the information stored in the learners mind, and skill as the actions and reactions performed to reach a goal, using this knowledge. Four types of knowledge ie. facts,

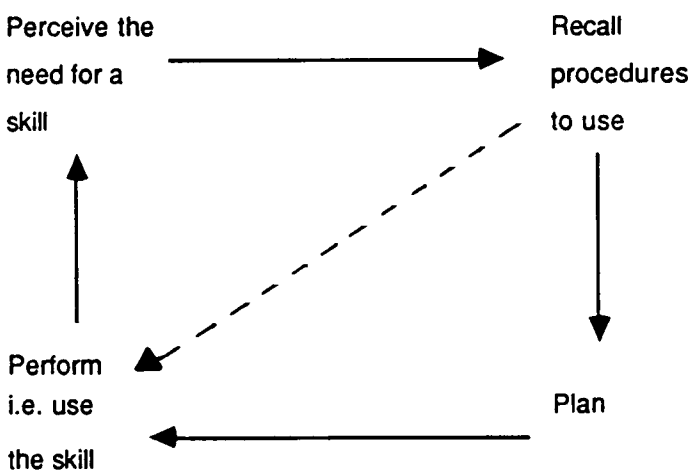


Fig. 2.6 Romiszowski's skills cycle

concepts, principles and procedures can be identified, and four types of skill i.e. cognitive, acting, reacting and interacting. Skills are analysed in terms of the cycle of fig. 2.6.

Different types of knowledge would be used at different times in the cycle eg., perception involves the recognition of stimuli requiring prior knowledge of such stimuli i.e. objects or concepts, and recall relies on the existence of the required procedures in memory.

2.11.6 Domain and Knowledge Analysis

Grover's methodology [Grover, 1983] of domain analysis concentrates on three main phases as outlined below which result in a Knowledge Acquisition Document Series which acts as a consistent basis for knowledge-base design :

i) Domain definition and description - the background to the problem is investigated and the problem described. This stage includes finding the relevant reference documents and experts within the domain, becoming familiar with the experts jargon and explaining various examples of the experts reasoning. The output of this phase is a Domain Definition Handbook containing the details resulting from this preliminary investigation.

ii) Fundamental knowledge formulation - the aims of this phase are to investigate example situations which fall into five categories of fundamental knowledge i.e. the most nominal, most expected, most important, most archetypal and best understood. This knowledge will form the basis of the system i.e. the main goals and rules which would be able to cope with the majority of cases. The 'rulebase' as it exists at this stage is written in English rather than in a formal language, and is supplemented by other knowledge such as meta-rules or strategies for possible inclusion as control rules, descriptions of objects and relationships in the domain, input sources and formats etc.

iii) Basal knowledge consolidation - this is the 'review and improve' phase of the knowledge acquisition cycle in which the lowest level of detail is elicited from the

expert. In addition to improving the rule base, consulting other experts and making refinements generally, this phase is concerned with e.g. control and sequencing of rules, testing consistency, etc.

Grover attempted four techniques for knowledge acquisition in the second of the above three phases of system development. The first, 'forward scenario simulation' involves the verbal 'walkthrough' of a problem as discussed at the beginning of this section. Although successful, this was found to have problems related to confusion over terms and definitions not covered sufficiently in the first phase, rule fanout which meant that at times certain reasoning paths were forgotten, and the distinction between what was actual reasoning, and what was the job of the reasoner.

The other useful technique tried was that of 'pure reclassification' whereby observations are reclassified into more specific objects and activities in order to construct rules which will fill in gaps in the observations. The remaining two methods used to elicit the fundamental knowledge corpus were 'goal decomposition' i.e. problem reduction and 'procedural simulation' i.e. protocol analysis with interruption by the knowledge engineer, as discussed in the following section.

This technique would seem similar in its approach to a data-oriented logical modelling approach due to the first phase of domain definition. The result of this phase, the Domain Definition Handbook is not however as detailed and thorough as e.g. a 'data dictionary' which would result from e.g. data flow analysis and/or ER modelling but is more a description of the background to the problem in terms of example reasoning scenario's etc. The method would not seem to offer many advantages over other functional approaches therefore, except that the knowledge engineers may have a better overall understanding of the expert's domain after the first phase of analysis.

Breuker and Weilinga [1983] suggested extensions to the above method of domain analysis in their scenario for knowledge analysis. In this approach, after an initial stage from which is produced a Domain Definition Handbook (after Grover), the analysis is directed at firstly the

task and its environment and then the user and domain expert. The result of the task analysis is a complete specification of the task, authoritative experts, the type of expertise etc. adding up to a Task Specification Handbook. Analysing the environment of the task produces a specification of the constraints imposed on the design and implementation of the system in terms of resources, legal issues etc.

The work of Breuker and Weilinga places considerable stress on the importance of analysing the user in order that a suitable user interface is designed. The method of analysis suggested includes interviews, simulation experiments, and analysis of expert-user interactions. The final stage of expert analysis as it is labelled in the scenario uses the methods of knowledge acquisition outlined above as used by Grover, and is therefore an analysis of the expert knowledge as opposed to the expert himself as would be implied by the name.

2.11.7 Extended Relational Analysis

Extended Relational Analysis (ERA) [Addis, 1986] is designed as a method of elicitation which involves the construction of a conceptual model in the form of a semantic functional dependency (sfd) graph. The method, which is based on relational analysis for database design, is concerned, as is the method in this thesis, with the analysis of the data of the problem in order to arrive at a model which represents the users view of the domain.

Addis distinguishes in his method, between the processes of elicitation and acquisition of knowledge, stating that elicitation is concerned with the interviewing process in order to arrive at the conceptual model (sfd graph) whereas acquisition is concerned with collecting the detailed knowledge which fits into the framework defined by the sfd. The result of elicitation thus defines the intension of the knowledge, and the acquisition the extension. For a database this would seem a good distinction but in terms of expertise and building knowledge bases, the distinction seems a false one. Whereas in databases the extension of the database is represented by occurrences of data, in knowledge bases, the extension of any logical model would almost certainly contain some of the expertise. For example, in the

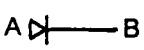
suppliers-parts example used to demonstrate the ERA process, it may be part of the expertise to know which suppliers supply which parts, which parts can be replaced by others etc. Thus the interviewing process goes far beyond defining the intension of a knowledge base. It may be the case that for knowledge bases, the logical model defines a meta-intension i.e. an abstracted view of the intensional knowledge. This intension/extension distinction is discussed in more detail in chapter 4 however.

In outline, the method is a ten stage modelling process based around normalisation theory. The first three stages result in a set of tables in third normal form (3NF) from which entity sets and the relationships between them emerge. The 3NF model is then refined in terms of expanding the attributes as necessary and replacing many:many relationships with one:many by the creation of intermediate entity sets.

At this stage the relationships between entity sets are equivalent to mandatory relationships of an ER model. In order to create the sfd however, existence constraints are introduced by replacing all one:many relationships with either 0-implies or 1-implies constraints. As is shown below, these are equivalent to the optionality constraint as used in the ER model.

sfd graph		ER Model
0-implies		
A >|—— B	Each entity of A maps into some entity of B but there may be some entity of B with no associated entity in A.	A >— - - - B

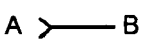
1-implies / surjection



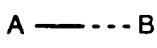
Each entity in A maps into some entity of B and an entity in B cannot exist without at least one entity in A.

The special cases of 0-implies and 1-implies known as injection and bijection can also be represented on an ER model using the one:one relationship as below.

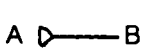
injection



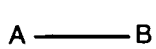
Only one entity in A is associated with any entity in B, but an entity in B can exist without an associated entity in A.



bijection



Only one entity in A is associated with any entity in B, and an entity in B cannot exists without an associated entity in A.



At this stage therefore the sfd can be said to be equivalent to an ER model although the method of reaching such a model uses a different approach to the ER modelling technique. In general, the relational modelling technique is an approach at developing a conceptual model using a 'bottom-up' approach [Howe, 1984] whereas the ER modelling technique based on Chen's work [Chen, 1976] is a 'top-down' approach. The former focusses on the interdependencies of data attributes and the subsequent identification of entity and relationship types, whereas the latter begins with the identification of entity and relationship types and then uses these to construct a framework into which the attributes may be slotted. The difficulty with the former approach is with the identification of the attributes without first identifying the entity types of which these attributes are properties.

The remaining steps of ERA are concerned mainly with fourth and fifth normal form i.e. consideration of update restrictions on the data. The resulting sfd includes the projection and join operations which represent these constraints. The ER model does not explicitly show such update restrictions, but for the purposes of building knowledge bases the representation of such detail on a logical model may not in fact be necessary.

2.12 Observation - Protocol Analysis

Protocol analysis, as originally posed by Newell and Simon [1972] is a popular technique within knowledge elicitation and one of the only methods to involve observation of the expert as opposed to direct interviewing. The technique is based around the idea of studying the behaviour of an individual solving a single problem. The data which results from the observation is called a protocol and consists of the subjects verbalisations during the problem-solving together with any other comments made by the analyst.

The experiment carried out by Newell and Simon involved the observation of a subject solving an encryption problem. The subject was asked to speak aloud at all times while he worked, and these verbalisations recorded. There was therefore no record of the written work of the subject. The resulting protocol was broken down into numbered phrases which

included remarks of the experimenter, with phrases correlated to assertions. The method of analysing the protocol is however very specific to the problem being solved.

Within the expert systems field, protocol analysis has been used successfully in the domain of leukemia diagnosis [Myers, Fox, Pegram & Greaves 1983]. The expert made diagnoses from 63 patient records which consisted of cell-marker data, and were of varying levels of completeness and quality. The method of analysing the resulting process in order to extract the knowledge for the system was a three-stage process as outlined below :

* highlight statements in the transcript which contain substantive information e.g.

  "...70% blasts, preparation gives enrichment of blasts, so 70% may be a minimum...",

* simplify these statements to represent basic relationships being described i.e. abstract/generalise e.g. from above,

  "preparation enriches for blasts, so take the blast count to be a minimum",

* prune these statements to give a summary table containing all knowledge as IF .. THEN rules.

The resulting system which consisted of 21 rules was refined and expanded to 49 rules then tested on 100 patients. Further minor modifications were made to the rule base and performance of this final system found to be 81% correctness on the test set used to elicit the rules from the protocol, and 59% on the new test set of 100 patients.

An observation made by the authors was that the system tended to over-diagnose which was probably due to the lack of negative rules in the system i.e. rules which rule out possibilities rather than suggesting them. This could be an inherent weakness in the technique of taking protocols, as in addition to verbalisation being a difficult task in itself, the expert may not verbalise what is not the case.

Protocol analysis is by nature a completely functional approach to elicitation, looking for 'how' the expert solves the problem. As a technique on its own it could suffer from a lack of focus, but were it to be used after a thorough analysis of 'what' the expert considers in the domain, then only the functions which concern the concepts or objects identified would need to be elicited from the protocol.

2.13 Automated Techniques

The two main approaches to automating the elicitation process involve the induction of rules from a set of examples, and the automation of the repertory grid analysis technique which can also be expanded to induce rules.

2.13.1 Rule Induction

Quinlan's ID3 (Iterative Dichotomiser 3) algorithm [Quinlan 1979, 1982] is one of the most widely known of the rule induction techniques, which was based on the concept learning system (CLS) of Hunt, and is used here to demonstrate the idea behind rule induction techniques.

The starting point is a set of examples, called a training set where each example or instance is represented by a set of attributes values and a 'class' to which it belongs. The class may be e.g. the decision which was reached about that example. In a system to aid recruitment, the two classes to which an instance i.e. an applicant may be assigned could be IN i.e. consider the applicant or OUT i.e. do not consider the applicant. The attributes which are of interest in this exercise are :

- Attribute A - Maths 'O' level grade,
- Attribute B - No. of 'O' levels,
- Attribute C - No. of 'A' levels.

The table of applicants (fig. 2.7), consisting of the values of the above attributes and the class assigned to each i.e. the decision made, can be used as the training set. The training set

should contain at least one example of every type of case the expert might have to deal with, in this case any combination of the permissible values for each attribute.

Applicant	Attributes			Result
	A	B	C	
1	A	6	3	Consider
2	C	5	2	Reject
3	B	1	1	Consider
4	D	8	4	Consider
5	A	3	3	Reject
6	A	7	3	Consider
7	D	7	2	Reject
8	B	8	1	Consider
9	E	7	3	Consider
10	B	2	2	Reject
11	C	6	3	Consider
12	C	4	3	Reject

Fig. 2.7 Table of Applicants

The first task with integer-type values as in this example, is to identify discrete values for each attribute. For attributes which have discrete values naturally eg shape - round, square, oval triangular, these discrete values may be used in the induction process. In this case however, for each attribute a critical value can be defined which classes the attribute as desirable or not for recruitment for this job. The result is, in effect, two values for each attribute as shown below.

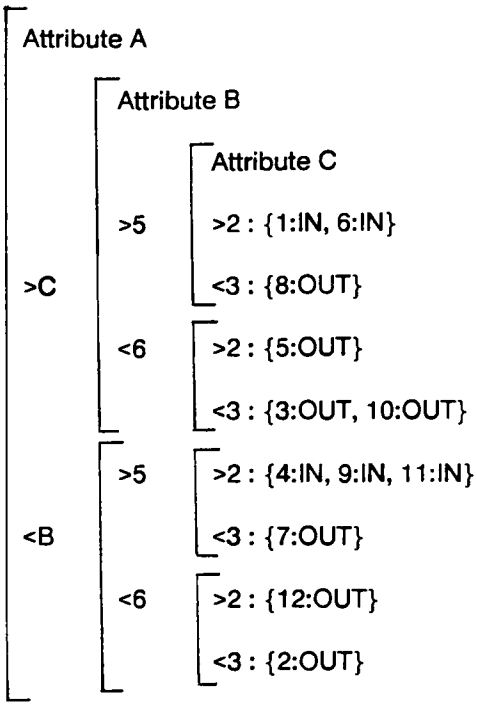
Attribute	Discrete Values	
A	>C	<B
B	>5	<6
C	>2	<3

Taking the first attribute, the algorithm splits the training set between those instances which satisfy each of the discrete values as defined above. In the case of attribute A, >C is interpreted as A or B.

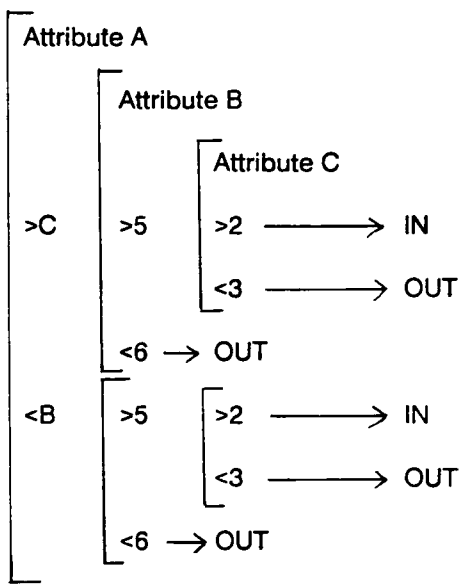
- Attribute A
  - >C : {1:IN, 3:OUT, 5:OUT, 6:IN, 8:IN, 10:OUT}
  - <B : {2:OUT, 4:IN, 7:OUT, 9:IN, 11:IN, 12:OUT}

Using the next attribute, the same process is then applied to the training set as split for the first attribute, and this process repeated until all attributes have been exhausted and/or each sub-collection of the training set contains instances belonging to only one class ie. IN or OUT.

- Attribute A
  - >C
    - Attribute B
      - >5 : {1:IN, 6:IN, 8:IN}
      - <6 : {3:OUT, 5:OUT, 10:OUT}
  - <B
    - >5 : {4:IN, 7:OUT, 9:IN, 11:IN}
    - <6 : {2:OUT, 12:OUT}



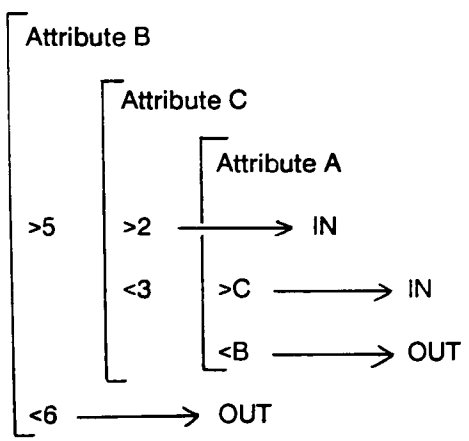
As each of the above sub-collections contains instances of only one class, the collections of instances can be replaced by classes, resulting in a decision tree as below.



This decision tree can then be expressed as a 'rule' which determines the class to which an instance should be assigned based on the values it takes for the three attributes A, B and C.

```
IF Attribute A >C
THEN
        IF Attribute B >5
        THEN 'IN'
        ELSE 'OUT'
ELSE
        IF Attribute B >5
        THEN
                IF Attribute C >2
                THEN 'IN'
                ELSE 'OUT'
        ELSE 'OUT'
```

In this example the method of selecting the next attribute to use in dividing the training set was arbitrary. The aim should however be towards the most simple tree. For the same set of attributes as above, and using the same set of examples, a much simpler tree than that achieved above can be achieved by selecting attributes in a different order. For example, the tree below results from taking attributes in the order B, C, A as opposed to A, B, C.



In the ID3 algorithm, the method of selecting the next attribute with which to split the training set is based on information theory. The aim is to select the next attribute in such a way that there is the greatest gain in information, ultimately resulting in the most simple tree.

This method of elicitation results in a set of rules which the expert can refine and which takes a relatively short time to achieve. The most difficult task is the selection of attributes and discrete values, which must take place before any induction algorithm is used. The expert must select the most relevant attributes, and must make the decision whether to include any unusual cases which might result in rules which cater for only a small percentage of the cases. Where the attributes take continuous values, the expert must also discretise the range of values. This process would not usually be as straightforward as in the example worked above, where a 'cut-off' point was selected for each attribute, thus defining two discrete values for each.

The induction of rules from examples described here supports the idea of discrete data by necessity as continuous data cannot be handled by the induction algorithm. Both the attribute values which describe the training set, and the classes to which examples in the set belong must be discrete.

It is the however the identification of the important ie. relevant attributes, and the discrete values which is the hardest part of the rule induction technique. This task could however be made easier by first carrying out an analysis of the domain using eg. a logical modelling technique, before setting up the training set.

Additional problems with the induction technique are concerned with the relative importance of attributes, and of identical examples, and the treatment of contradictory examples in the training set [Hart 1984]. It has been suggested that the algorithm could be improved as regards the latter two points by replacement of the information statistic with chi-squared, used to test which attribute is the best discriminator between classes, but this too poses problems resulting in the need for a large training set.

2.13.2 Automated Repertory Grid Analysis

There have been two main approaches to automating the repertory grid technique; the first consists of a set of techniques for analysing the grid and the second of a program for producing a prototype from a grid.

The grid analysis tools of Gaines and Shaw [Shaw 1981, Gaines & Shaw 1986] are numerous, and make use of statistical techniques to achieve various analyses of the grid contents, and therefore the domain.

FOCUS, ENTAIL and PEGASUS are all used to rearrange the grid so that related elements or constructs are near to each other. FOCUS uses hierarchical cluster analysis to calculate similarity between elements or constructs on a percentage basis, and rearranges the grid accordingly, reversing constructs where necessary. ENTAIL attaches more meaning to the similarity between constructs or elements, by building an entailment or implication graph.

PEGASUS, which is part of an interviewing system called PLANET, automates the 'matching scores' method of verifying relationships, and refines the grid accordingly. This technique is also used however, for comparing grids elicited from different people eg. a novice's grid against an expert's. This has been used in the training environment to assess the amount of training required, and could be used in the similar situation of knowledge elicitation to assess the difference between the experts and users view of the domain.

MINUS and CORE (also parts of PLANET) are also used for the above purpose of comparison but with MINUS, superimposing one grid on the other highlights exactly where the differences and similarities are. The CORE system holds the elements constant whilst calculating the change in the constructs and vice versa, then combines the two grids into one for analysing by cluster analysis. In this way, both the agreement and understanding between two people can be assessed, with the relative size of the core (ie. common) grid indicating the extent.

The expertise transfer system (ETS) of Boose [1985, 1986] automates the whole process of constructing the grid to building a prototype rule-based expert system. The following basic steps are taken :

1 Construction of the rating grid by interaction with the expert.

2 Analysis of the grid, using a method based on ENTAIL to produce an entailment graph of implication relationships between constructs. This graph can be reviewed at a later stage.

3 Generation of production rules :

a Assign concept names to the bipolar construct pairs.

b Rate the relative importance of each in the problem-solving process (eg. on a 1-5 scale).

c Generate conclusion rules for each grid location eg. from the example grid for personality, the following rules may be generated for the first grid location:

IF desired characteristic is HAPPY
THEN suitable person may be A (n).
IF desired characteristic is SAD
THEN suitable person may be A (n).

The conclusion of each rule has associated with it a certainty factor (n) which indicates the strength of belief of the rule on the scale -1 to 1. The certainty factor is calculated from both the importance of the concept and the rating in the grid. For the above rules therefore, the first will have a low certainty factor as person A is rated as 5 (ie. sad) for the happy/sad construct, and the second will have a higher certainty factor.

d Generate intermediate rules from implications in the entailment graph (ie. between constructs), again with a certainty factor associated with each conclusion, based on the strength of the implication eg.

IF co-operation is CO-OPERATIVE

THEN motivation is MOTIVATED (n)

ie. if an element possesses the 'co-operative' trait, then they are likely (to degree n) to possess the 'motivated' trait.

e Review of the rules by the expert, with re-examination of the ratings in the grid where necessary, and use of the laddering technique where this helps to identify sources of conflict.

f Test the knowledge base ie. for each construct pair, the expert enters the desired trait, and the system suggests the elements which best fit, according to the rating grid.

g Further refinement by eg. matching scores, adding new traits to increase distinctions and subsequent re-generation of the entailment graph (based on PEGASUS), elicitation of new elements etc.

Like the induction of rules from examples, ETS provides a starting point for further elicitation and refinement of the prototype. The technique of starting with elicitation of the grid provides an aid to eliciting knowledge which is usually difficult to articulate ie. concepts which are relevant to the problem-solving process, and relationships between constructs which may not always be obvious even to the experts, defined automatically from grid ratings.

2.2 Conclusions

In addition to specific conclusions reached about the techniques discussed in this section, some general conclusions can be reached concerning both problems with the approaches taken and similarities and/or differences with the methods proposed in this thesis.

Firstly, the approaches taken are mostly isolated techniques to carry out specific tasks, with no suggestion as to how they fit into the whole process of knowledge elicitation. Concept sorting for example has been proved to be a useful technique for discovering how an individual perceives the problem space, and the differences between the perceptions of two individuals, but this does not constitute a method for knowledge elicitation as such.

Exceptions to this are those techniques which result in a rule-base eg. the automated techniques of rule induction and repertory grid analysis, but even so, there is no indication of how to go about the elicitation of concepts, attributes etc. from the expert. Such methods are therefore incomplete, although they result in a prototype expert system. In addition, the knowledge analysis scenario of Breuker and Weilinga which expands on Grover's domain analysis, and covers many aspects of the system development process, and the ERA technique of Addis which results in a logical model of the data associated with the expert task are closer to being complete methods.

Secondly, most of the techniques discussed take a functional approach to elicitation, working towards the formulation of the physical representation of the expert knowledge eg. a rule base. The problem associated with this approach is that the data, or concepts on which the representation is based may not be fully understood by the knowledge engineer, possibly resulting in conflicting rules, missed rules etc. If the domain was studied first, in such a way that the concepts understood by the expert were defined and structured, then such problems may be reduced. An exception is the method of Grover, which lays emphasis on a 'domain definition' phase at the beginning of the knowledge elicitation process. Even this approach however lays more emphasis on the background to the processes within the domain as opposed to the underlying data or concepts. The relational modelling approach of ERA satisfies the need to first analyse the data of the domain, but it is not made clear how the resulting sfd graph could then be used to build a knowledge base.

Another major problem posed by most of the techniques discussed is the lack of consideration and involvement of users of the resulting expert system, in the elicitation stage.

Where the users are the experts, this point is not so important, as the experts must be involved in the elicitation stage but where the intended users are of a different level of ability to the experts, then the issue of user independence becomes important.

The intention of an expert system to aid the latter type of user would be to enable them to carry out tasks they could not previously perform without consulting, or referring the problem to, an expert. The aim should therefore be towards a system which will produce the same answers to a specific problem, regardless of the user. In order to achieve this, the input to the system must be defined at the level of understanding of the intended users, and thus involvement of these users during the elicitation stage is of utmost importance.

Related to the involvement of users, ie. experts or non-experts, is the issue of graphical and written techniques for communication and documentation. Grover's domain definition phase would possibly be improved by the addition of a graphical technique to represent the data defined. The idea of node-link graphs is used in several approaches, eg. in the 'teachback' interviewing technique, but the inclusion of processes reduces the ability of these graphs to describe the domain data or concepts, becoming more of a technique for describing the rules. Graphical techniques which helped to describe the domain itself could be used to elicit the level of understanding of the users in addition to the experts.

The above problem can be associated with the issue of logical modelling, an approach only strictly taken by one of the methods discussed ie. ERA. Some of the techniques eg. teachback interviewing and repertory grid analysis, do work towards some form of intermediate representation (ie. a representation which lies between the experts formulation and the machine representation of the knowledge) which makes more explicit the knowledge elicited. Such approaches therefore recognise the need for an explicit formulation of the knowledge which is independent of the implementation, as does the logical modelling approach of this thesis.

The use of limited data which is a central issue in the ideas behind the proposed method for knowledge elicitation, is necessary for certain techniques such as rule induction from

examples. Generally however, the form of data to be input to the system is not taken into account in the methods discussed and therefore they do not advocate the use of either limited or continuous data.

In terms of the definition of a structured method in systems analysis ie.

* logical/physical split,
* graphical tools,
* user involvement,

few of the techniques covered apart from ERA could be described as structured. Some have one of the above elements eg. repertory grid analysis produces a form of logical model, and also can involve users in order to find the different perceptions of the problem; teachback interviewing results in the node-link network which although not entirely logical, is an aid to analyst-user (expert) communication and Knowledge Analysis includes a 'User Analysis' stage in order to produce a suitable user interface.

CHAPTER 3 THE EXPLICITNESS BOUNDARY

3.1 Introduction and Definition of Limited Data

The aims of this chapter can be outlined as :

i) justification of the explicitness boundary as a 'limited data' boundary,

ii) investigation of the use of the data flow technique for defining the boundary.

Firstly however, the definition of 'limited data' needs to be clarified. The way in which any object or concept is defined depends partially on the particular view taken of that concept. Bunt [1985] considers the concept 'water' to be continuous in nature if considered 'naively' but discrete if considered as e.g. a collection of H_2O molecules. This type of classification is based on whether or not the physical properties of water are considered.

Similarly, different views can be taken of the data associated with, or describing a concept. For example, humans can detect continuously varying data e.g. temperature or speed, but use that data in one of two forms. If the data are used in e.g. feedback mechanisms regulating body temperature, then the actual values of temperature will be used for such processes. Similarly in order to perform calculations, a human will need to use the specific values of temperature as measured, which belong to an infinite data set. The form of the data in both of these cases can be described as quantitative and is comparable to the idea of the 'continuous' nature of water above. However, for purposes of communication, decision-making etc., a qualitative form of the data is more likely to be used. In order to consciously evaluate temperature for some purpose e.g. whether or not to take a warm coat, descriptive terms from a limited data set would be used e.g. 'freezing', 'very cold', 'warm' etc.

The view taken of data therefore depends partly on the use which is to be made of it e.g. for algorithmic processing, verbal communication, reasoning etc. The diagram of fig. 3.1 identifies three general types of processes which use the two types of data discussed above.

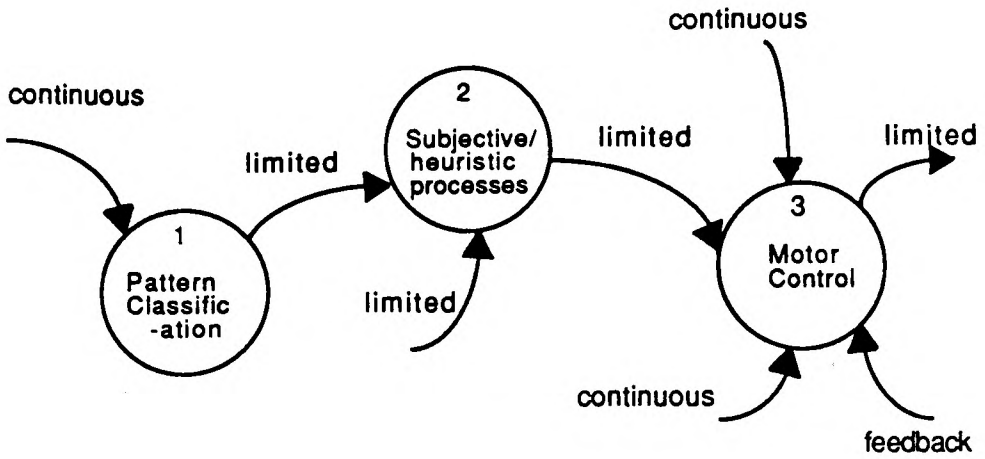


Fig. 3.1 Continuous and limited data and associated processes

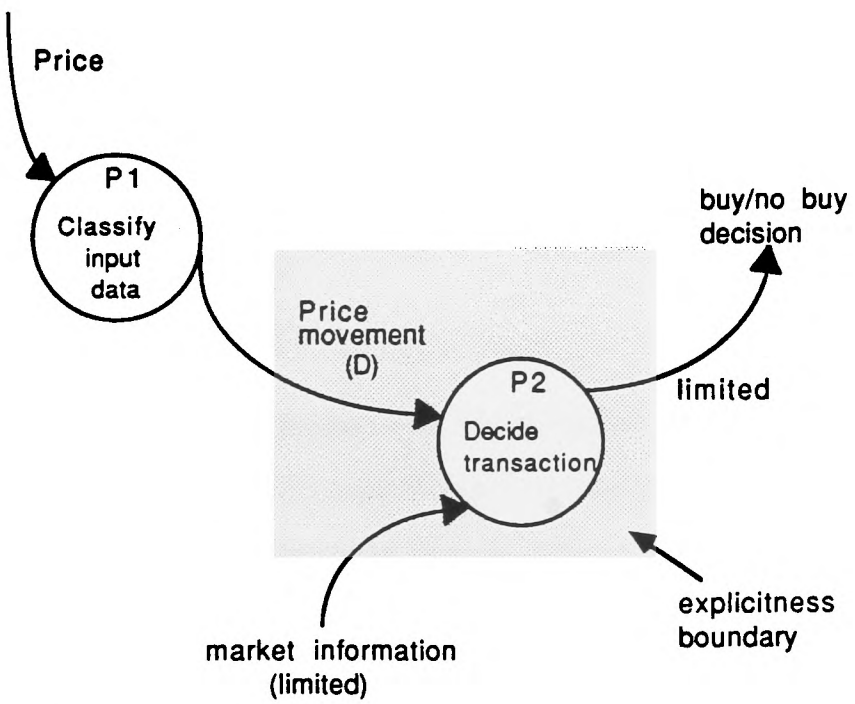


Fig. 3.2 Limited and continuous data and processes of a stock market problem

These data types are labelled as continuous and limited for the purposes of this diagram, in order to convey the idea of continuously varying, possibly numeric data, and descriptive data respectively.

Process 1 represents a pattern classifying process which converts continuously varying data into limited sets of values e.g. temperatures into qualitative terms such as hot, cold etc. There may be definite 'rules' for this transformation in which case such a process could be included as part of an expert system knowledge base, or the process could be more subjective or intuitive in which case the classifier would be a human.

Process 2 consists of more subjective or heuristic processes which use the limited data sets resulting from process 1 in order to result in more limited data values as output. These are the types of process which are usually associated with expert systems, and are thus the part of the expert knowledge which needs to be made explicit. Examples of this type of process or knowledge include the following 'rules':

if 'temperature hot'
then 'heating too high'

if 'temperature cold'
then 'heating too low'

where 'temperature hot' and 'temperature cold' form the limited data resulting from process 1, and 'heating too high' and 'heating too low' form the limited data output of process 2. The latter data elements could be used in further processes of the same type e.g.

if 'heating too high'
then 'turn heating down'
...

Process 3 represents reflex actions e.g. automatic avoidance of pain and other instinctive reactions. The data input to this type of process may be continuous in nature e.g. intense

heat which causes an automatic response due to pain, or qualitative, belonging to a limited set of values e.g. a red light which causes the response to stop.

It is now easier to see the distinction between the terms *continuous* and *limited* when applied to data. Continuous data can be defined as data which are continuously varying on an infinite scale whereas limited data are the opposite i.e. defined and part of a small data set. The term discrete is not used in this context as it introduces confusion as to where the boundary lies between continuity and discreteness and does not convey the idea of a small finite data set. For example, the set of all integers can be described as discrete but cannot be described as a small finite data set.

The terms continuous and limited can further be amplified in terms of quantitative and qualitative data. Quantitative data are concerned with the concepts of quantity and measurement (of size), whereas qualitative data are concerned with a small set of qualities. A quantified concept is measurable (often by an instrument) e.g. in inches, degrees etc., whereas a qualified concept is one which is described or characterised in a particular way e.g. by a limited number of descriptive terms.

Consider the following example of a system to capture the expertise of a stock market expert, advising on buying and selling. The data input to such a system will be partially qualitative e.g. 'is the market bullish, bearish, stagnant ?', 'nature of stock in question : European, Japanese etc. ?'; all data representing memorised 'human' concepts. Other data however, for instance buying and selling prices will normally be quoted as numeric data to several decimal places and are not therefore in a qualitative form. In order to match prices against stored concepts a pattern classifier is required (fig. 3.2) which analyses the input to produce chunks of information for conceptual processing. This might be done e.g. by a simple process of banding the input data (e.g. rising gently, rising rapidly, steady, erratic etc.), by some reference to average industry prices or by some more sophisticated means.

A production rule form for such a system might contain the following rules :

IF $P_{NEW} - P_{OLD} > 4\% \times P_{NEW}$ THEN RISING-RAPIDLY 1

IF $4\% \times P_{NEW} > P_{NEW} - P_{OLD} > 2\% \times P_{NEW}$ THEN RISING-GENTLY 2

IF RISING-RAPIDLY AND MARKET BULLISH THEN BUY-SOME 3

IF RISING-RAPIDLY AND MARKET STAGNANT THEN BUY-MANY 4

The system divides into two types of rule : rules like 3 and 4 define the conceptual processing with limited data on both sides of the rule whereas rules such as 1 and 2 define the interface at the explicitness boundary as they perform operations on continuous variables (i.e. P_{OLD}, P_{NEW}) in order to produce values belonging to a limited data set (D in fig. 3.2) which describes the situation. The method of evaluation of the left-hand side of the latter type of rule does not need to be explained as it lies outside the boundary and does not need to be made explicit.

The concept of limited data then becomes more clearly definable as :

'data which are described or characterised by a set of finite terms each of which has definite semantic content with respect to the domain in question.'

Section 3.2 discusses the justification of the limited data boundary, based on the above definition of such data, and section 3.3 looks at the suitability of the data flow technique for representing the different processes within the expert task and thus identifying the explicitness boundary.

3.2 Justification of the Limited Data Boundary

3.21 Class of Problems

In the previous section, the meaning of limited data was defined in terms of the qualities of those data. The resulting definition implies that the heuristic processes which make up the expertise of a human expert use only descriptive values from a small set of values as input data, and produce as output a similar type of data. For such expertise, the explicitness boundary will always be defined outside such processes as all data entering the expert system will be naturally limited. The set of expert problems for which the expertise uses data in a limited form as defined thus provides one justification for the method of elicitation proposed.

3.22 Implications of the 'Human Window'

In this section the argument that a small memory as required for a process to be within the human window (as outlined in section 1.41) implies the use of a limited data set is investigated. The implication of this is that in order for a process to be explicitly defined and therefore be included in the explicit knowledge base of the expert system it must deal only with such limited data.

The importance of defining the data required by processes then becomes a necessary part of knowledge elicitation, in order to define exactly which processes become part of the knowledge base. Data exist however in more than one form, i.e. a data item is not always derived from a small data set, e.g. a real number. It is important therefore to first consider the types of data which exist, and to identify how each should be treated with respect to the explicitness boundary.

3.22.1 Qualitative Data

Taking into account the nature of qualitative data, an expert process which deals with this type of data will have only limited data as input and output. The values which make up the

data set will be either descriptive terms which describe a concept e.g. hot, warm, cold for the concept temperature, or distinct (numeric) values which either the expert recognises as significant or important for the decision making process, or which result from non-algorithmic processes. These numeric values become equivalent to descriptive terms as they have a definite meaning attached to them as opposed to numeric values as defined by quantitative data.

For example, process 4 of fig. 3.3 (constructed from the ARIES Equity Selection documentation [Logica, 1986]) represents a process which involves data of both the types described above. The process takes as input, qualitative data of the first type, i.e. descriptive terms representing user opinions on e.g. industry prospects, company prospects etc. It produces as output a series of values from a small finite set of numbers representing loadings which match these opinions.

The process can be exploded as in fig. 3.4 which shows the various processes involved in evaluating the loading to apply for e.g. industry prospects. Processes 4.1, and 4.2 of this data flow diagram illustrate situations where both the input and output consists of descriptive terms. For each of a series of questions which may be asked, the users are given a limited (small) set of values from which to select an answer e.g.

"Are the industry prospects ...

- long term bad, short term good,
- long term good, short term bad,
- bad,
- about average,
- good,
- more detailed input necessary."

The rules within process 4.2 then combine the answers resulting from 4.1 in order to reach an overall opinion about industry prospects. This opinion is transformed into a loading figure

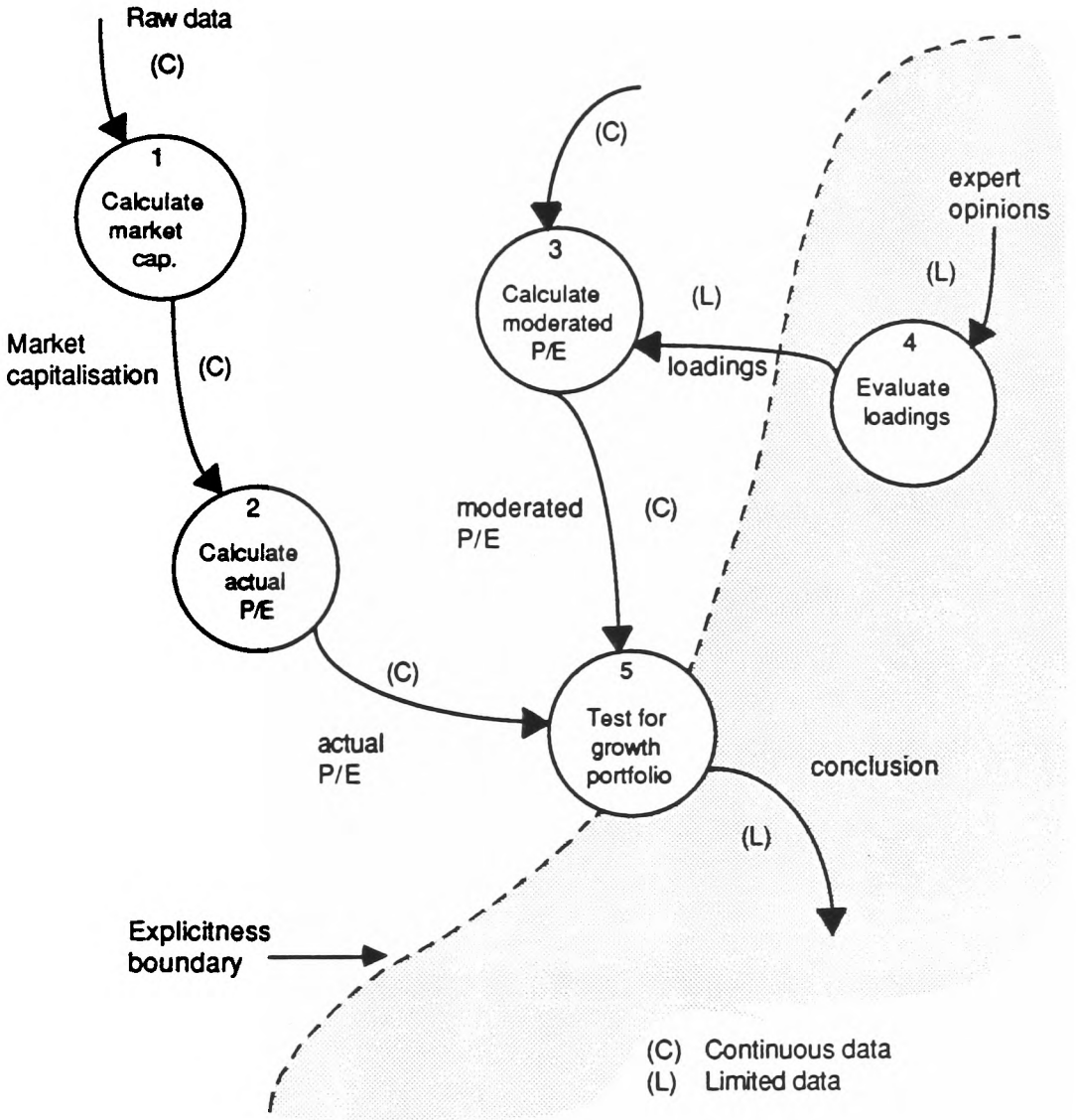


Fig. 3.3 Equity selection data and processes

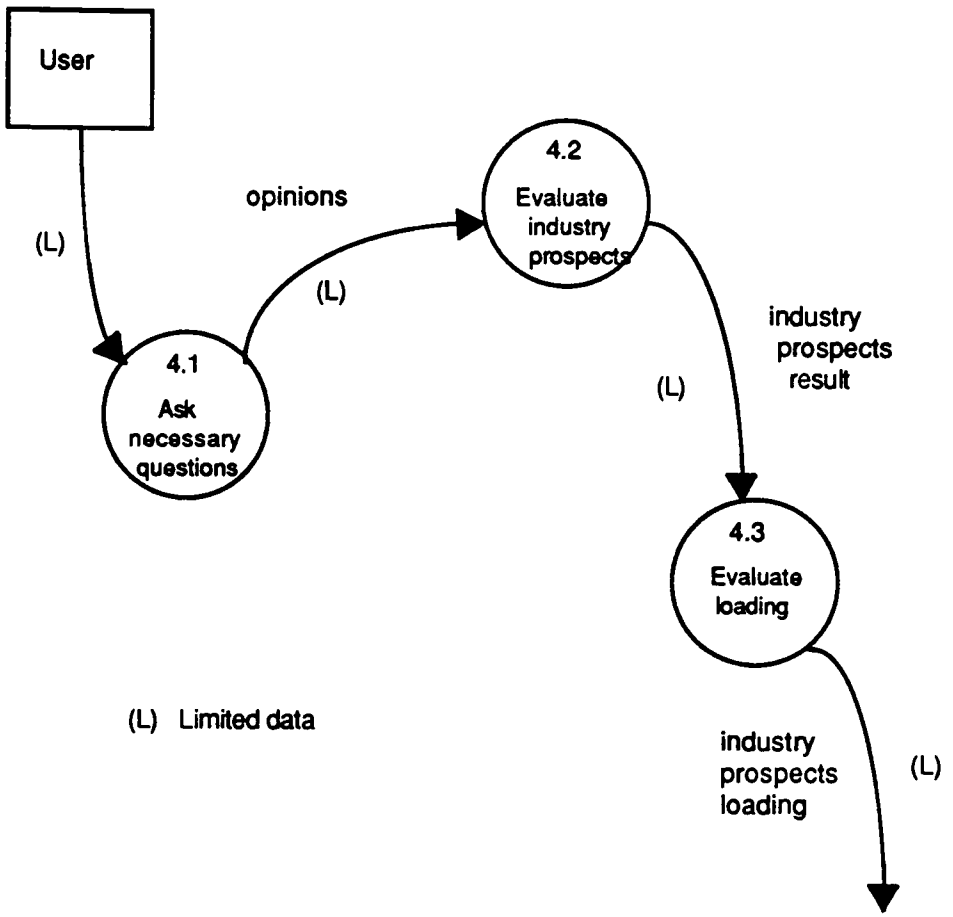


Fig. 3.4 Detail of process 4, fig. 3.3 - Evaluation of loading figure

by process 4.3, thus producing output in a form acceptable for process 3. The loadings are evaluated using rules of the form

IF industry-prospects are 'Good'

THEN industry-prospects-loading = 20,

IF industry-prospects are 'About average'

THEN industry-prospects-loading = 0,

and are thus equivalent to the terms 'Good', 'About average' etc., i.e. they are basically labels and can therefore be treated as qualitative data although numeric values. It should be noted here that numbers are, in a case such as this, used only for convenience and thus although they have semantic content for this application, could not be said to have so in general terms.

Both the input to and output from a process of this kind is limited in nature i.e. values are taken from small data sets. Such processes are therefore explicit according to the human window constraint of small memory and simple processing, and the constraints of comprehensibility and communicability. As such they would lie completely within the explicitness boundary, as shown on figs. 3.3 and 3.4.

As another example, the continuous range of temperatures detectable by humans (or sensors) constitutes a very large data set but, for decision making as regards e.g. controlling central heating, only a small number of data elements or values need be remembered and thus used in the decision making process. Take the following set of rules :

IF t >70 THEN 'heating off'

IF t >60 AND t <70 THEN 'heating on, low'

IF t <60 THEN 'heating on, high'

the important values in the decision making process are 60 and 70. There are however, two forms in which the data can be input to a 'system' consisting of rules as above,

i) as actual temperatures e.g. 52 , 64 , 71,

ii) as judgments on the temperature e.g. >70 , >60 , <60 .

If the first of the above methods is used, then the input data are really quantitative in nature and the portion of the rule 't > 70' is a mathematical process. This example will therefore be discussed in the next section (3.22.2) on quantitative data. If however, the second of the above input methods is used, then the value selected by the user will automatically match the condition part of one of the rules. The portion 't > 70' can therefore be treated as a 'label' rather than as a process i.e. the idea of 't > 70' can be associated directly with the action 'heating off' in the same way as could a descriptive term e.g. 'too hot'.

In all of the above cases, the qualitative data which both exist within the explicitness boundary, and cross that boundary belong to a small data set. The constraint of the explicitness boundary can be clarified from this therefore, as a boundary within which all data are defined as limited data sets and across which only data in this form can pass.

Data in a limited form as defined, are naturally explicit i.e. a small data set satisfies the constraint of small memory, is comprehensible and communicable. Processes which either have this form of data as input, output or both will therefore lie either on or within the explicitness boundary. By clearly defining the data sets which are used by these processes, the identification of which processes should be included in the explicit part of the knowledge base is made an easier task.

As another example, consider an expert in statistical analysis carrying out a task to decide whether or not to treat a data set as deriving from a normal population. The decision process illustrated by fig. 3.5 takes as input data a mixture of qualitative and quantitative data. The explicitness boundary in this example incorporates processes 1, 2 and 7 and excludes processes 3 - 6. Processes 1 and 2 are concerned with the users opinions of the data sample and source population respectively, which are qualitative in nature. Processes 3 - 6 however are concerned with testing and plotting the actual data sample and are therefore dealing with quantitative data. The output of process 3, 'test data', would consist of a measurement of the evidence for non-normality of the data sample which although numeric could be

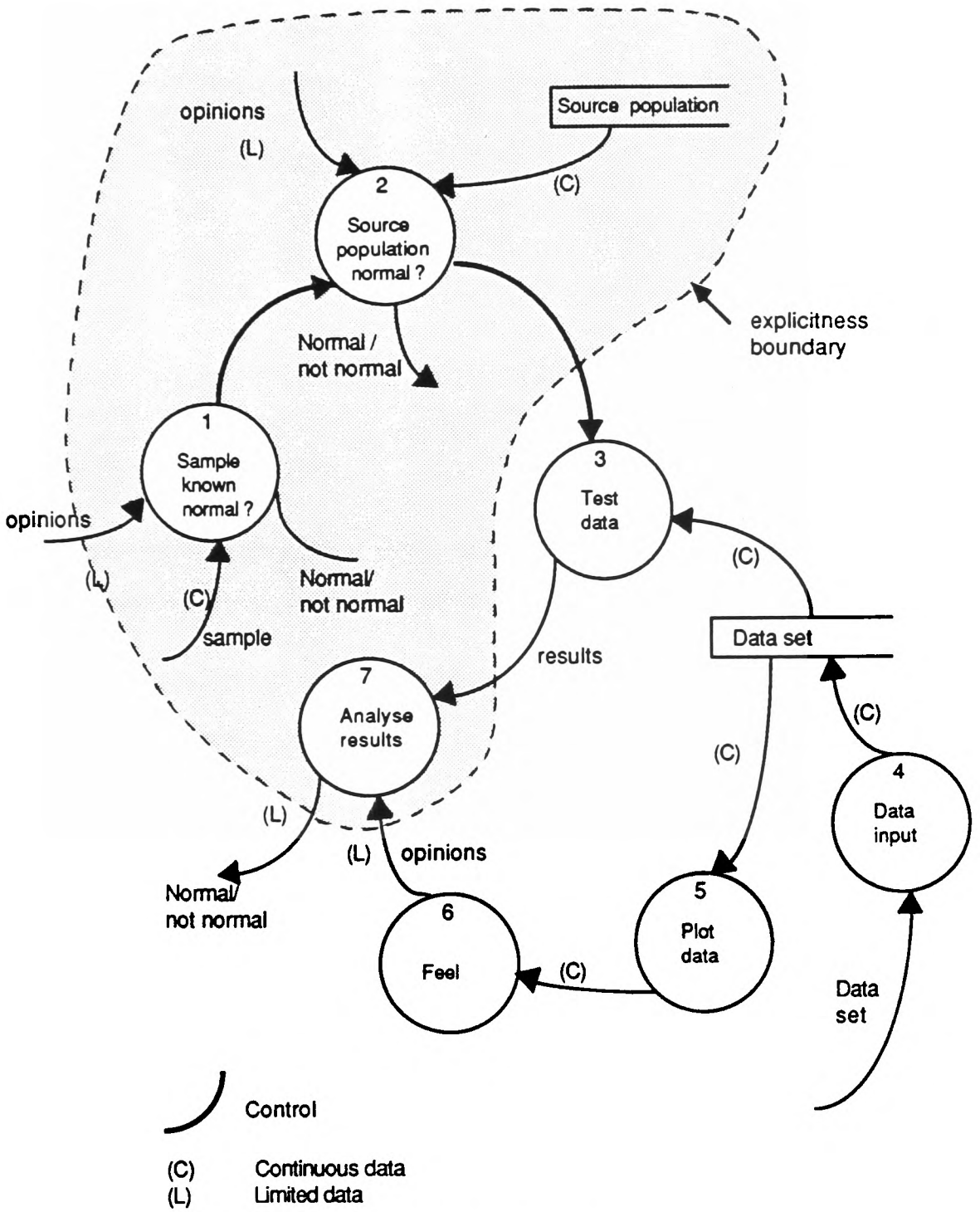


Fig. 3.5 Qualitative and quantitative data for 'decide normality'

considered as having the characteristics of limited data as it contains semantic content to the expert. Process 7 therefore lies within the boundary (or defines the boundary if the definition of limited data is interpreted more strictly). A selection of possible production rules representing the part of the system within the boundary are as follows :

a IF known-normal THEN result = normal

b IF source-population ... THEN result = normal

c .

d .

e IF source-population not-known THEN action = test-data

f IF result ....

g IF .... THEN normal

Rule e expresses the interface to the explicitness boundary, this time on the right of the rule. We are not concerned with how test-data calculates the results - this process is outside the boundary.

3.22.2 Quantitative Data

Data as used in expert tasks may also however exist in a quantitative form i.e. numeric data derived from instruments, financial data etc. In such cases, how can the explicitness constraint as described above for qualitative data be satisfied especially as the constraint of small memory is no longer satisfied ?

In terms of processing numeric data we can concentrate purely on binary operations as all higher level operations can be reduced to this form. There are two types of binary operations involving quantitative data, which can be identified as

i) comparisons,

ii) calculations.

i) Comparisons

Comparison processes such as 'i > x', 'i = x', 'i < x' may either be placed outside the explicitness boundary or may actually form the boundary i.e. the transformation between continuous and limited data, e.g.

IF i < x THEN 'x-value high'

IF i = x THEN 'x-value medium'

etc.

In the latter case the process is acting as a pattern classifier. The values on the right-hand side of the above rules would be recognised by the explicit processes within the boundary and therefore the process would lie outside the explicitness boundary as in fig. 3.6. The process would only lie within the boundary if the actual process of deciding i < x was made explicit.

Alternatively, the process which compares i with x may be part of the reasoning process itself as in the central heating example of the previous section where the input data are in the form of actual temperatures. This process can be represented in two ways i.e. as one process or as two. If represented as one process, as in fig. 3.7, then the explicitness boundary will be shown to go through the process, as the first half of the rule deals with quantitative data, and the second with qualitative data.

This representation shows clearly that this process forms part of the explicitness boundary even though the whole rule will obviously lie within the knowledge base. In order to make the boundary clearer however, the process can be split into two parts (fig. 3.8), the first being the part which does not need to be made explicit (i.e. we do not care how t>70 is evaluated') and the second the part which we do want to be explicit i.e. how the decision 'turn heating off' is made.

In this example, the process 't > 70' is a straightforward operation with no extra processing to be specified and would therefore be represented as part of a rule within the knowledge base.

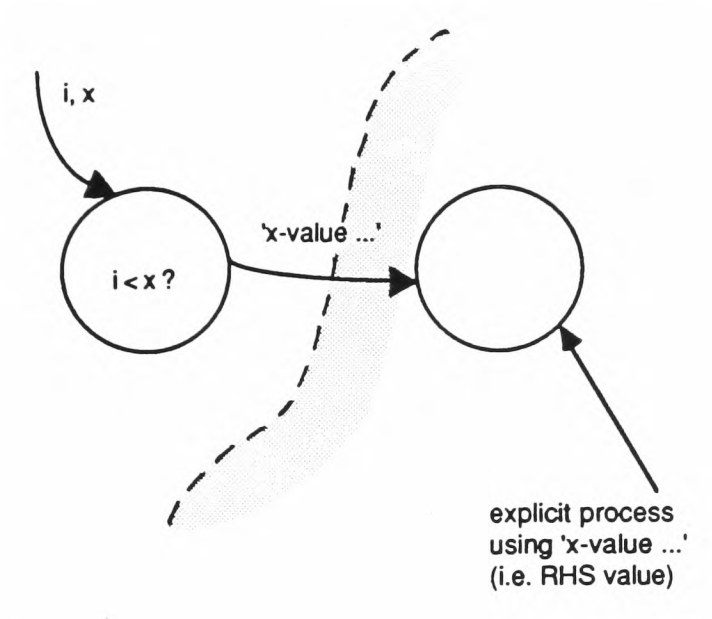


Fig. 3.6 Process as pattern classifier converting continuous data to limited data outside the explicitness boundary

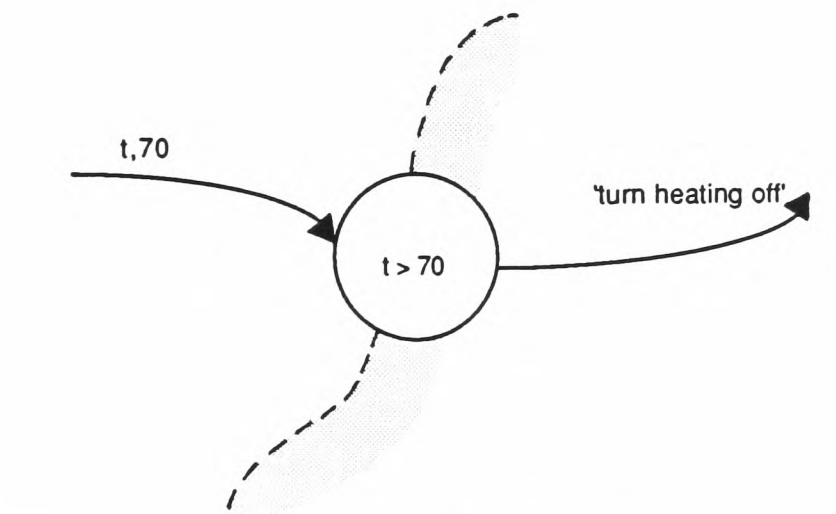


Fig. 3.7 Process forming part of the explicitness boundary

In a more complicated example where the process required extra coding the separation into processes as above may be necessary.

Process 5 of fig. 3.3 forms another example of a process which forms or lies on the explicitness boundary. The rule which makes up this process takes the form

IF actual P/E < moderated P/E

THEN 'buy for growth portfolio'

The values 'actual P/E' and 'moderated P/E' both result from purely algorithmic processes and are not therefore derived from small data sets. Also, the comparison itself is not explicitly defined, and therefore this portion of the rule cannot theoretically lie within the explicitness boundary. As for the above example, there are two ways of representing this, as shown in fig. 3.9.

Process 5 (fig. 3.3) could be made more explicit if required i.e. if it was desired that the actual process of deciding whether 'actual P/E' < 'moderated P/E' be made explicit. In order to satisfy the concept of explicitness, one of the constraints to be satisfied is that the representation of the process should lie within Michie's human window. The two forms shown in figs. 3.10a and 3.10b of representing the process A < B lie at either end of the scale of memory usage and processing ability and do not therefore satisfy that constraint.

The first of these figures (3.10a) shows the process A < B as purely a look-up table where all combinations of A and B are stored with the result Y(es) or N(o). For all possible combinations of A and B (i.e. where both belong to the set of real numbers), the look up table is clearly too large to be memorised. The second figure (3.10b) represents the process as a calculation which subtracts B from A and considers the sign of the result. For very large numbers, or those to a large number of decimal places, the calculation power required to carry out the process in one step is too large to be carried in the head.

In order therefore to satisfy the constraint of small memory and limited processing ability, a compromise between the two methods of representation needs to be found. For this

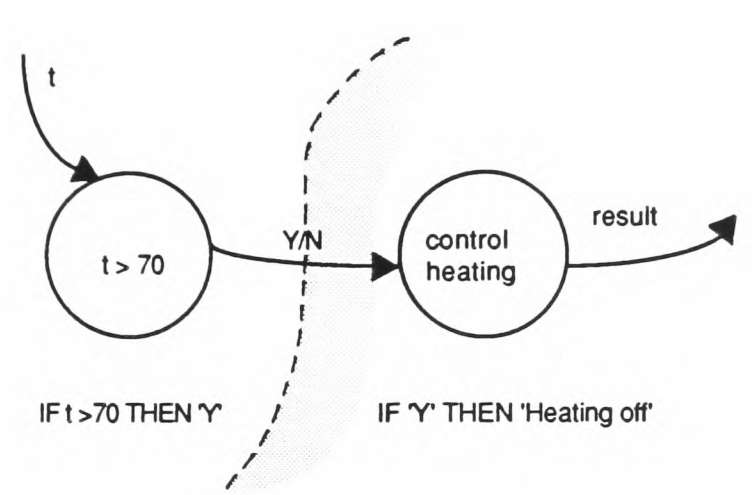


Fig. 3.8 Making the process of 3.7 explicit

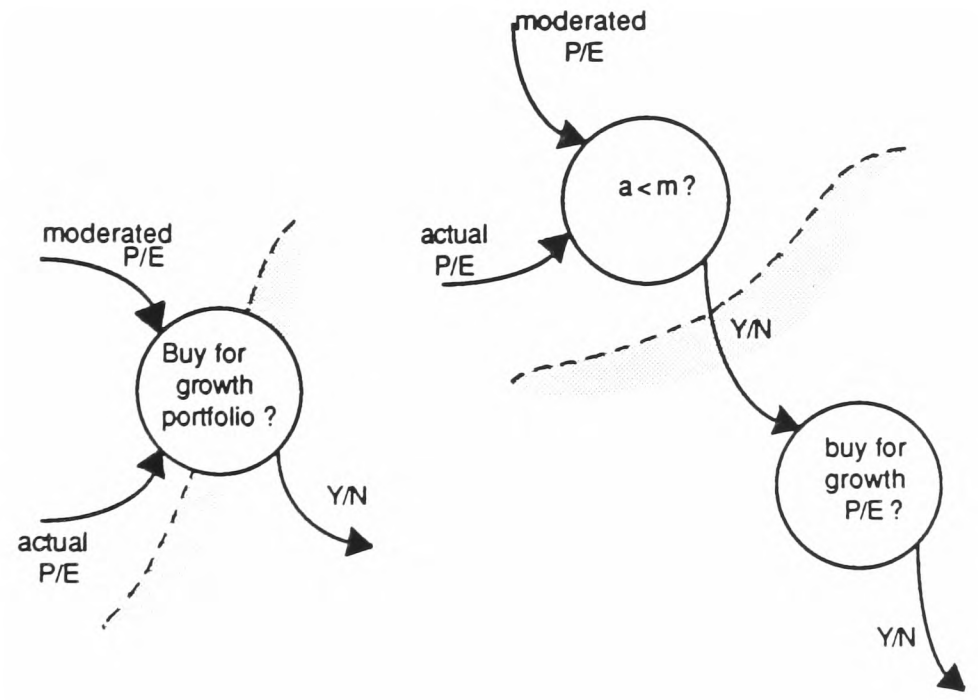


Fig. 3.9 Process lying on the explicitness boundary

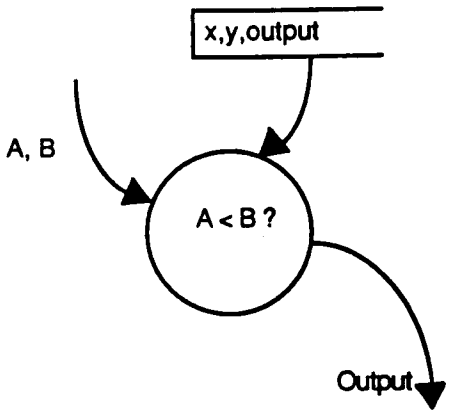


Fig. 3.10a A < B as a look-up process

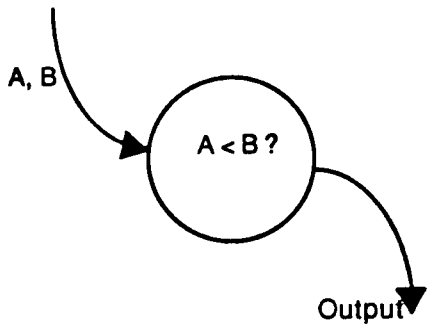


Fig. 3.10b A < B as a calculation

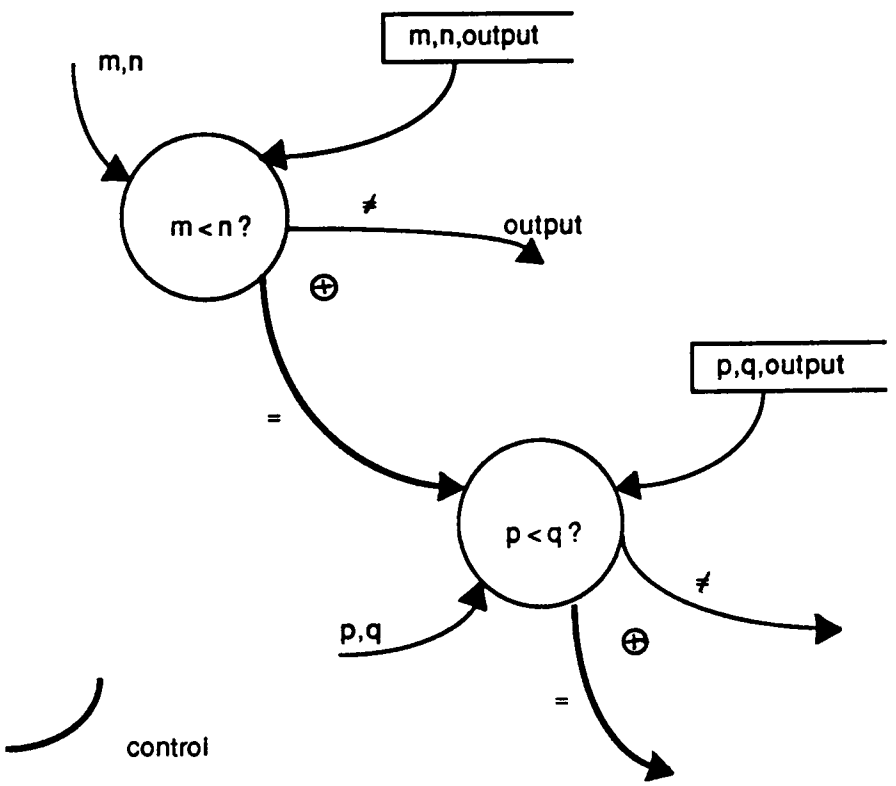


Fig. 3.11 Explicit representation of the comparison process A < B

particular problem, the 'human' method incorporates the need to represent each of the two numbers A and B as a series of digits. The method of comparing A and B then involves first comparing the number of digits in each number (i.e. before the decimal point) and only if these are equal, comparing the digits themselves. As each digit can only take one of 10 values (i.e. 0 - 9), the comparison which needs to be made can then be represented by a small look-up table, requiring only a small amount of memory. The explicit representation of the comparison A < B is shown in fig. 3.11 where m,n represent the number of digits in each number, and p,q each pair of digits.

ii) Calculations

The data associated with purely algorithmic processes of the second category i.e. calculations are not confined to small data sets e.g. the processes 1, 2 and 3 in fig. 3.3 where the input and output is purely numeric. Two views of such processes can be taken :

a) Such processes are not part of the expertise we are interested in and therefore do not need to be part of the knowledge base. They do not therefore need to be made explicit and thus the small data set constraint is not important. Such processes will lie completely outside the boundary, which will be formed by a later process using the output from these algorithmic processes e.g. process 5 (fig. 3.3).

   As an example, for the decision as to whether to buy or not buy an equity, the method of calculation of the various P/E ratio's is not part of the expertise as such. The values which result from these calculations are used however in order to make judgments about the company performance, future etc. The actual P/E calculations will therefore lie outside the explicitness boundary i.e. processes 1, 2 and 3 of fig. 3.3.

b) It may be desirable to make these processes explicit however, if for example the aim is to teach mathematics. As in the previous case of making the comparison of two

numbers explicit, in order to force an algorithmic process inside the boundary, it is necessary to pay attention to the form of the data input to the process.

In summary, qualitative data which are explicit by nature pose no problem when defining the explicitness boundary. Processes which use data of this type will always satisfy the constraint of small memory and will therefore lie within (both input and output are qualitative) or on (input or output only is qualitative) the boundary. Quantitative data however, does not satisfy the constraint of small memory and thus requires different treatment as regards the explicitness boundary.

Comparison type operations treat quantitative data as labels which may be meaningful and thus belong to a small data set and in which case, can be considered as explicit, whereas calculations which produce numeric data belonging to large or infinite data sets cannot. The latter type of process is therefore considered either to be outside the domain of interest as concerns the expertise within the explicit knowledge base, or needs to be made explicit either by explicit procedures or by the use of small data sets.

The explicitness boundary is therefore partly decided by the nature of the data, and is partly a design decision. The action of defining the type of data used by processes which make up the expert task has the result of helping to define the boundary by means of deciding which processes are naturally explicit, which need to be made explicit and which may remain outside the explictness boundary.

By making the data associated with a process explicit, the process itself becomes explicit and thus satisfies the constraints required to be within the explicitness boundary. A process may indirectly deal with a very large data set e.g. the set of real numbers (for an addition process) but the data explicitly required by the process may belong to a very much smaller, limited data set e.g. the set of digits 0 - 9.

3.23 User Independence

3.23.1 Definition and Implications of User Independence

The issue of user independence is concerned with the robustness and consistency of the expert system in the hands of the users. Robustness in this context is concerned with the reliability of the system in terms of accuracy and consistency. For a given set of users of the same level of ability, (whether expert or novice), and the same set of input data, the individual results output by the system should be the same. In conventional data processing systems, where the data are strictly defined i.e. there is no interpretation of the problem to be carried out by the user, this question does not arise. Typically however, problems solved by experts are more subjective in nature, each problem requiring some form of analysis and interpretation before the input to the system can be defined. As an example, the input stage of the ARIES Fire Risk system (discussed in chapter 6) includes the interpretation of the surveyors' report on the premises being assessed. Different users will place different interpretations on this report, for example the degree of importance of certain aspects of the building or trade. The problem of consistency between different users therefore becomes one which needs to be resolved if the expert system is to offer any advantages to the experts or the users.

If the users of the expert system are the experts, then the problem of user independence does not arise as much as if the users are of a different level of ability. As experts in the domain, the former type of user is familiar with the terms or jargon used in the problem solving task, which will also become part of the user interface to the expert system. The latter type of user will not however have the same level of familiarity with the domain and will therefore find using the system more difficult, although this is a problem which could possibly be overcome by training. A problem which could not be overcome so easily by training is concerned with discriminative ability i.e. experts at a task may be able to discriminate more finely (and thus reach a wider range of decisions) than users i.e. discrimination is often a significant element of expertise. In order to be an expert in a given domain, an individual

must be able to carry out all stages of the problem solving process, including the initial interpretation of the problem case, and extraction of the relevant details or 'data' which forms the input to the expert system. This is the part of the expertise which is most concerned with making discriminations. A more naive user may however have problems in carrying out the latter task, and thus will require some help from the system at the input stage.

If the above case applies then the user is likely to input the wrong values for data items at the input stage, thus producing incorrect results when the data are processed by the system,. The robustness of the expert system in the hands of the users is thus greatly reduced if insufficient attention is paid to the ability of the users when designing the expert system. This ability is concerned only with the initial interpretation of the problem rather than knowledge of how to actually solve the problem.

To summarise, the aim of making expert systems user independent is to reduce variability and thus increase consistency and accuracy of the system.

For the purposes of user variability, the rules or functions within the knowledge base of an expert system can be considered as a static 'black box' i.e. for the same input they will always produce the same output in the same way as a conventional data processing system. The level of variability and hence reliability is therefore related to the degree of reliability attached to the input data as opposed to the functions.

Involvement of the target users of the system at the early stages of system development i.e. in defining the explicitness boundary in terms of the qualitative data sets should ensure an improvement in the degree of user independence of the system.

3.23.2 Modes of User Input

If the input data are quantitative in nature e.g. financial data, data from sensors etc. then the degree of variability at the data input stage of using the expert system should be almost negligible. For any form of input where the data are not strictly defined however, i.e.

qualitative data or quantitative data which are open to some estimation or interpretation, then there are two extreme methods of asking the user for the input.

At one end of the range we have the use of continuous scales where the user is asked to input his judgement on a data item or concept within the domain on e.g. a scale of 0 - 10 on which decimal places to one point are allowed. The user is being asked to make very fine discriminations on a numeric scale of, in this case, 101 points which may be a straightforward exercise to those who are numerically minded, but is a difficult task for those who are not. The method can be made easier and less numeric by giving the user a continuous 'line' on which to place their judgement. The user is still having to make discriminations of the same granularity however, and in addition, the input has to be interpreted in some way 'behind the scenes'.

At the other end of the range of methods, the scale for input as used above can be divided into a small number of values e.g. a three point scale [Greenacre 1984] where users are asked whether they either a) agree, b) are undecided or c) disagree with a set of statements.

Dependent upon the domain and the problem however, the number of points required on the n-point scale will vary. The above 3-point scale may for example be found to be too restricting and so to increase the freedom of choice extra points e.g. agree slightly and disagree slightly may be introduced. As opposed to continuous scales this method offers certain advantages and makes the task of discrimination easier for the user. The user can select a value which fits the data as opposed to having to make a decision as to where on the scale to place his judgement, and the discriminations he has to make are not so fine. In addition, each value is given a descriptive name making the input process more meaningful to the user than does the use of numeric scales.

In terms of user independence, the latter method of input is the one most likely to produce robust systems as the ability of users to make fine discriminations on continuous scales will almost certainly vary with ability. If the users are faced with a set of descriptive values to

select from, the degree of error should be reduced if only because of the increase in 'grainsize' of the discriminations they have to make.

If using limited data however, there are still certain problems which need to be resolved, concerned with how many points on the scale should be defined. This will be dependent upon

i) the level of ability of the user set of the system to make discriminations, as compared to the expert ability to carry out the same task, and

ii) the required level of reliability, or conversely the acceptable level of variance or error

.

In some cases, the number of points on the scale, or bands may be obvious to the expert e.g. a flame can either be blue, yellow, orange or a variant on one or more of these making e.g. a 9 point scale. This will either be due to a physical constraint e.g. a flame can only ever be one of those colours, or it may be the case that those colours are the only ones relevant to the expert process. In either case it may be necessary to carry out tests to see whether users can discriminate between flame colour as well as the experts, and hence the number of options which should be presented to the user. It may be the case that users can only recognise say 5, 6 or 7 different flame colours, and thus certain functions within the knowledge base would never be activated and consistency between users would become a problem.

In a situation as above, there are two methods of arriving at the optimum number of bands for the factor 'flame colour'. Probably the most practical is for the expert to define the bands, then for tests to be carried out with the users to see how well they can discriminate on the given scale. This method is discussed further in chapter 6.

It may be desirable however, to determine the number of bands by analysing user data i.e. to find the 'natural' bands which the user set in general can discriminate on. The users cannot therefore be given a classification scale with which to classify examples, but instead can be asked to rank a set of N examples according to a factor such as 'flame colour' above. From

this set of rankings, it is possible to find bands of rankings e.g. by using blockmodelling techniques which are explained in more detail in 3.23.3, where each band becomes a point on the scale.

If we take the two extreme cases where the number of bands or classes is very large or very small, we can see how the level of error changes. The highest number of bands that can be defined for a set of examples in the domain, is equivalent to the number of examples in that set. In order to achieve such a set of scale points however, all observers would have to attribute the same rankings to examples. This is however unlikely, and the degree of variability in the rankings is indicative of the degree of variation or error which would occur were this scale to be used as the method of data input for that factor.

The opposite case, is where the example set is divided into two subsets where one subset is always or generally ranked higher than the other. In the latter case, the two subsets would be chosen where the lowest degree of error occurred. This degree of error would again be indicative of the error which could be expected at the data input stage. The degree of variability would certainly be lower than that of the above example, as the user only has two bands to choose from as opposed to many.

This argument can be generalised in that as the number of subsets is increased and thus the size or 'width' of each is decreased, so the degree of error increases. Hence, if the level of error acceptable is decided first, then the optimum number of bands into which the scale for a factor should be divided can be found.

In the next section, an example shows the use of the blockmodelling technique to find the 'natural' banding from user rankings, and illustrates the increase in error with the increase in the number of bands.

3.23.3 Use of the Blockmodelling Technique

As discussed in the previous section the limited data sets would in most cases be defined by the experts as part of the elicitation process. The testing stage of the elicitation process

would then involve testing the ability of the users to make the necessary discriminations on the same sets of test cases as the experts. If the users can carry out this task satisfactorily then the system should reach the same conclusions whoever the user, and it can be said that user independence has been achieved.

An alternative method is to define the input data sets automatically from user data using methods based on blockmodelling techniques. The following example, carried out as an exercise to demonstrate this technique, and based on rankings of students by lecturers outlines the method.

Lecturers were asked to rank the set of students on the concepts 'motivation' and 'ability', using a scale of 1..n where n is the number of students in the class.

Various operations were then carried out on the resultant user x test case matrix, in order to see how well the lecturers could agree on the use of the discriminating factors they were asked to rank on. The exercise was carried out on both the whole matrix, and on smaller matrices representing 10 test cases at a time (the results of which were then collated). For illustrative purposes, one of the latter will be used during discussion of this exercise. Fig. 3.12, is an example matrix representing how 3 lecturers ranked 10 students.

From this initial matrix, a rank x test case matrix was built up representing the combined rankings of all users for these ten test cases. Fig. 3.13 shows this matrix, with all zero rows omitted. The matrix was built up then converted to a probability matrix, thus, as there were 3 users in this exercise, a '1' in any position indicates that all 3 users ranked this student the same, a '0.67' represents the agreement of 2 users, and a '0.33' represents the ranking of just one user. The sum of every column in the matrix is therefore equal to 1.

From this matrix, we can see the degree of total agreement between the users, by the number of '1's' in the matrix. We cannot however, see how closely they have agreed otherwise. By rearranging the above matrix, so that all non zero elements are along the leading diagonal of the matrix, we should be able to see the degree of this agreement, and

	Students									
	30	**7**	**26**	**24**	**6**	**36**	**20**	**35**	**44**	**45**
Lecturers										
1	10	1	33	27	44	9	26	32	11	46
2	18	1	44	25	26	33	46	31	8	47
3	19	1	47	28	35	29	44	33	37	23

Fig. 3.12 Initial User / Test Case matrix

	30	**7**	**26**	**24**	**6**	**36**	**20**	**35**	**44**	**45**
1	0	1.00	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0.33	0
9	0	0	0	0	0	0.33	0	0	0	0
10	0.33	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0.33	0
18	0.33	0	0	0	0	0	0	0	0	0
19	0.33	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0.33
25	0	0	0	0.33	0	0	0	0	0	0
26	0	0	0	0	0.33	0	0.33	0	0	0
27	0	0	0	0.33	0	0	0	0	0	0
28	0	0	0	0.33	0	0	0	0	0	0
29	0	0	0	0	0	0.33	0	0	0	0
31	0	0	0	0	0	0	0	0.33	0	0
32	0	0	0	0	0	0	0	0.33	0	0
33	0	0	0.33	0	0	0.33	0	0.33	0	0
35	0	0	0	0	0.33	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0.33	0
44	0	0	0.33	0	0.33	0	0.33	0	0	0
46	0	0	0	0	0	0	0.33	0	0	0.33
47	0	0	0.33	0	0	0	0	0	0	0.33

Fig. 3.13 Rank / Test Case matrix for all users

further, divide the matrix into blocks representing areas where the users cannot discriminate entirely between test cases. Fig. 3.14 shows the matrix rearranged into this block diagonal form.

By looking at this matrix (fig. 3.14), we can see clusters of non zero elements, which can be blocked, representing a class of test cases which fall into a band of the ranking scale used.

Such classes show the degree to which the users have agreed on the rankings assigned to the students. If the classes are very large, then the non zero elements are very widespread, showing little agreement between users. The smaller the blocks or classes, the better is the agreement between the users, and the smaller the range of ranks for each class. Fig. 3.15 shows the matrix of fig. 3.14 blocked, but without an algorithm it is difficult to find the best blocking i.e. the one which includes as many of the users rankings as possible, without making the classes too large. An algorithm would be able to set an agreement level, e.g. 80%, and block the matrix in the best way possible to reach this agreement level. Algorithms such as CONCOR [Arabie, Boorman & Levitt, 1978] carry out this form of blocking, but for illustrative purposes this exercise was carried out by hand. Without an algorithm, the best classes are identified by eye, and the agreement level calculated after.

From this matrix, we can see what level of agreement we have obtained, from studying the non-zero elements not included in a class. The percentage disagreement is calculated as below:

$$100 * \frac{\text{no. non-zero elements omitted}}{\text{total no. user rankings}}$$

When all the test matrices for the above example are taken into account, the agreement for this exercise is about 75%, which indicates a fairly good ability to discriminate on this factor at the 5-band level.

The overall result of this exercise is the same as that achieved from eliciting classes directly from the experts, i.e. a picture of the domain obtained from the breaking down of each discriminating factor into classes or bands. From the collation of results from the blocked

	7	44	30	24	36	35	26	20	45	6
1	1.00	0	0	0	0	0	0	0	0	0
8	0	0.33	0	0	0	0	0	0	0	0
9	0	0	0	0	0.33	0	0	0	0	0
10	0	0	0.33	0	0	0	0	0	0	0
11	0	0.33	0	0	0	0	0	0	0	0
18	0	0	0.33	0	0	0	0	0	0	0
19	0	0	0.33	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0.33	0
25	0	0	0	0.33	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0.33	0	0.33
27	0	0	0	0.33	0	0	0	0	0	0
28	0	0	0	0.33	0	0	0	0	0	0
29	0	0	0	0	0.33	0	0	0	0	0
31	0	0	0	0	0	0.33	0	0	0	0
32	0	0	0	0	0	0.33	0	0	0	0
33	0	0	0	0	0.33	0.33	0.33	0	0	0
35	0	0	0	0	0	0	0	0	0	0.33
37	0	0.33	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0.33	0	0	0.33
46	0	0	0	0	0	0	0	0	0.33	0
47	0	0	0	0	0	0	0.33	0	0.33	0

Fig. 3.14 Rearranged matrix

	7	44	30	24	36	35	26	20	45	6
1	1.00	0	0	0	0	0	0	0	0	0
8	0	0.33	0	0	0	0	0	0	0	0
9	0	0	0	0	0.33	0	0	0	0	0
10	0	0	0.33	0	0	0	0	0	0	0
11	0	0.33	0	0	0	0	0	0	0	0
18	0	0	0.33	0	0	0	0	0	0	0
19	0	0	0.33	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0.33	0
25	0	0	0	0.33	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0.33	0	0.33
27	0	0	0	0.33	0	0	0	0	0	0
28	0	0	0	0.33	0	0	0	0	0	0
29	0	0	0	0	0.33	0	0	0	0	0
31	0	0	0	0	0	0.33	0	0	0	0
32	0	0	0	0	0	0.33	0	0	0	0
33	0	0	0	0	0.33	0.33	0.33	0	0	0
35	0	0	0	0	0	0	0	0	0	0.33
37	0	0.33	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0.33	0	0	0.33
46	0	0	0	0	0	0	0	0	0.33	0
47	0	0	0	0	0	0	0.33	0	0.33	0

Fig. 3.15 Blocked matrix

matrices (as in Fig. 3.15), we can draw the picture as shown in fig. 3.16 of the factor 'ability' on which the users were ranking students.

An alternative method of representing the input data sets is by using a structure diagram, as in fig. 3.17. This diagram tells us that the examples or test cases are fully ordered with respect to this factor i.e. that there are six invariant classes for this discriminating factor, which can be discriminated by all users and put into order. In this example however, we have allowed a tolerance level, or error level of α (about 25%), indicating that α % of users may not discriminate cases into the same classes as the other (100 - α)%, i.e. there may be up to α % error.

In order to demonstrate the increase in error with the increase in the number of 'bands' or data points, the above set of rankings were divided into blocks ranging from 1 to 6. The table of fig. 3.18 shows the results of these blockings. In each case, the blocking was such that the minimum error level was achieved.

In this example it can be seen that the error level rises quickly from 7% to 20% with the increase from 2 to 3 blocks, then plateaus at 20%.

To conclude, the development of an expert system which is robust in terms of consistency and reliability in the hands of the users, depends largely on the robustness which can be attached to the input data of the system. As the reliability with which the users can be expected to select the correct input values decreases with an increase in the number of data points to choose from, then advantages can be gained from using limited data sets at the input stage.

3.24 Communicability

Communicability in this context is concerned with the ability of an expert to communicate and explain his knowledge to others, both expert and non-expert, in the same domain. The ease of this task will vary depending on the type of process and related data.

1	2 - 24	25 - 28	29 - 37	38 - 42	43 - 47

Fig. 3.16 Classes within the factor 'ability'

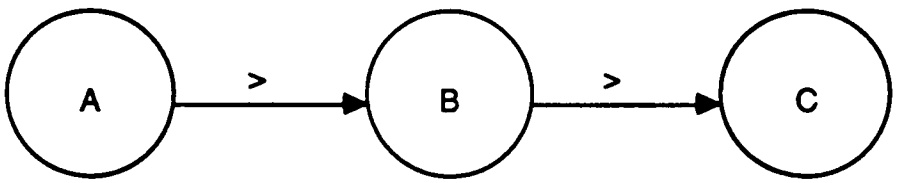


Fig. 3.17 Structure Diagram

No.of blocks	Error (%)
1	0
2	7
3	20
4	20
5	23

Fig. 3.18 Relationship of error level to number of blocks selected

A continuous process e.g. the speed of a car, poses difficulties due to the inability of humans to express verbally such continuous changes. Such data can only therefore be expressed by means of descriptive words and phrases from a limited data set. The speed of a car for example could be described by the terms : fast, slow, accelerating, decelerating, starting, stopping etc. all of which communicate the continuous action of the car.

As related to the user independence aspects of expert systems, communicability is also concerned with the relative discriminative abilities of the experts and users. In order for the expert to explain his reasoning, or the data he is using, to a less knowledgeable user, the terms used need to be at the level of discrimination of the user. The data set used therefore needs to be limited to the set of terms which are suitable for communication with the users of the system.

3.3 Use of the Data Flow Modelling Technique

In the previous sections data flow diagrams were used as the method of representing various tasks as solved by humans. This method of representation has certain advantages related mainly to the fact that it concentrates primarily on the data that flows through a system, but also suffers from inflexibility, due to the issue of control.

The traditional use of data flow diagrams is for modelling the data flows in an existing (manual) system (e.g. order processing, stock control) which is to be either partially or wholly automated. In such systems, the data flows correspond to "packets" of information such as physical orders, invoices etc., and the processes this data flows through correspond to the functions which are applied to and transform the data.

For this type of application the data naturally passes through several stages and transformations and therefore the control information which is inherent in the data flow technique suits such applications.

When modelling the way in which humans process information and solve problems however, this sequential treatment of the data is not always entirely suitable. At the highest level of determining where the explicitness boundary lies, the data flow method with its inherent flow of control is useful for several reasons.

i) The data flow method can be used to partition into different types or groups of processes e.g. pattern classification, processes using quantitative or qualitative data, motor control etc. The interfaces between different parts of the problem e.g. continuous and limited data and processes are therefore easily identified/defined as is the automation boundary in conventional data processing applications.

Fig. 3.19 illustrates the division of a problem into six main processes. Two of these lie outside the explicitness boundary, the first, process 1 being a pattern classification process and the second, process 6 being a pure calculation involving quantitative data as both input and output.

Process 3 is interesting from the point of view that it lies on the explicitness boundary. The process of arriving at a discount/loading figure for an assessment area may be carried out in one of two ways :

a) by calculation from discount/loading figures for individual factors (data from process 2),

b) by direct translation from the 'verbal' result for that assessment area (data from process 4).

The calculation part of the process therefore lies outside the explicitness boundary, whereas the 'translation' part lies inside. In either case, the discount/loading figure arrived at is directly comparable to a 'verbal' result, as one is always derivable from the other. The data output from process 3 can therefore be considered as qualitative in this context.

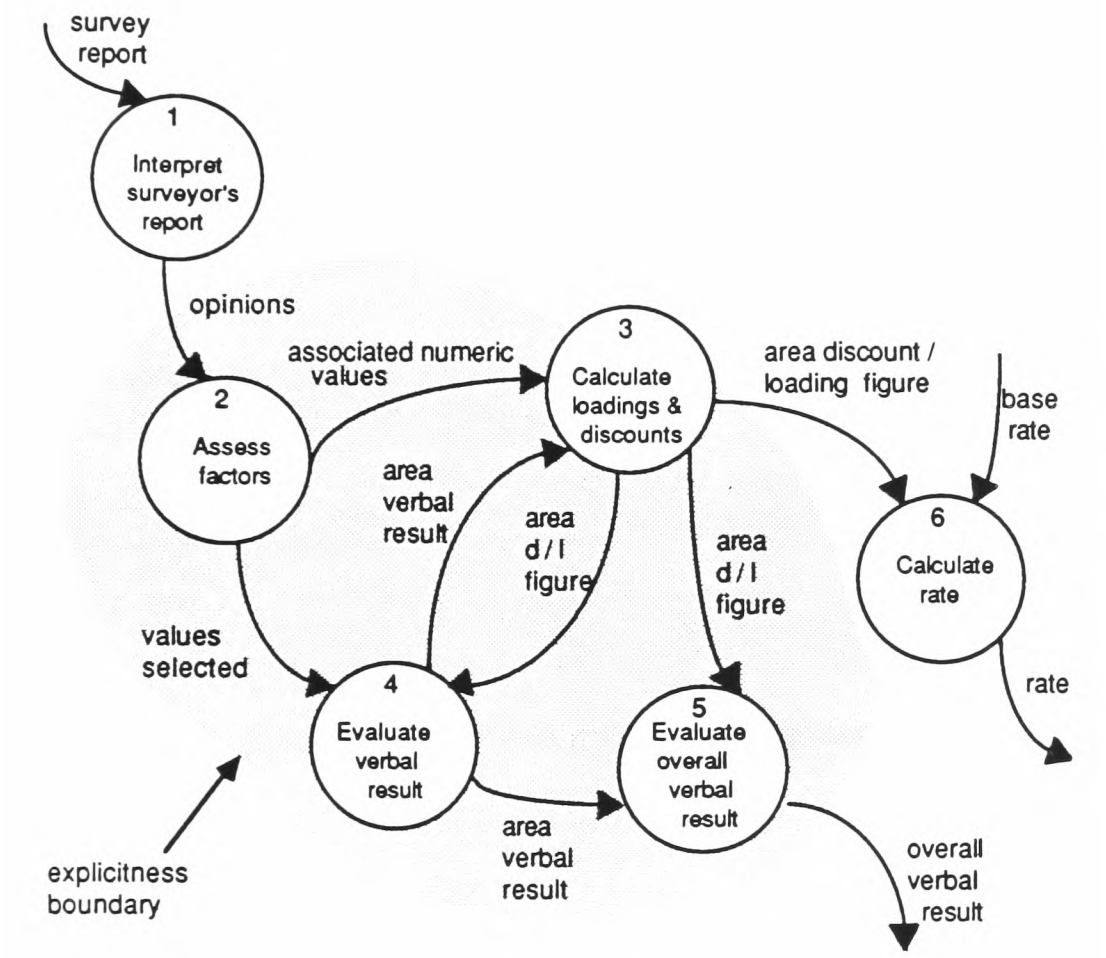


Fig. 3.19 Underwriting clothing trade fire risks - the explicitness boundary

The remaining three processes all involve the use of limited data sets and thus lie within the explicitness boundary.

ii) The inherent control feature of the method is good in that it shows the ordering of such processes within the human problem solving activity e.g. pattern classification before qualitative data processing.

In the same example (fig. 3.19), the data flow diagram clearly shows the order in which the processes can be carried out. Process 1 i.e. interpretation of the input data must always be carried out first and similarly process 6 which calculates the rate to charge must always be carried out last. Processes 3 and 4 however may be carried out in either order.

Figures 3.2 - 3.11 also illustrate the usefulness of data flow diagrams in determining where the explictness boundary should lie, but when we look more closely at these applications and begin to attempt to model the data and processes inside the boundary, we meet difficulties. Taking the mathematical application (fig. 3.11) as an example, we meet the problem of inflexibility due to the inability to show control information where desired, coupled with the necessity to show some control due to the method itself.

In fig. 3.11 control information has been included in two places, as shown by the heavy lines. According to data flow convention, this control information should not be included on the diagram, as no data are flowing from process 1 to process 2, and similarly from process 2 to process 3. A data flow should not merely consist of a prompt for a process to begin.

It is clear however that this particular flow of control is required at a more detailed level i.e. the data/information within the knowledge base needs to be treated in a certain way. We therefore need a mechanism by which this type of control can be represented either explicitly or implicitly at some stage of the modelling process, not provided by the data flow technique.

The problem we are faced with when using data flow diagrams as a method of representing this type of information can be summarised as follows :

'although data flow diagrams have inherent control information due to representing flow of data, they do not have the flexibility to show any other control information, data dependencies etc.'

When related to human data processing the relevance of this problem is that human processing is often not a set of sequentially ordered functions in which a piece of data goes through a series of transformations. On the contrary, as in the previous example of comparing two numbers (A<B ?), there are several possible processes/transformations which can occur, but these will not always occur, or occur in the same order, and also, will often use different states of the data, if not different data. We therefore need to be able to show this kind of control information and/or data dependencies.

Although the data flow method is suitable for defining the explicitness boundary, a method whereby the meaning of the data and its relationships to and dependencies on other pieces of data can be represented is required for modelling the processes inside this boundary. The processes being modelled begin to look more like a real time system, where a function is selected according to the present conditions i.e. the states of the data in the knowledge base. Such methods are more likely to be found from database design methodologies where the data are initially analysed independently of the functions.

3.4 Conclusions

In this chapter, the hypothesis that the an explicitness boundary should be defined as a 'limited data' boundary, has been investigated. This imposes a constraint on the system development process that all data both entering and leaving the explicit knowledge base of the system should be of the limited data form defined.

This constraint has the advantage of aiding the identification of the explicit processes which are required to be modelled for the knowledge base.

It has been argued in this chapter that the use of limited data sets to define the boundary is justified in terms of two main issues i.e.

i) Michie's 'human window'

ii) user independence.

i) The concept of the human window suggests that in order for a process to be explicit i.e. comprehensible to a human, it must be represented in such a way that the balance of memory usage and processing is at the level of human ability.

It is shown by means of examples that qualitative data naturally takes the form of limited data sets in decision-making i.e. decision-making involves the use of descriptive terms with semantic content. The decision as to whether such processes are within the human window is straightforward in this context i.e. a limited set of descriptive terms which are understood by the expert, communicable to others and used in the decision-making process, must lie within the human window for that task. Such processes therefore also lie within the explicitness boundary. It may be the case that some intermediate processing is required which transforms quantitative data into a qualitative form. The transformation may be made by the expert in which case it takes place outside the knowledge base (and system) or it may be able to be automated. In such cases the explicitness boundary is actually defined by the transformation of the data. The quantitative data cannot be said to exist within the human window in the sense that they are not usable for processing in that form i.e. the expert needs to first transform the data into values belonging to limited data sets which can then be said to satisfy the constraint of small memory and simple processing.

For some expert tasks the use of quantitative data for processing may be necessary e.g. for financial decision-making. Two views can be taken of such data :

a) the specific tasks which use quantitative data are not an important part of the expertise and therefore do not need to exist within the explicitness boundary. In such cases it does not matter whether the constraint of small memory and simple processing is satisfied.

b) These tasks are an important part of the expertise in the context of the system being developed. In such cases, the processes must be made explicit requiring that the data used by the processes must be analysed carefully in order that it can be designed to derive from a limited data set.

ii) The concept of user independence places a constraint on system development that potential or representative users of the system should be involved and be the target of the design. The use of limited data sets for user input is justified in this chapter in the following ways :

a) The user of the system may not have the same discriminatory powers as the expert thus indicating an increase in the 'grain-size' of the input data towards limited data sets.

b) Limited data sets imply descriptive terms for input thus guiding the user as to the correct term to select in the same way as a concise 'help' message.

c) It is shown by example that limited data sets naturally exist even if continuous rankings are used as a method of data collection from users. Additionally, as the number of values in the data set is increased (i.e. towards a continuum) so the degree of error increases thus indicating that robustness of the system is improved in terms of the input, by the use of limited data sets.

Due to the definition of the explicitness boundary being part of the **logical** design of the knowledge base, it is not envisaged that any constraint is placed on the type of system to which the method is applicable. The definition of the data entering the explicit part of the

system i.e. the knowledge base as limited data sets does not have to constrict the physical implementation to crisp reasoning.

The use of the data flow technique for modelling the expert decision-making process has both advantages and disadvantages as discussed in section 3.3. Advantages of the method include

* the ability to isolate different processes or groups of processes and thus partition the experts problem,

* the ease with which the explicitness boundary can be defined in the same way as the automation boundary for traditional systems design.

The main disadvantage of the use of data flow is concerned with the issue of control. Human data processing is not a sequentially ordered activity as implied by the data flow technique and thus control information needs to be modelled in addition to data flow. The data flow technique is therefore useful at the highest level of defining the explicitness boundary, but is probably not a viable technique for modelling the decision-making process in any detail.

CHAPTER 4 LOGICAL MODELLING WITHIN THE EXPLICITNESS BOUNDARY

4.1 Form of the Explicit Knowledge

In order to decide whether logical data structure modelling is a suitable method for formulating the expertise within the explicitness boundary, first the form of the explicit knowledge needs to be defined. As used for logical database design, the (logical) data model represents the intensional view of the database. Date [1981] defines the intension and extension of a relational database in terms of time dependence i.e. the intensional part of the database is the part of the database which is independent of time. As such, the intension defines the permanent part of a relation, and also defines all permissible relations. The extension is equivalent however, to a 'view' of the database and is thus made up of the set of tuples of that relation at a given instant. As tuples may be created, amended or destroyed, the extension of the relation will vary with time.

In the same way as a set of relations defines the intension of a relational database, an ER model also defines the intension of a database, relational or otherwise. The entity and relationship types represent the static parts of the system which are least likely to change with time, whereas the individual occurrences of these types represent time dependent applications or instances.

Mealy [1967] has also applied the intensional/extensional distinction to databases and the symbols within it. Files of data which describe the 'real world' represent extensions, whereas the constraints in the data dictionary are meaning postulates which define the intension. The idea of the intension of the database is thus linked with the idea of the meaning of the data.

Quine [1961] defines the concepts of intension and extension in terms of the theory of meaning and reference (naming). As an example, the two extensions '9' and 'number of planets' name the same abstract entity but are not alike in meaning. The abstract entity which can be described as the intension is the number nine of which there are an infinite number of

extensions. For the above example, observation was required in order to determine the equality between the two terms, that equality could not be determined from the meanings of the two terms. Thus, the class of all entities of which a general term is true is called the extension of that term. Those statements for which observation is required for verification are known as synthetic statements as opposed to analytic statements which are true by definition.

In terms of knowledge as opposed to data, the notion of intensional or extensional knowledge is linked with the form of memory. Tulving [1972] classified memory into two categories : semantic (after Quillian) and episodic. Episodic memory is concerned with the occurrence of temporal episodes and events, and the temporal-spatial relations between them. Each event in episodic memory is describable in terms of both the latter relations i.e. autobiographical reference to other events, and perceptible attributes.

Conversely, semantic memory consists of organised knowledge about words and other verbal symbols in terms of their meaning, referents and relations. Semantic memory also consists of rules or general principles for manipulating these symbols, concepts or relations.

Examples of episodic memory may be :

"I remember that I have a dental appointment at 9:45 a.m. tomorrow morning", or

"John is an A-level candidate".

Semantic memory examples are more general, not relating to specific events or occurrences, e.g. :

"I remember that the chemical formula for water is H_2O", or

"A-level candidates may pass or fail".

Such examples show that semantic memory could be considered as 'knowledge' rather than 'remembering', although the existence of episodes which enter information into semantic

memory are obviously necessary, and can change the content of semantic memory e.g. learning chemical formulas.

In the same way therefore, as the intension of a database defines the general principles of the database i.e. the types of data and constraints on that data, so the semantic memory defines the general principles which constitute knowledge.

The task of knowledge elicitation therefore becomes one of eliciting the analytic statements of semantic memory i.e. the intensional knowledge of the expert. Like the data in a database, this intensional knowledge can be described as the static part of the knowledge base i.e. it is (to a certain degree) independent of time. The problems which are solved using the intensional knowledge will vary however, and will be dependent upon time and are thus the extensions of the knowledge. In terms of an expert system therefore, it is the instantiation of data items at run time which constitutes the extension of the knowledge base. The intension is defined by the knowledge base itself.

In terms of databases, it has been stated previously that the data model which results from logical data structure modelling, defines the intension of the database. In terms of knowledge bases, it seems that the knowledge base is the intension. A data model would thus only define part of the intensional knowledge of the expert, possibly an abstraction or meta-intension of this knowledge. The 'rules' which make up the remaining part of the knowledge base, i.e. the reasoning associated with the expert task, would then be the detailed intensional knowledge of the expert. Thus the intension/extension distinction as applied to databases is not as straightforward when applied to knowledge bases.

In summary the form of the explicit knowledge to be represented in the logical data structure modelling phase of knowledge elicitation is the intensional knowledge of the expert. The following sections consider two aspects of this intensional knowledge :

i) the intensional view of the data associated with the expert task,

ii) the activation of processes or functions within the expert task, as controlled by the values of specific data items defined in i) as status attributes.

4.2 Entity-Relationship Modelling of Explicit Knowledge

The Entity-Relationship (ER) modelling process can be split into three phases as follows :

- entity-relationship analysis,
- attribute analysis,
- instantiation of static knowledge.

The first two of the above phases are concerned with the construction of the data model to represent the intensional view of the domain knowledge. In particular it is the 'data' or concepts associated with the expert task which will be represented by the resulting model, as opposed to the functional knowledge of the expert. A certain amount of knowledge as opposed to just data could however be said to exist within the model e.g. relationships between concepts or objects, as will be seen in the examples later in this chapter.

The third phase is concerned with more detailed intensional knowledge, which can be considered as static i.e. the same for all problems. Data and relationships which will vary depending on the specific problem being solved are conversely, dynamic. Although such knowledge can be shown on the ER model, dynamic data and relationships cannot be instantiated before consultation with the system.

4.21 Entity-Relationship Analysis

Firstly, the entity types must be defined - i.e. the factors or concepts within the expert domain, which can also be considered as the data relevant to the expert task. The term 'data' should not be confused with the lowest level data items or instances which may occur in specific tasks, but should strictly be interpreted as the types of data i.e. the intensions that may exist within the task domain.

Relationships between entity types must then be defined and named. These show logical groupings of factors, and thus portray to a certain degree the way in which the expert task is carried out using these factors, although not from a procedural angle.

Through defining the entity and relationship types in this way, an entity-relationship (ER) model is built up, which represents the logical structure of the intensional, static knowledge of the expert. Any entity types which are unrelated to others are probably not relevant to the problem solving task, and any one:one relationships should be examined to see whether one entity type is the attribute of another, as opposed to being an independent factor.

The resulting ER model is a 'global' view of the intensional knowledge of the expert, and is not specific to particular instances of the problem. It represents part of the static knowledge of the knowledge base, which is least likely to change.

In expert systems, maybe more so than conventional data processing systems, a distinction can be made between statically and dynamically instantiated knowledge in the knowledge base as mentioned previously. It may therefore be useful to show on the ER model those relationships which only occur dynamically i.e. there may not always be a relationship between two entity types and if there is the connection may only be made at consultation time.

Using this technique as opposed to say, semantic networks at this stage eliminates the necessity to explicitly include such dynamic relationships. At implementation time, using for example frames which are in effect a form of semantic network, such relationships can be easily instantiated by the addition of the necessary slots within specific frames or instances of frames. As is shown in the example of section 4.31 however, it may sometimes be advantageous to include these dynamic relationships on the ER model i.e. where they form an important part of the expertise.

4.22 Attribute Analysis

For each entity type identified, the attributes must be defined e.g. an entity type 'person' may have the (relevant) attributes 'age', 'sex', 'marital status' etc. Included in the attributes for some entities, will be a status attribute. These attributes may not be discovered however, until further analysis has been carried out into the functions related to the expert task.

Status attributes describe the state of the knowledge base and are thus, in a sense, partly concerned with the control knowledge associated with the expertise. An attribute (of an entity) which becomes instantiated during the system consultation and whose value is important to the expert reasoning process, can be called a status attribute. Events trigger procedures which change the values of status attributes and thus change the state of the knowledge base thus enabling further processes to take place.

4.23 Instantiation of 'Static' Knowledge

Parts of the ER model which are static, i.e. will be the same for all instances of the expert task, can be instantiated at this stage of analysis. This applies to both entity and relationship types, as follows :

- entity types may have a limited set of occurrences; their attributes may have a limited set of values.

- relationship types - an occurrence of one entity type may always be associated with the same entities of another type, via the same relationship.

- in addition, certain 'rules' or constraints may be obvious both from the ER model e.g. the status attributes which need to become instantiated, and from the problem description e.g. calculations required.

4.3 Examples

4.31 Fire Risk Assessment

The Fire Risk Assessment system developed as part of the ARIES project at Logica [Logica, 1986, Butler, 1987] is an expert system to aid branch office underwriters in the task of rating fire risks within the clothing trade. The domain is split into six assessment areas, each of which is assessed for risk according to a number of factors. The underwriter answers a series of questions concerning these factors, and is presented with a set of values for each factor from which to select an answer. These answers i.e. the values selected by the underwriter are combined to provide two results for each assessment area - a verbal result and a numeric result. These are then combined in order to produce both a verbal recommendation e.g. accept, reject, and a numeric rate to charge for the risk.

4.31.1 Entity-Relationship Analysis

The entity types within the narrow domain of fire risk selected can be identified as follows

- rate assessment
- assessment area
- factor
- discrete value

Although not generally the case in entity modelling, these four entity types have a hierarchical structure as shown on the ER model in fig. 4.1. The 'rate assessment' entity type is a dynamic entity type i.e. an occurrence is only created when a rate is to be calculated for a particular risk. The relationship 'calc-from' is therefore a dynamic relationship which is shown on the ER model by a heavy line. The symbol $\forall$ is used to indicate that the assessment is made from all areas.

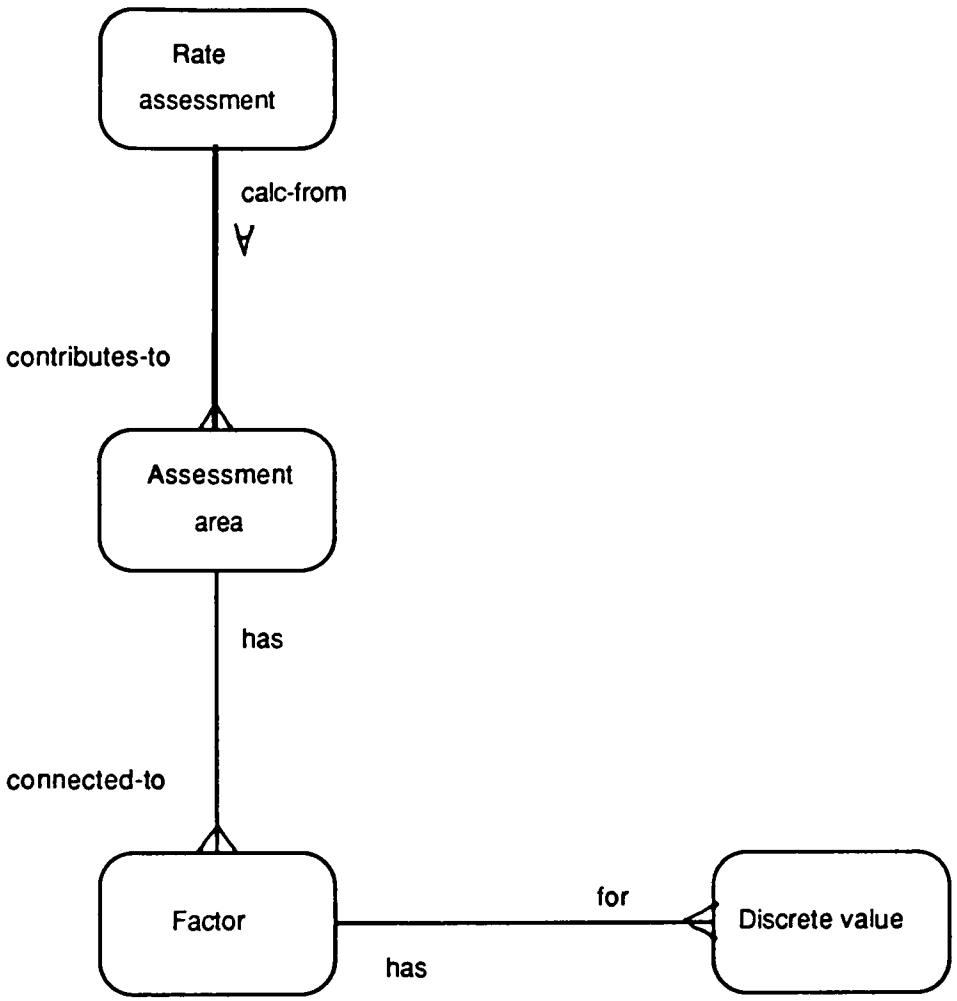


Fig. 4.1 ER Model for Fire Risk Assessment

4.31.2 Attribute Analysis

Rate assessment : rate; rate-status; numeric result.

Assessment area : name; area-status; numeric-result; factors.

'Rate-status' and 'area-status' are status attributes which indicate both the verbal result taken by the two entities and the 'status' of the entities. The values taken by these attributes are considered to be 'states' which the entities can be in. The use of these status attributes is made clearer in 4.4 where entity life cycles are discussed. 'Factors' indicates the relevant factors for the assessment area.

Factor : name; description; factor-status; values.

As above, 'factor-status' is a status attribute which will indicate which of the discrete values the factor has been instantiated to, if changed from the default value. The 'values' attribute is designed to hold a list of the relevant discrete values from which 'factor-status' can take its value.

Discrete value : name; description; numeric-value.

4.31.3 Instantiation of Static Knowledge

Entity types and occurrences

Entity Type	**Occurrence**
Assessment area	Construction
	Trade Processes
	Management
	etc.
Factor	standard of building construction
	number of storeys

	number of employees
	number of tenants
	electricals
	smoking
	etc.
Discrete value	essentially grade I
	essentially grade II
	< 3
	4-12
	above average
	average
	etc.

Attributes and values

Entity Type	**Attribute**	**Value**
Rate assessment	rate	numeric
	rate-status	v∈V/not known
	numeric-result	numeric
Assessment area	name	alpha
	area-status	v ∈V/not known
	numeric-result	numeric
	factors	list
Factor	name	alpha
	description	alpha (text)
	factor-status	d∈Dx/default
		(also ∈ Dx)
	values	list (Dx)

Discrete value	name	alpha
	description	alpha
	numeric-value	numeric

V above is the 'domain' of values which can be taken by the attributes 'rate-status' and 'area-status', and has the following values :

Discount, Normal rate, Load, Consider Reject, Reject

Dx is the 'domain' of discrete values i.e. the limited data set which corresponds to each factor. These would be specified in e.g. a frame system in the 'values' slot as a list of the discrete value instances which are applicable to that factor. An example domain Dx is shown below for the factor 'collection of trade waste'.

At least daily, Less than daily, Not known

Relationships

Rate assessment	*calc-from all (∀)*	assessment areas.
Assessment area	*has one or many*	factor(s).

Each assessment area has its own limited set of factors applicable only to that module, for example :

Construction : standard of building construction, no. of storeys, incombustibility of external walls, standard of roof, combustible floors, ...

Trade Processes : no. of employees, no. of tenants, no. of trade process floors, quality of hazardous appliances/electricals, type of cloth, ...

In, for example, a frame system this would be represented as a list of instances of factors in a 'factors' slot in the 'assessment area' frame.

Factor	*has one or many*	discrete value(s).

As described above, each factor has its own domain of values Dx represented e.g. as a list in a frame system, in the 'values' slot.

4.31.4 Extending the Entity-Relationship Model

One of the difficulties with using ER models to represent the above problem is concerned with the distinction between instantiation of entity types and relationships during knowledge elicitation (static knowledge) and that during system consultation (dynamic knowledge). This constraint can be included in the ER model by the use of heavy lines to indicate dynamic relationship types. An additional difficulty which occurs in this problem is posed by the constraint that certain dynamic instantiations can only take their value from a limited set of entities taking part in the relationship.

In the case of fire risk, one assessment area is only connected with a specific, known set of factors and similarly one factor is only associated with a specific set of discrete values. This in itself is not a problem, as the physical implementation can handle such relationships e.g. in a frame system the 'list' data structure can be used as a slot value in the 'assessment area' frame to represent the relevant 'factor' entities/instances for that area.

This form of relationship could be shown on the ER model as in fig. 4.2

i.e.

one	assessment area	*has one*	factor set,
one	factor set	*connected-to one*	assessment-area,
one	factor set	*consists of one or many*	factors,
one or many	factors	*belong to one*	factor set.

Clearly however, the 1:1 relationship between 'assessment area' and 'factor set' is redundant and could be omitted.

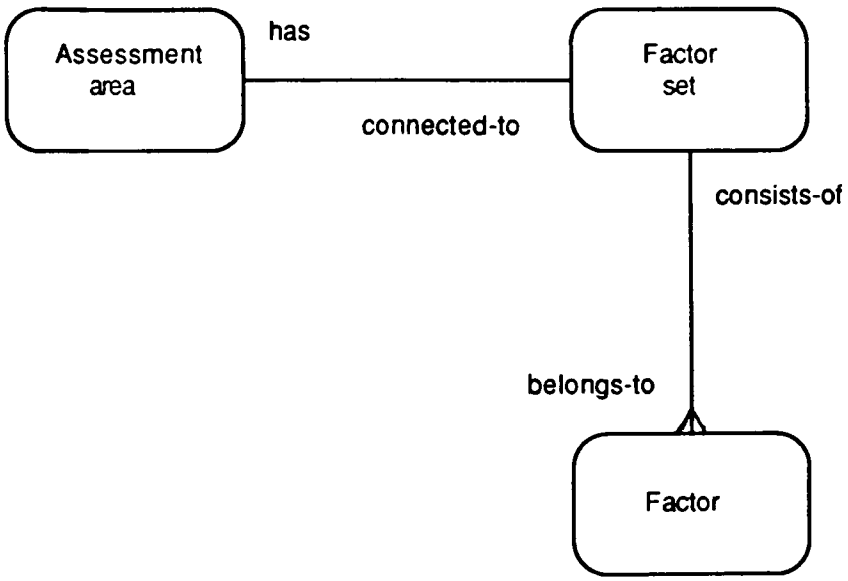


Fig. 4.2 Constraint of 'factor set' per assessment area

This problem with entity models becomes important however when there is a need to indicate constraints of this type which are instantiated **dynamically** i.e. at consultation time rather than statically as above. In the original ER model for fire risk, the relationship between factor and discrete value (fig. 4.3) is over-simplified and does not represent the dynamic instantiation of the knowledge base.

As for the 'assessment area - factor' relationship, each factor has a distinct set of discrete values, which could be represented by the ER model of fig. 4.4.

The knowledge represented by this model is static knowledge instantiated at knowledge acquisition time. In this case however at consultation time a factor becomes dynamically associated with only one discrete value from the discrete value set it is connected with. One method of representing this constraint may be to introduce an extra relationship which links 'factor' and 'discrete value' as in fig. 4.5.

The extra relationship is intended to denote the constraint that the discrete value which becomes dynamically associated with the factor can only be from the discrete value set which is also associated with that factor. The use of binary relationships as above does not however allow this constraint to be modelled explicitly, as the three relationships are independent of each other.

A different method of representing the constraint may be to use the concept of ternary relationships [Teorey, Yang & Fry, 1986] as in fig. 4.6, where three entity types are involved in one relationship thus overcoming the independence problem above.

This relationship explicitly states that for a given pair of occurrences of factor and discrete value there can exist only one discrete value set occurrence, thus the discrete value must belong to that set. Similarly for a given pair of occurrences of discrete value set and discrete value, there can exist only one occurrence of factor. For a given pair of occurrences of factor and discrete value set however, there can exist many discrete values.

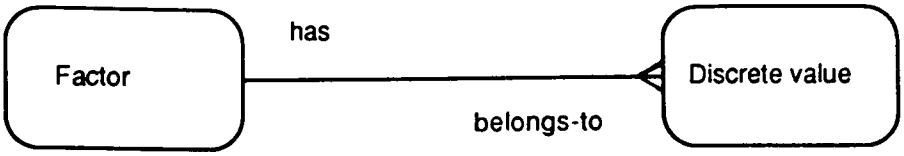


Fig. 4.3 Factor-discrete-value relationship of original ER Model

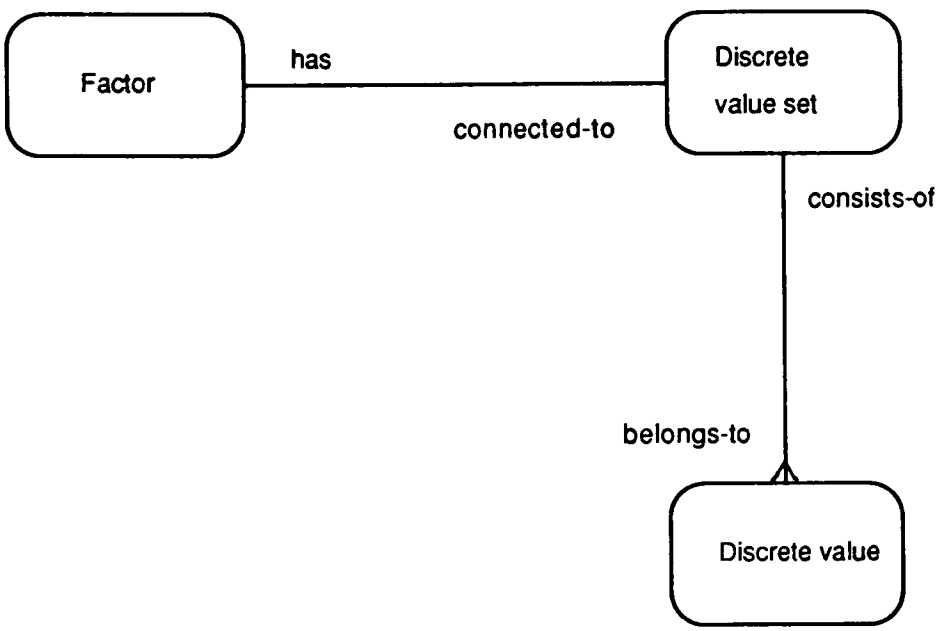


Fig. 4.4 Introduction of a 'discrete value set' entity type.

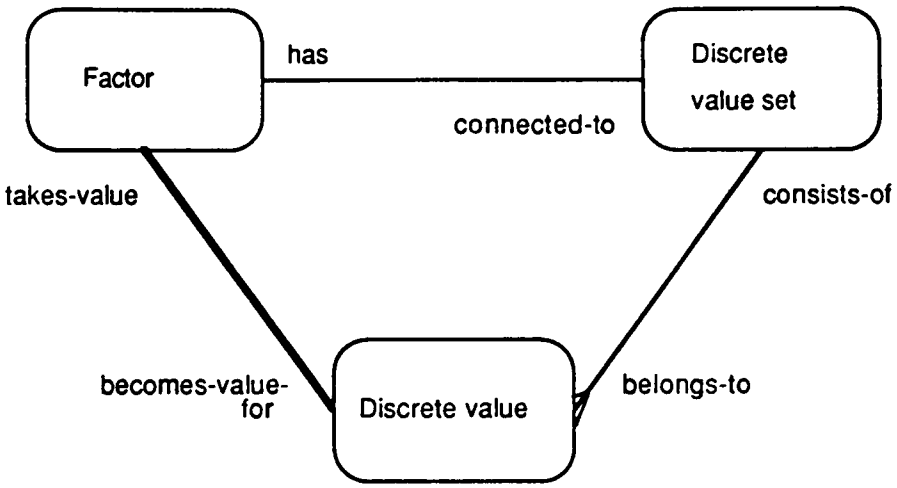


Fig. 4.5 Introduction of a third binary relationship

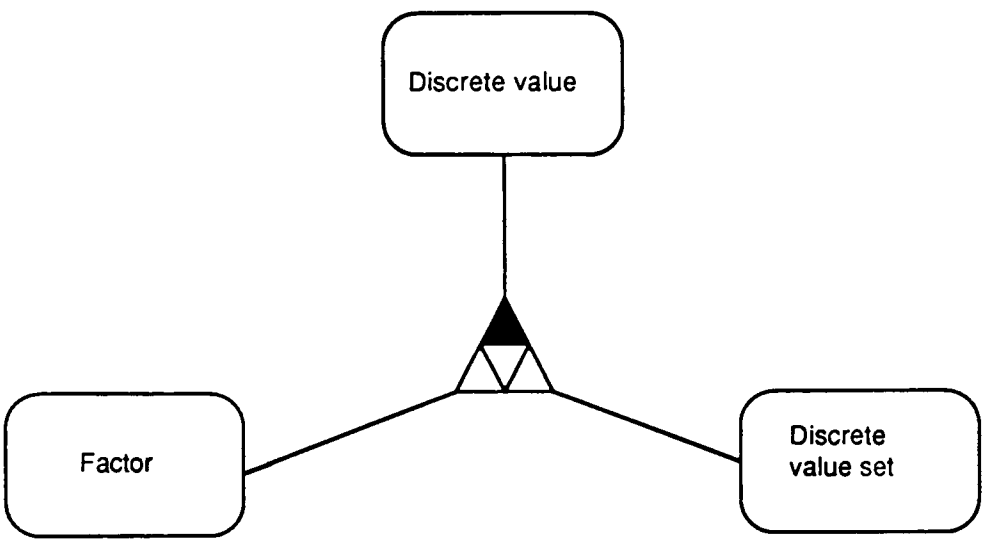


Fig. 4.6 Use of ternary relationships

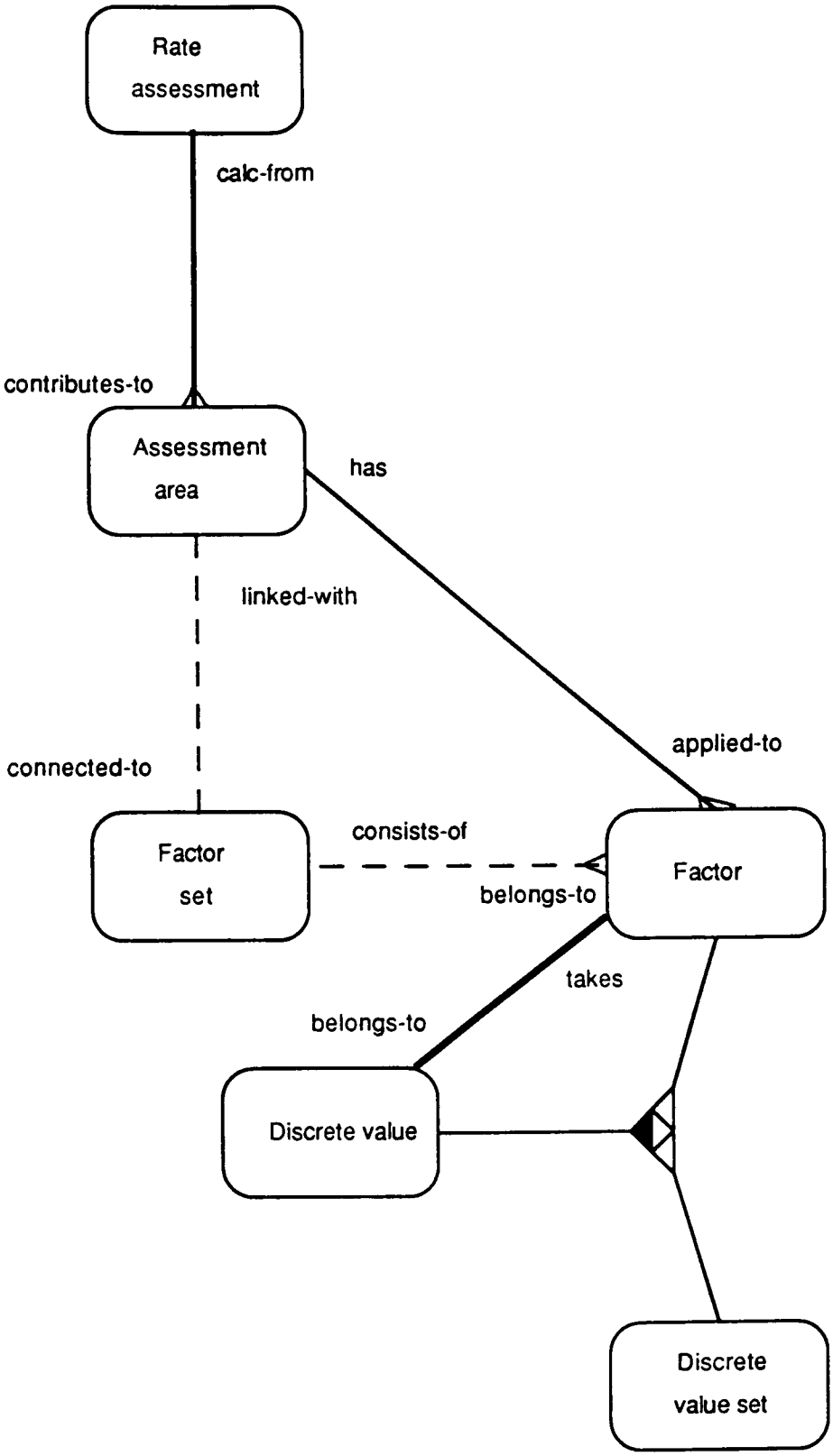


Fig. 4.7 Revised ER Model for Fire Risk

The use of the above ternary relationship results, however, in the loss of representation of the dynamic relationship between factor and discrete value. This could be overcome by the inclusion of the latter binary relationship in addition to the ternary relationship. The ER model for fire risk can now be redrawn (fig. 4.7) to include the above constraints. The dotted lines represent the optional entity type factor set and its associated relationships, and the heavy lines the dynamic relationships.

4.32 Plant Diagnosis

The following example is taken from the ESP Frame Engine User Guide [Expert Systems International, 1985], but is extended to incorporate more entity and relationship types in order to demonstrate more fully the use of the ER modelling technique.

The purpose of the system is to diagnose the diseased state of a given plant by looking at the symptoms or diagnostic signs present on the plant. Diagnostic signs e.g. rust spots, weak stem, etc., have either a positive (add 1 to disease score) or null effect on the 'disease state' of the plant. The final disease score attributed to the plant indicates the diseased state of the plant i.e. the degree to which the plant is suffering from the diseases found to be present.

4.32.1 Entity-Relationship Analysis

A plant may suffer from one or many diseases and conversely a disease may be attributed to one or more plants i.e. a many:many relationship exists between plant and disease. The intermediate entity type disease-state eliminates the many:many relationship and represents pairs of occurrences of plant and disease.

For every disease plant pair therefore, there may exist a disease-state indicating the degree to which plant x suffers from disease y. The disease-state entity will not, however, exist until run-time of the system i.e. it will be dynamically created and thus can be called a dynamic entity type. The relationships in which the entity type participates will also therefore be dynamic in nature i.e. they cannot be instantiated during knowledge elicitation. The intension

shown by the ER model represents a set of extensions such as that in fig. 4.8 where the heavy line is used to show dynamic entity and relationship types.

A plant possesses many parts, a part may show one or many diagnostic-signs i.e. disease symptoms. A disease may be indicated by many such symptoms and similarly a diagnostic sign may be a symptom for many diseases. The resulting many:many relationship is eliminated by the introduction of the entity type disease-sign-effect which represents the contribution (positive or null) which a specific diagnostic sign makes to the strength of a specific disease. The intension shown by the ER model represents the types of extension a shown in fig. 4.9.

The disease-state entity also takes part in a one:many relationship with the disease-sign-effect entity, as the disease-state score is equivalent to the sum of all relevant disease-sign-effect entities. As an example, for the disease-plant pair d1-p1, the disease-state score would equal the sum of all disease-sign-effect instances which are relevant to disease d1 and for which the relevant diagnostic signs have been found to be present. This relationship is a dynamic relationship due to the latter condition i.e. the disease-state score is dependent upon the diagnostic signs found to be present, whereas the former set of relationships constitutes static knowledge. It is known previous to consultation i.e. at knowledge acquisition time that certain diagnostic-signs have either a positive or null effect on certain diseases and therefore the disease-sign-effect entities can be instantiated. In fig. 4.10 which shows the extensions represented by the ER model, the heavy line indicates the dynamic relationship between disease-state and disease-sign-effect.

The disease-state of the plant is thus represented by the score gained for each disease found to be present.

The ER model of fig. 4.11 represents all the above concepts and relationships.

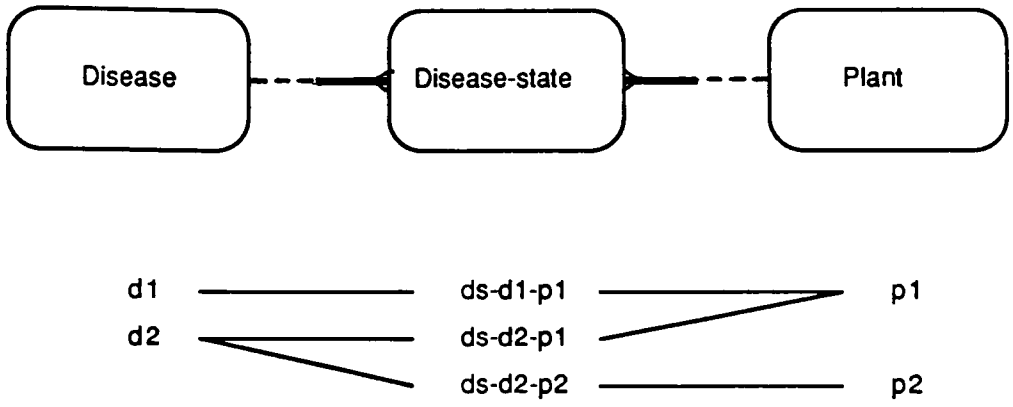


Fig. 4.8 Disease-state entity type and relationships

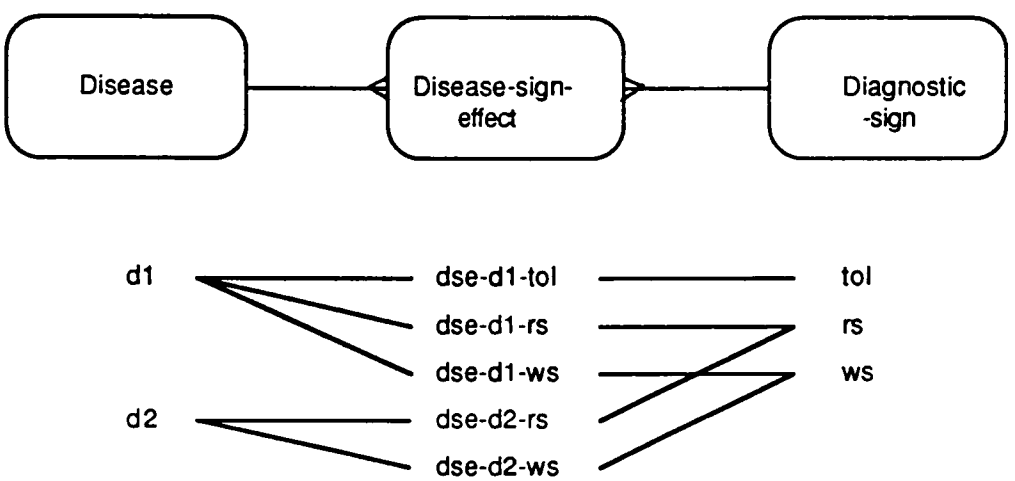
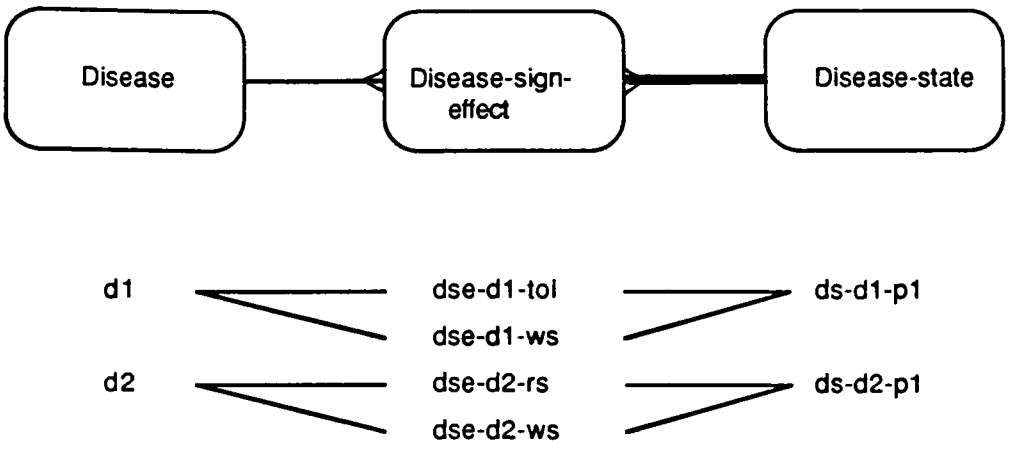


Fig. 4.9 Disease-sign-effect entity type and relationships (1)



Fig, 4.10 Disease-sign-effect entity type and relationships (2)

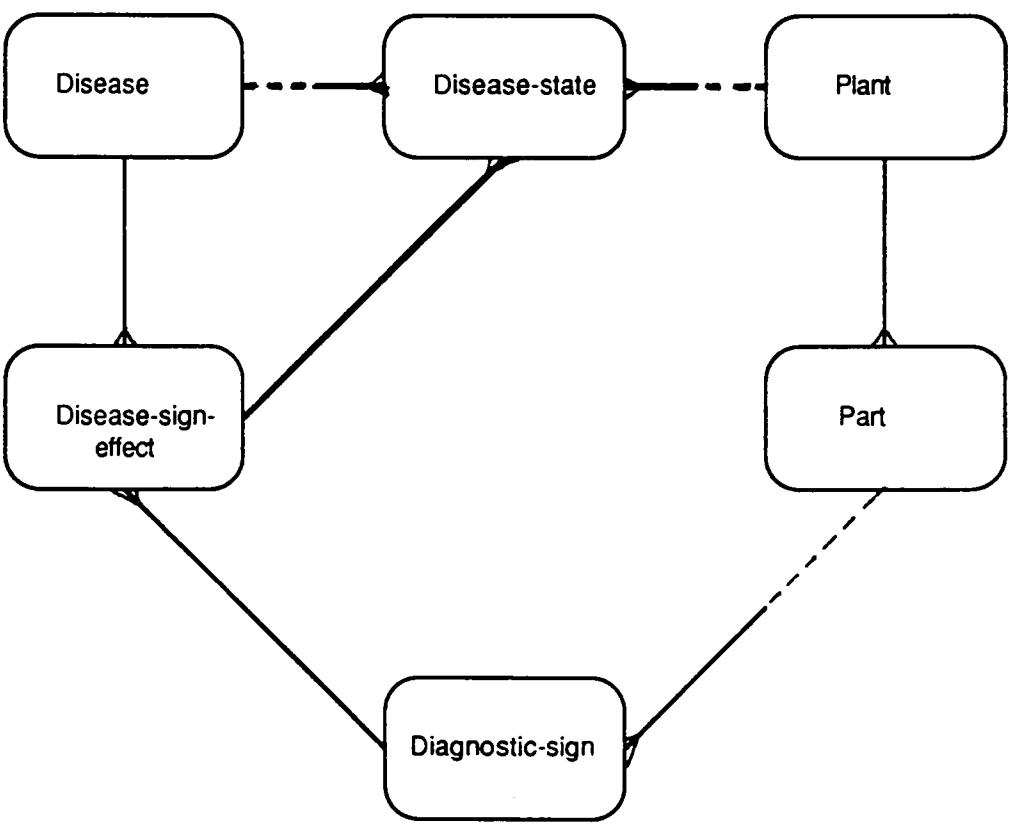


Fig. 4.11 ER Model for plant diagnosis

4.32.2 Attribute Analysis

The following list of entity types and their associated attributes illustrate an example of the attributes likely to exist for each entity type.

Plant : type;

Part : name;

Disease : name, description, treatment;

Diagnostic-sign : name, description, status;

Disease-state : score, status;

Disease-sign-effect : effect;

Status attributes in this example act in the same way as e.g. 'if-needed' demons in a frame system i.e. if the status of diagnostic-sign for plant x is 'not-known' then the relevant rules need to be fired in order to change the status to either of the other states i.e. 'present' or 'not present'. They are thus concerned with the control aspects of processing.

4.32.3 Instantiation of 'Static' Knowledge

Entities

For all of the static entity types in the ER model, a number of occurrences can be pre-defined e.g. for the disease entity type, the entities d1, d2 may exist. For some of the entity types, e.g. disease-sign-effect, the occurrences will be more complicated eg dse-d1-tol i.e. the effect which diagnostic sign tiers-of-leaves (tol) has on the disease d1.

Relationships

Static relationships which do not change at run-time can be specified at this stage e.g. parts of plants, diagnostic signs shown by parts, and disease-sign-effects indicating diseases due to the presence of diagnostic-signs. Such relationships are constraints on the knowledge base i.e. for existence, updating etc.

Additional constraints which are part of the static knowledge base are formed by 'rules' such as 'the disease-state of the plant is known when all (relevant) diagnostic-signs known to be present are evaluated', and 'the disease-state score for each disease is equivalent to the sum of the disease-sign-effect instances relevant to that disease'.

4.4 Control of Activation of Processes

Having defined the 'data' of the expert knowledge, it is necessary then to look at the knowledge which controls the use of that data. The degree of such knowledge involved in a task will vary from being a minor part dealing mainly with control issues, to a large part i.e. the procedural knowledge of the task. In either of these cases, the knowledge base, and specifically the data associated with the knowledge base will change state throughout the consultation. This section is concerned both with the analysis of the procedural or control knowledge, and modelling of the knowledge base states and events which cause a change in status, as the two are not entirely independent of one another.

As in real-time systems, where certain procedures are triggered by states of the data in the database, an important aspect of procedural knowledge is the control of activation of processes. The methods of modelling this control, as used in structured analysis i.e. data flow and data structure methodologies will be looked at to see if they can be applied to the modelling of the processes within the explicitness boundary.

4.41 Data Flow Approach

The pitfall of the data flow method is its inability to model the control aspects of processing. A certain degree of control is inherent in the method, as the processes are ordered in terms of the data which flows into and out of them. The data flow method does not however allow any additional control information to be modelled. Thus, control of activation of processes is governed only by the data arriving at a process, as typified by a data processing batch

system. Data which is intended purely to prompt a process is not considered as a legitimate data flow [de Marco, 1978].

The problem caused by this constraint is that some other means of modelling the activation of processes in relation to each other as well as the data must be found, with the risk of contradiction and resulting inconsistencies. It may not be the case that a process is always activated by the arrival of data resulting from another process - it may be activated by specific values or states of that data.

Fig. 4.12 shows a simple data flow diagram in which process 2 is preceded by process 1, process 3 by process 2 etc. It may not always be the case however, that process 3 is carried out after process 2 i.e. there may be other conditions which trigger process 3.

Buffers are used in data flow diagrams to indicate a store of data. This has an added effect of decoupling the processes which either send data to or receive data from the buffer, with the result that the control of activation of a process is not strictly dependent on the receipt of that data. In the above example process 4 is activated on the termination of process 3 and receipt of the associated data; process 6 may be activated at any time as the data resulting from process 5 is sent to a buffer. Process 6 does still however rely on using this data and can only process it on termination of process 5.

Strictly speaking therefore, representation of the control of process activation other than by receipt of data is explicitly prohibited by use of the data flow approach.

4.42 Data Structure Approach

The data structuring methodologies, such as ER modelling keep the issue of control independent of the data. The status of the data is related to the processes in the system, but the activation scheme is flexible. The states of the data are linked with events but the sequence of these events i.e. when they are activated, is independent. An event will trigger one or more processes which will change the status of the data in the database. This approach considers the status of a data item as an attribute of the associated entity, which

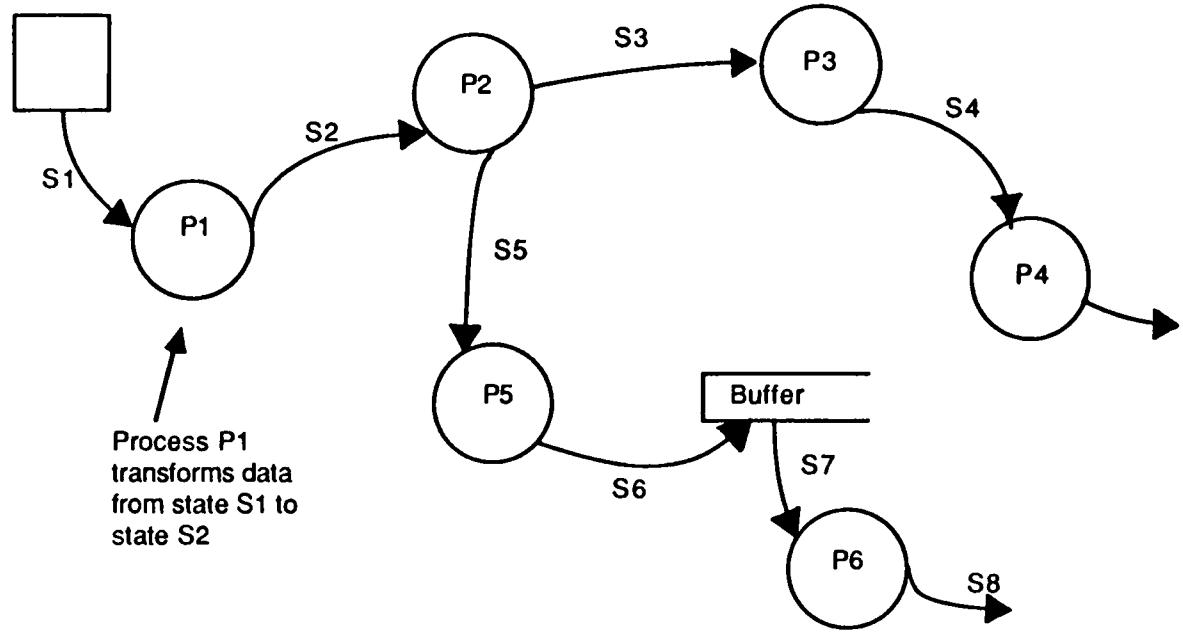


Fig. 4.12 Inherent control in the data flow method

will have a life cycle within the system. An entity type may have more than one of these status attributes; each entity life cycle diagram however only represents the changes in state of one entity (type) due to events which change the value of one status attribute.

Thus control is not modelled explicitly, rather the states in which the data can be, and the associated events which will cause a change in the status of the data and hence a change in the database.

As an example, consider the attribute 'marital status' of an entity type person. The entity life cycle in fig. 4.13 [from Rock-Evans, 1981], shows the states this entity type can be in and the associated events which cause the changes to occur.

Although the term entity life cycle is used, the initial state of the entity is not necessarily the same as the final state. In the above cycle, any of the states could be the initial or final state for a given entity occurrence.

Comparing entity life cycles with data flow diagrams, the former could be considered as the inverse of the latter. A data flow diagram represents the transformations of a 'packet' of data by a series of processes, thus between each process the data can be considered to be in different 'states'. If the data flow diagram of fig. 4.12 is redrawn as an entity life cycle (fig. 4.14), the process bubbles are actually represented by events which trigger one or more processes which cause a change in the status of an entity. The data flows thus become represented by the entity states themselves.

The entity life cycle can then be considered as equivalent to a data flow diagram where all processes are decoupled by use of a data store e.g. fig 4.15. The data stores represent the different states of the data. The entity life cycle does not however restrict the order of process activation as does the data flow diagram, other than by status.

In order for a process to be triggered (by an event), the data must be in the correct state i.e. state ES1 and event E1 will cause a change to state ES2. The processes are thus in a sense independent 'demons' which are activated by certain values in the data/knowledge base, as

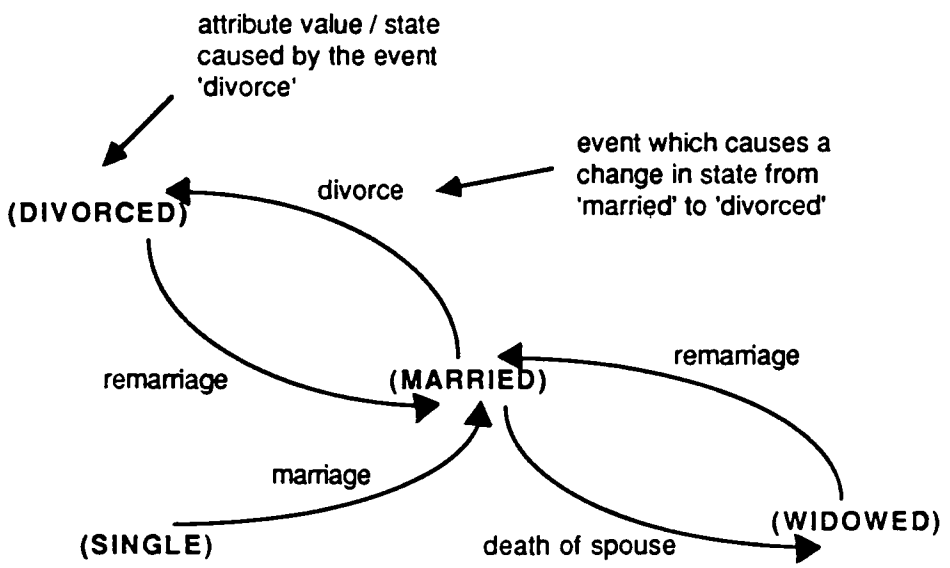


Fig. 413 Entity life cycle for 'marital status (after Rock-Evans).

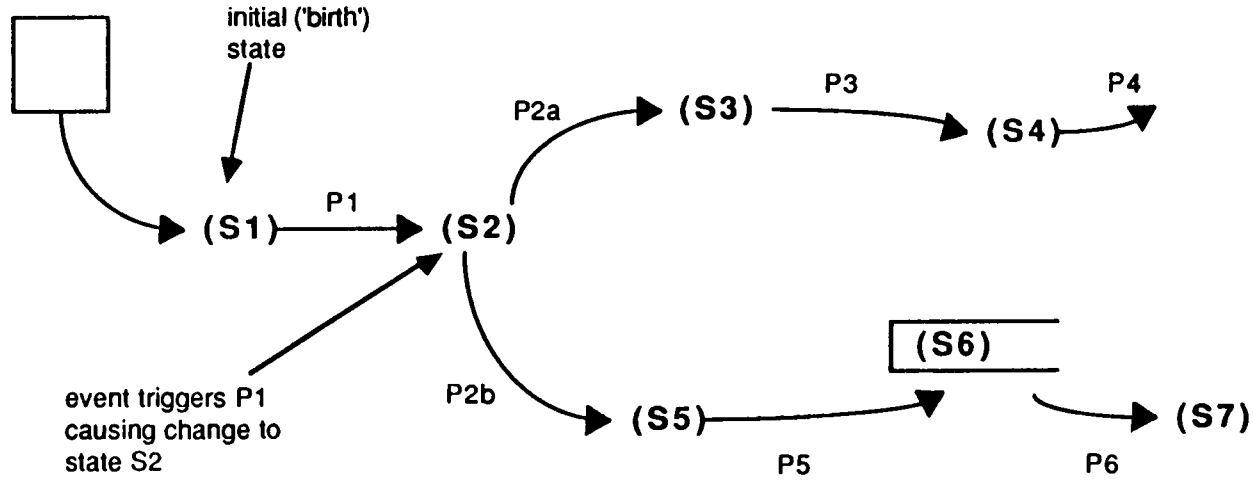


Fig. 4.14 Entity life cycle diagram for the DFD of fig. 4.12

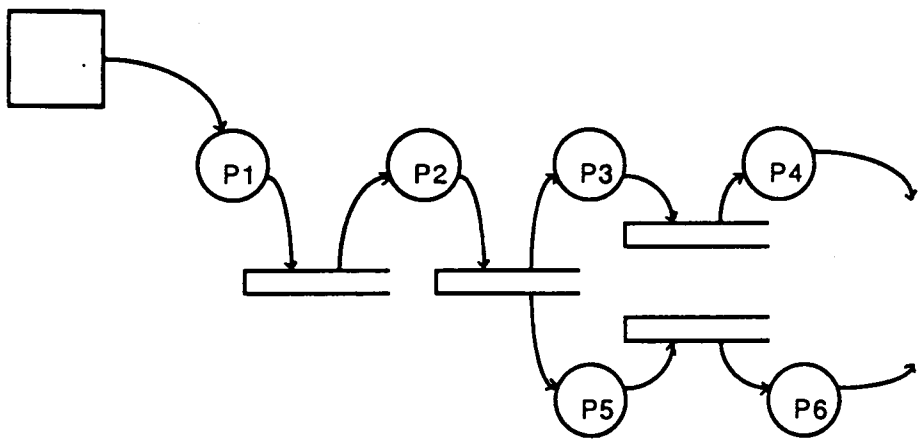


Fig. 4.15 Decoupling the Process

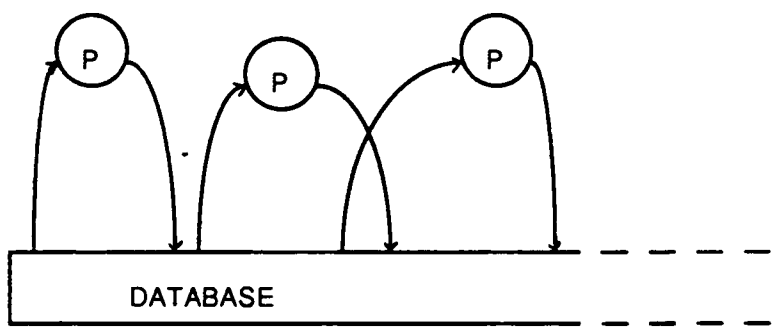


Fig. 4.16 A further stage of decoupling

represented by fig. 4.16. As long as the data is in the correct state for a process to fire, it can do so, and can also access other data in the database.

4.43 Within the Explicitness Boundary

The deficiency of data flow diagrams in modelling human processes is linked with their inability to model control of process activation, as discussed in 4.41. As we saw in figure 3.11, humans do not act like batch processing systems for data. The data flow method does not allow for links between processes which are purely representing control as in this example. The processes shown in this figure may however be regarded as in figure 4.16 i.e. as demons waiting for status values in the database to change, allowing them to act. To model this architecture however, we need to construct status attributes for the entities involved.

Consider the example of figure 3.5; the entity concerned is a sample of data from a source population. If we consider simply the possible statuses of such a sample within the 'decide normality' system, an entity state diagram as shown in figure 4.17 will result.

As part of the attribute analysis stage of the entity relationship modelling exercise therefore, the status attribute must be defined, as below :

decide-normality-status : initial / source population to be investigated / data to be tested / test results ready / normality decided.

This attribute and its possible values would be represented for example in a frame based system, as a status slot.

The general architecture of a cognitive model based on entity modelling, as illustrated in figure 4.16 is that of a production system. [Waterman & Hayes-Roth, 1978]. The production rules of the form

IF condition THEN process

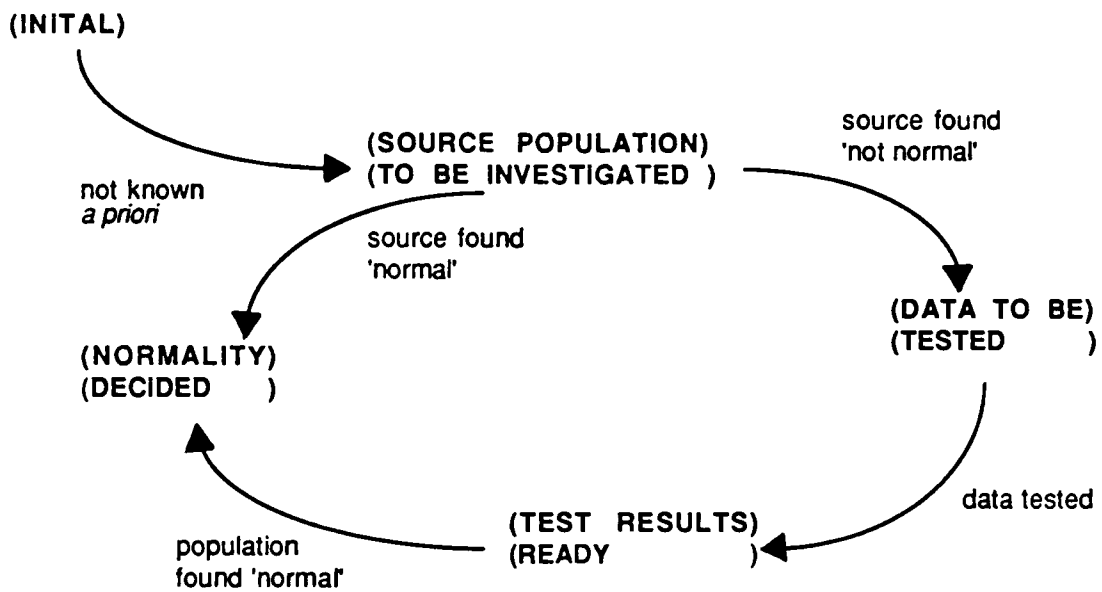


Fig. 4.17 Entity life cycle for the 'data sample' entity of fig. 3.5

search for conditions in the database, to activate processes which change the database. Production systems operate a cycle of three phases :

recognise phase : rule conditions are coupled with the database in a search for matches;
conflict resolution : a decision is made as to which rule to fire;
act phase : the process is activated.

The entity modelling approach deals primarily with the design of the database for the cognitive model, leaving the conflict resolution to be defined separately. The provision of status attributes does however ensure that some procedural control can also be built into the condition part of the rules.

This section is concluded by two examples. In the first, the entity analysis is carried out for a simple system where procedural knowledge is important whereas the second illustrates the use of entity life cycle diagrams for the less procedural task of fire risk assessment.

The former example is extracted from an operators knowledge about the control of an aluminium electrolytic furnace cell [Crittenden, 1988]. It illustrates one aspect of the interpretation of noise in the voltage across the cell, which may (among other things) be due to one of the 12 anodes on the cell being burnt out. Fig. 4.18 represents a simplified ER Model of the situation described.

The attributes for the entity types 'Cell' and 'Anode' are :

Cell : number, noise level, burn-off status.
Anode : number, replacement date.

All that the operator needs to know is encoded in the status diagram of fig. 4.19 in the form of limited data i.e. a continuous spectrum of times, noise levels etc. have been reduced to a limited set of states (for human processing).

When noise goes above a certain level a burn-off is suspected, and an operator is directed to investigate visually. If burn-off is negative then there is no need to investigate again until a

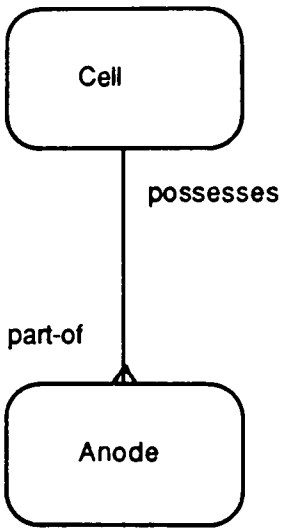


Fig. 4.18 ER Model for control of aluminium electrolytic furnace

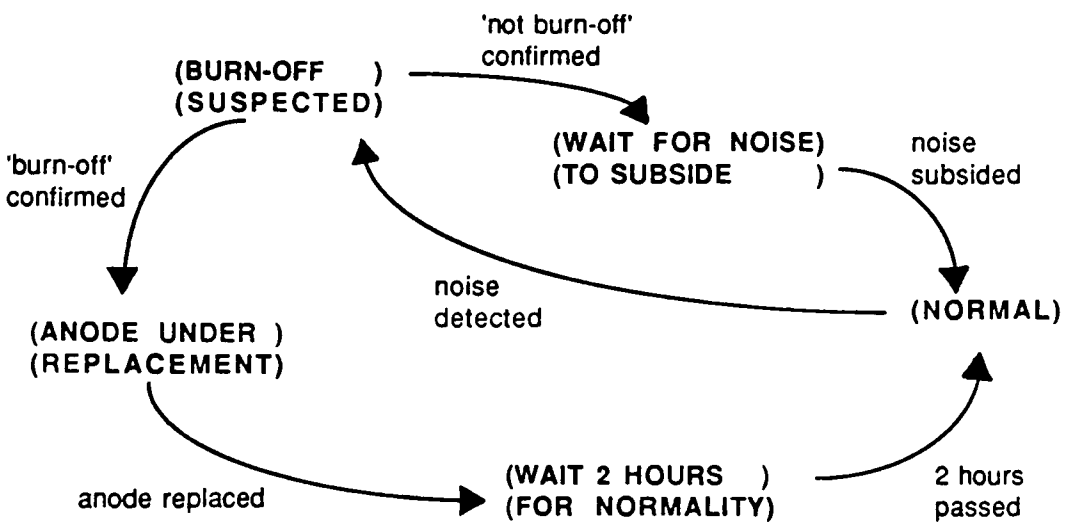


Fig. 4.19 Entity life cycle for 'burn-off status'

reasonable time (4 hours here) has elapsed. If burn-off is positive then the anode must be replaced. Noise will continue for a period after this event (2 hours here) before the operator needs to take notice again.

This procedural knowledge enters the database as a set of possible status values, and a set of constraints e.g. :

IF normal AND noisy THEN burn-off suspected

ask question

etc.

where 'ask question' is a side-effect.

There is an obvious distinction between the procedural knowledge embodied in the entity life cycle diagram of fig. 4.19 for this example, and that declarative knowledge of the previous example in section 4.31 i.e. fire risk assessment which is conveniently represented as constraints on a static database. However procedural knowledge is represented as constraints on the development in time of a dynamic database i.e. the statuses are linked to time cycles. For instance the presence of noise and 'normal' state at one time restricts the database state to be 'burn-off suspected' at the next time cycle. Again the constraints appear as rules, but relating the database at one time cycle to that at the next, via the action cycle of the production system.

IF normal AND noisy THEN state to 'burn-off suspected'

↑ this cycle ↑ next cycle

Other parts of the database are also affected besides the status attributes e.g.

IF anode under replacement AND anode replaced

THEN change status to 'wait 2 hours'

ask (anode number)

change replacement data

The fire risk assessment example illustrates the use of entity life cycle diagrams in a less procedural task, where the complexity of the diagrams is the analysts decision. The following ELC diagrams represent the status attributes identified in the attribute analysis of section 4.31.2.

Rate assessment - rate-status

Generally, the diagram of fig. 4.20 represents the change in status from 'not known' to a value $v \in V$.

We do however know the values in the domain V therefore a more detailed ELC (fig. 4.21) can be drawn which includes these values as states of the attribute 'rate-status'.

Assessment area - area-status

Again a general diagram can be drawn which represents the change in state of the attribute 'area-status' from 'not known; to 'v $\in$V' as in fig. 4.22.

However, not all assessment areas take exactly the same set of values v from the domain V i.e. some e.g. 'location' take only a partial set. Also, the events which cause the states to change will differ from one assessment area to another. Specific ELC diagrams will therefore differ as shown in figs. 4.23 and 4.24 for the Trade Processes and Location modules.

At this stage of analysis however, this detail may not be known in which case the event to change status to e.g. reject, may be simply 'reasons to reject'.

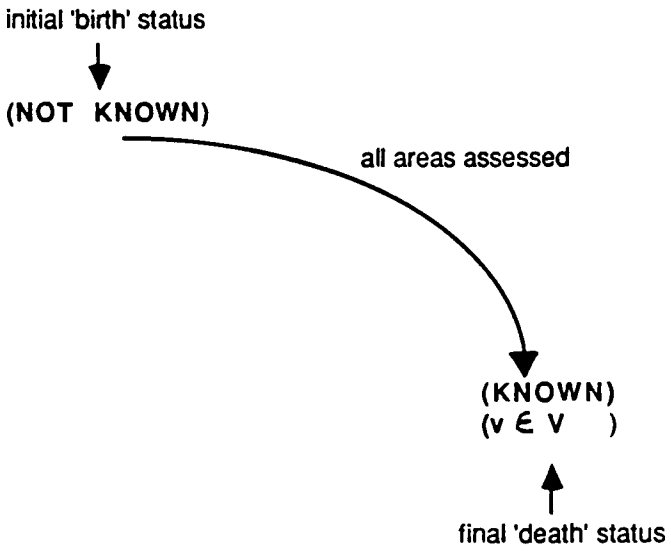


Fig. 4.20 Simplistic entity life cycle for 'rate-status'

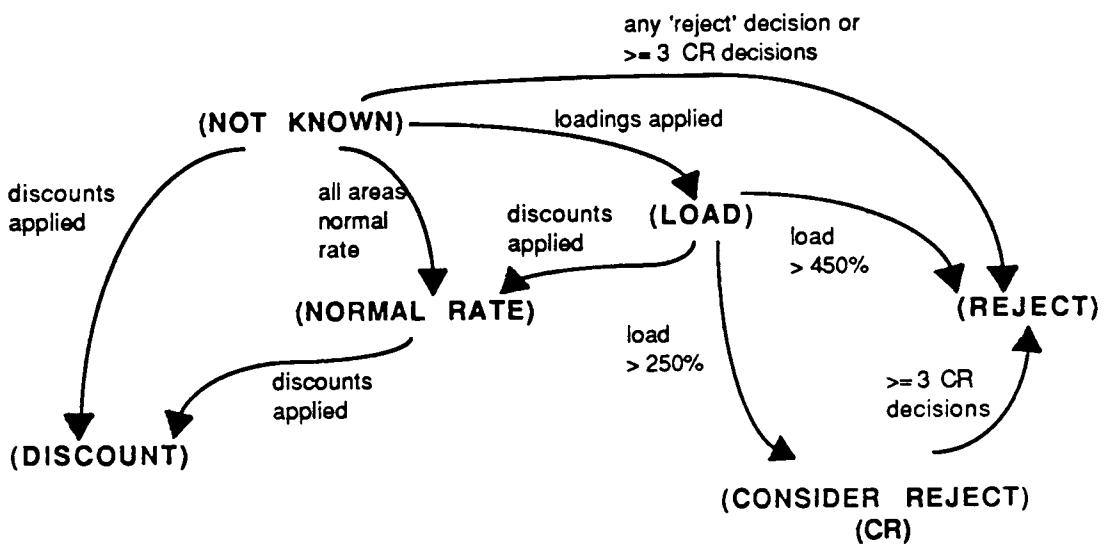


Fig. 4.21 Entity life cycle for 'rate'status' showing all possible states

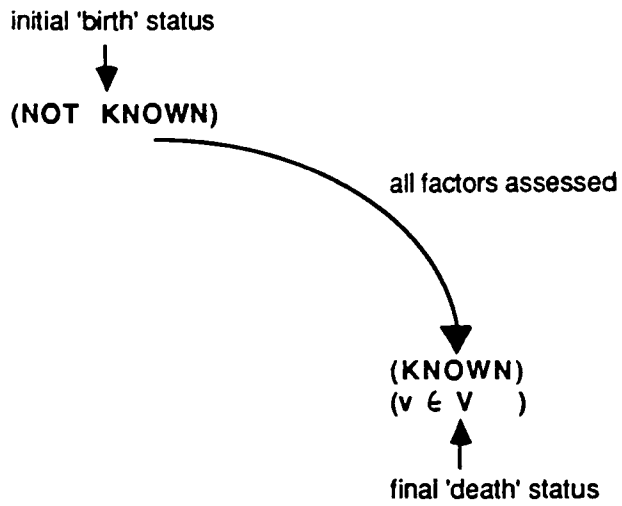


Fig. 4.22 Simplistic entity life cycle for 'area-status'

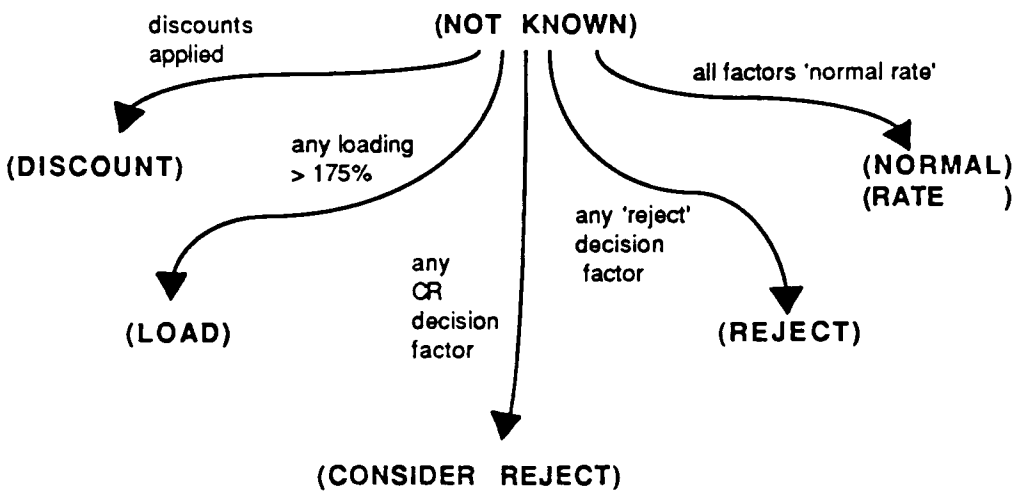


Fig. 4.23 Entity life cycle showing all states for Trade Processes Module, 'area-status'

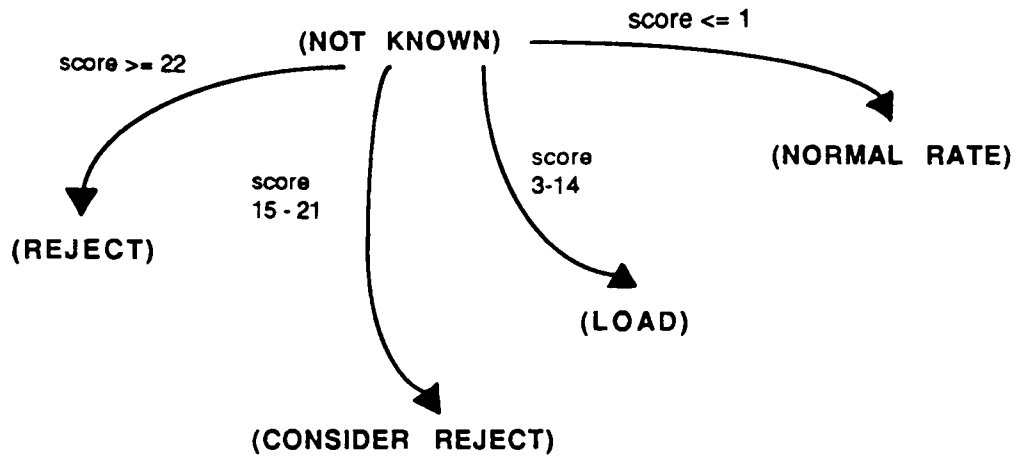


Fig. 4.24 Entity life cycle showing all states for Location module, 'area-status'

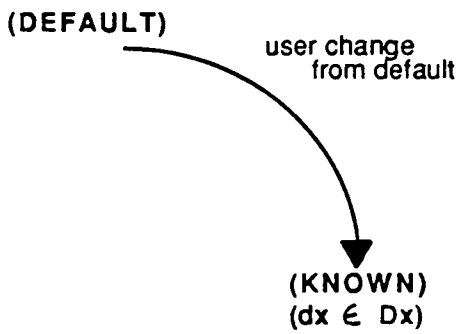


Fig. 4.25 Entity life cycle for 'factor-status'

Factor - factor-status

All factors have a default value, therefore the initial state of the attribute will be a default value from the domain Dx. The action of a user changing the default will change the state to a different value dxϵ Dx as shown in fig. 4.25.

4.5 Conclusion

The intensional knowledge of a database or knowledge base can be related to the static knowledge or data structures within the domain. In a database this knowledge consists of the data structures, and the relationships or constraints between them. Instances of data can be added to the database, i.e. the extensional knowledge is the dynamic part of the database, but the structures and constraints in the data dictionary will be static.

Similarly, in knowledge bases, there will be certain factors or 'data' existing with relationships between them, which will constitute the static part of the knowledge base. The dynamic part of the knowledge base will be those parts of the knowledge base which are instantiated at a later date. This dynamic or extensional knowledge takes two forms - the first consists of extensions or occurrences of factors, attributes of factors, or relationships which only occur for specific factors rather than generally; the second constitutes the 'data' instantiated at 'run-time' of the system.

Although there is a need to recognise the differences between data processing systems and knowledge based systems, many logical modelling methods used to design the former are also applicable to the process of knowledge elicitation. In particular, the methods of data flow analysis, entity-relationship modelling and entity life cycle analysis can be used to define an intermediate representation of the expert knowledge which is independent of the physical implementation. Structured methods offer the same advantages in expert system design as in traditional systems design i.e. a graphical communication mechanism (mainly for communication with the expert(s) as opposed to end-users) giving the benefits of :

* improving knowledge elicitation sessions, by e.g. providing a focus for discussion;

* separation of logical and physical models, resulting in the ability to identify errors in the logical model rather than the physical implementation (altering the structure of the knowledge base after implementation has begun is far more difficult).

Use of the logical modelling techniques discussed in both this and the previous chapter offers a flexible modelling technique as the required level of detail can be chosen at each stage of modelling. As with any set of tools, there is also the ability to use some of the techniques, but not others e.g. data flow analysis for partitioning, but not ER modelling, and *vice versa*. The techniques described thus form a 'toolkit', not all of whose tools would always need to be used for all problems. The main uses of each technique for knowledge elicitation can be summarised as below :

Data Flow Analysis :

- partitioning of the problem domain,

- identification of different types of process,

- identification of the explicitness boundary,

- identification of the data required at the boundary of the knowledge base.

Entity-Relationship Modelling :

- identification of objects and concepts in the expert domain, and the relationships between them,

- definition of important 'status' attributes which control the activation of 'processes' within the knowledge base,

- partial logical modelling of production systems and frame systems.,

Entity Life Cycle Models :

- definition of the 'legal' values of status attributes and the events which cause status changes,

- partial logical modelling of production systems i.e. the allowed changes in value of any attribute defined in the ER Modelling stage.

An important aspect of any analysis and design tool for expert systems, is concerned with its ability to integrate with traditional analysis and design methods. A set of systems for which the modelling techniques discussed are suitable (in particular the method of data flow modelling), are those where the expert system is to be integrated within a larger system incorporating e.g. database technology.

The final conclusion which can be reached as regards the use of the logical modelling techniques described is concerned with the ease of translation to commonly used representation schemes, as discussed in the following chapter. This offers the advantage of versatility in that the representation scheme to be used for the physical implementation need not be decided before analysis.

CHAPTER 5 IMPLEMENTATION OF THE LOGICAL MODEL

Following the logical modelling of the expert domain and task, is the design and implementation of the knowledge base. In this chapter, the interpretation of the two parts of the logical model i.e. the ER model and the ELC diagram as presented in chapter 4 are addressed, with respect to two of the most common knowledge representation schemes - production systems and frame systems. First, those two methods of knowledge representation are briefly described.

5.1 Production Systems

Production systems are the most commonly used method of knowledge representation, either alone as in, for example the shells *Crystal* and *XiPlus* or combined with other methods such as frames, as in *Leonardo* and *Nexpert*[1]. In it's most basic form, a production system takes the form of a series of productions or 'rules' of the form

IF <condition> THEN <action>

The execution cycle of a production system consists of three basic phases :

recognise: observed data is matched to the condition parts of the rules

resolve: the correct rule to fire is decided

act: the appropriate actions are carried out

---

[1] *Crystal*, *XiPlus*, *Leonardo* and *Nexpert* are registered trade marks of Intelligent Environments Ltd, Expertech Ltd, Creative Logic Ltd and Neuron Data, respectively.

A production system is therefore by nature, data driven i.e. a rule cannot be fired unless the correct data is observed.

In a knowledge based system, it is the inference engine which is responsible for the selection of rules to fire. A rule-base (i.e. a production system) is therefore not ordered in the same way as an algorithmic procedure - i.e. the rules can exist in the knowledge base in any order, and will be selected by the inference engine as appropriate.

An extension to the basic production system architecture, which is important is the ability to backward chain on the rules. As described so far, a production system is forward chaining i.e. a rule can only fire if the appropriate data is observed which matches the condition part of the rule. Very simply, backward chaining involves matching against the action part of the rule as opposed to the condition part although it is still the condition part of the rule which is ultimately tested.

As an example, the following (very small) knowledge base would be a complete knowledge base in Leonardo which uses both backward and forward chaining.

```
IF weather is good
THEN  entertainment is picnic

IF weather is bad
AND person is adult
THEN entertainment is museum

seek entertainment
```

The goal to prove is 'entertainment', (indicated by the 'seek' directive) and the possible values which it can take and which are found on the action side of the rules, are 'picnic' and 'museum'. Leonardo will first investigate 'picnic', and looking backwards will try to find the rules which result in the action 'picnic'. In the above case there is only one rule, with the condition 'if weather is good', which can only be proved by asking the user. On entering a

value, the system will then forward chain, in order to reach the goal. If, the user entered the value 'bad' for weather, the selected rule would fail on firing i.e. the condition part of the rule would not match. The above process would therefore be repeated for the next goal value, 'museum' and will therefore attempt to fire the next rule in our sequence above. The first condition will in this instance match, and if the user enters 'adult' for the asked value of 'person', the rule will be fired, resulting in a value 'museum' for the goal 'entertainment'.

Production systems have been further extended to allow the inclusion of an element of uncertainty into the reasoning process. For example, the MYCIN system for the diagnosis of blood disorders [Buchanan & Shortliffe, 1985] uses a system of certainty factors attached to the rules, and the Prospector system for mineral exploration [Duda, Gaschnig & Hart, 1979] uses a modified form of Bayes rule as part of the method of inference.

Rules in a Prospector-type system and as employed in the expert system building shell Micro Expert [Broughton & Cox, 1983], take the following form :

IF E

THEN (to degree LS, LN) H

where E denotes the presence (or absence) of some evidence, LS (Logical Sufficiency) and LN (Logical Necessity) (both optional depending on the type of rule) are weighting factors, and H is the hypothesis (assertion), the odds of which may be either increased or decreased by the evidence E.

5.2 Frame Systems

The notion of frames introduces a structured method of representing "chunks" of related knowledge , as opposed to a series of unrelated fragments [Minsky, 1975]. Each frame is a data structure which holds information about a stereotyped object or event (the latter case is often referred to as a 'script') and a collection of frames forms a network of nodes and relations, called a "frame system".

Each frame has associated with it a number of slots which can take the form of straightforward facts i.e. properties of the objects being described, rules, results (dynamically updated) or pointers to other frames.

A significant feature of frame systems is the notion of property inheritance. As an example, a frame which describes a dog will have property slots such as 'legs', 'hair colour', 'temperament' etc. which can conceivably have default values of '4', 'brown', and 'gentle'. This frame is known as a Class or parent frame, and it defines a class of objects of one type. A number of Member or child frames can be defined which can either take the default values for the properties defined, or can be given individual values. A member object 'Alsatian' of the Class 'dog' may therefore be defined which takes the default values for the slots 'legs' and 'hair colour', but which is given a different value, 'vicious' for 'temperament'.

Frames are used in knowledge based systems as part of a hybrid knowledge representation scheme, together with production systems e.g. in Leonardo and Nexpert. Frames define the static objects whereas production systems allow these objects to change state and value.

5.3 Interpreting the ER Model

5.31 Production Systems

General Relationship

Production systems can, and indeed have been, implemented in many different languages (both conventional (e.g. Pascal), AI (e.g. Lisp, Prolog), and ad hoc (i.e. a specific rule language as in most shells)). The notation used here is Prolog.

Prolog is by nature, a declarative language where the program is written as a set of 'facts' (the database) and rules which are applied in order to solve the problem. In order to program a problem in Prolog therefore, the first task is to define the database in which all 'objects' are defined and any relationships which exist between these objects. The database which can be

both queried and added to during a consultation, consists of a series of 'facts' about the domain. These facts take the form

predicate(arg1,arg2 ...)

where a predicate is either

i) a straight-forward fact-predicate which either

a) defines an instance of an object,

i.e. predicate as entity type, arguments as entities,

b) defines a property of an object

i.e. predicate as attribute, arguments as entity types, entities or values,

c) indicates a relationship between objects.

i.e. predicate as relationship, arguments as entity types or entities.

or

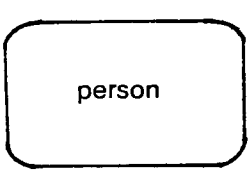
ii) a function-predicate which specifies the treatment of the arguments. Built-in functions such as "add" are used in this way.

The arguments can themselves be facts (the result being a nested list of facts) or values of defined objects, or predicates.

In relation to an ER model, the predicate in a fact such as

person(john)

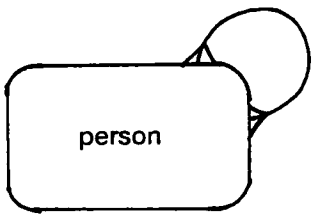
is defining an occurrence or entity of the object (or entity type).



In a fact such as

likes(john,mary)

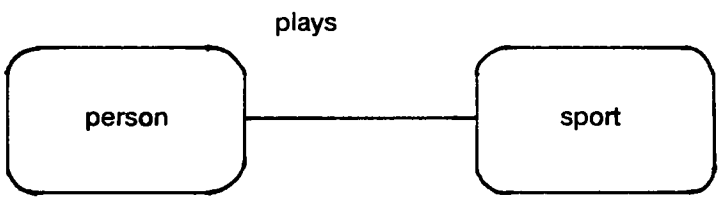
the predicate 'likes' is describing a relationship between two entities of the same type i.e.



Relationships between entities of different types are represented in the same way e.g.

plays(john,golf)

is describing an occurrence of the following relationship:



The uncertainty element of production systems forces us to in fact look the other way at this point, to check whether, in fact, uncertainty could be represented on the ER model. For a fact such as that above, the ER model conventions could be extended to include uncertainty in such a way that the relationship took, in addition to a name, an attribute such as 'degree'. The above relationship 'plays' could therefore be an uncertain relationship in the sense that different people play sport to different degrees. Similarly, entity types could also have certainty measures or 'weightings' attributed to them. In the plant diagnosis example, the entity type disease-state reflects the diseased state of a plant with respect to a specific

disease. An extra attribute could be introduced which indicates the degree to which this suffering exists in the plant.

Example

Using the Fire Risk Assessment example, for which the ER model was developed in 4.31 it is shown here that it is a short step from the graphical ER model to production system formalism.

Static relationships between entity types

Relationships on the ER model such as

rate-assessment	*calc-from*	assessment-area
factor	*applied-to*	assessment-area

can be represented by the following clauses in e.g. a Prolog type syntax :

calc-from(rate-assessment, assessment-area)

applied-to(factor,assessment-area)

Static constraints can also be defined in order to preserve integrity, such as:

default-value(f,d) => possible-value(f,d)

The latter clause forms an integrity constraint which ensures that the discrete-value (d) forming the default value of a factor (f) must belong to the set of discrete-values which form the possible values for that factor. The above constraint implies the existence of the relationship defined by :

posesses(factor, discrete-value)

Attributes of entity types

For each entity type in the ER model of fig. 4.7, the attributes can be defined as follows :

factor(name,description,factor-status, ...)

assessment-area(name,area-status,numeric-result,verbal-result, ...)

Dynamic Constraints

In some cases, consistency constraints will be required at run-time of the system e.g. integrity constraints for data entry. The following is one such dynamic constraint :

actual-value(factor,d) => possible-value(factor,d)

which states that, as above for static default-values, the value taken by a factor at consultation time is restricted in value to a discrete value belonging to the set of possible values for that factor.

5.32 Frame Systems

General Relationship

A frame describes an entity type i.e. an object or concept within the expert's task domain. In the same way that an entity type has associated with it one or more attributes, a frame has a number of slots which represent the properties of the frame.

Although an entity type will always have associated with it at least one entity i.e. an occurrence of that entity type, a frame may not always have associated (child) frames or instances. A child frame which inherits properties from the parent frame will only be created if there is to be a 'static' database of details of 'objects' of this kind. An instance will only be created where one or more slot values will be dynamically changed. An instance is therefore a specific occurrence of a frame which will have a number of slots, each instantiated to a specific value (either statically or dynamically), representing a unique object.

A slot may have either a single value or a series of possible values associated with it. In the former case the value is a static part of the frame; in the latter an instance of the frame will have a slot instantiated to one of the possible values. An attribute in an ER model will

however, always have a set of possible values. Similarly, a slot may take the form of a rule, pointer or procedure.

A relationship exists between two or more entity types, and in a database application will represent a constraint on the database. There is no direct equivalent in the frame system, although relationships clearly do exist between frames e.g. a plant shows diagnostic signs - an instance of the plant frame, and instances of the diagnostic frame, will be linked in a relationship by means of a list of relevant diagnostic sign instances being stored in a slot of the plant frame instance. This could be described as a constraint on the knowledge base, but in other cases, the form of the constraint may vary therefore a more general method of representing such relationships is required.

Inheritance, which is made explicit in a frame system is partially implicit in an ER model. As described above, the notion of entity types and entity occurrences implies the inheritance of properties (as *via* an Is-A relationship in a frame system) i.e. all entities of the type person will have attributes for, say, hair colour, age, sex etc. The types of inheritance which are difficult to explicitly represent using ER models are multiple and partial inheritance, (via AKO (a kind of) links in frames). Multiple inheritance means that a member frame may inherit from more than one class frame, (e.g. a frame 'dog' may inherit some of the properties from two more general frames, 'animal' and 'family pet'). Partial inheritance [Graham & Jones, 1988] means that a member frame may only inherit a property partially i.e. to a degree (e.g. it may be desirable to indicate that a particular dog inherits the property of 'gentle nature' only to a certain extent). (It is worth noting here, that this is an aspect of frames which is not as yet supported in commercial products.) Both of these forms of inheritance could be represented on an ER model, but the representation of partial inheritance would require the relationships to be labelled with the degree of the property inheritance.

Example

Each entity type on the entity model for plant diagnosis of fig. 4.11 is represented by a frame in the frame system e.g. 'frame plant', indicating a Class. Occurrences of entity types are

defined using instances or Members of the defined (Class) frames, which describe specific, unique objects/concepts of the type described by the frame.

Attributes of entity types are represented within each frame by slots which have a type and/or a range of possible values. Within each instance created the value of each slot becomes instantiated (statically or dynamically) to a specific value from the possible range.

Frames

frame plant.

slot parts type list of instance of part.

value parts = [foliage, stem, ...].

slot states type list of instance of disease-state.

end frame.

The 'parts' slot represents the one:many relationship between 'plant' and 'part'. As all plants are assumed to have all parts, this slot is given a value within the frame definition. Similarly, the one:many relationship with 'disease-state' is shown by the states slot, but no value is assigned to this slot in the frame definition, as it will vary for each instance of a plant.

frame part.

slot diagnostic-signs type list of instance of

diagnostic-sign.

end frame.

A part of the plant may show one or many diagnostic signs, hence the 'diagnostic-signs' slot consists of a list of diagnostic sign instances (different for each instance of the 'part' frame).

```
frame diagnostic-sign.
        slot description type text.
        slot present type boolean.
        ds-present present::if needed present.
                        ask.
        end rules.
        slot ds-present-effect type list of instance of disease-sign-effect.
end frame.
```

In this frame, the slot 'present', is a boolean slot representing the status attribute which will cause a question to be asked about the presence of the diagnostic sign. The slot 'ds-present-effect' forms the one:many relationship with the 'disease-sign-effect' entity i.e. there will be a list of instances of 'disease-sign-effect' representing the effect each diagnostic sign has on each disease (i.e. positive (1) or null (0)). An instance of the diagnostic-sign frame will reference only those disease-sign-effect instances relevant to that diagnostic sign.

Instances

An instance of plant, sick-plant has in this example 2 disease-states, one for each of the diseases d1 and d2.

```
instance sick-plant of plant.
        value states = [ds-present-d1-sick-plant, ds-present-d2-sick-plant].
end instance.
```

The part instances each have the diagnostic-sign slot instantiated to only those diagnostic signs relevant to that part of the plant.

```
instance stem of part.

        value diagnostic-signs = [weak stem, stem rot].

end instance.

instance foliage of part.

        value diagnostic-signs = [tiers-of-leaves,
                                  missing-leaves,
                                  brown-leaf-margins,
                                  spots-on-foliage,
                                  fungus].

end instance.
```

Associated with each diagnostic-sign instance is the question to ask the user in order to see whether or not the symptom is present. Each instance also has a list of relevant disease-sign-effect instances e.g. for the first diagnostic sign 'tiers-of-leaves', only those disease-sign-effect instances with 'tol' in the name are referenced.

```
instance tiers-of-leaves of diagnostic-sign.

        value description = [...    ].

        value present%question =

                "Does the plant have widely spaced leaves or tiers of leaves ?"<>n1.

        value ds-present-effect = [dse-d1-tol, dse-d2-tol].

end instance.
```

```
instance weak-stem of diagnostic-sign.
        value description = [...    ].
        value present%question =
                "Is the stem or trunk of the plant weak or   spindly ?"<>n1.
        value ds-present-effect = [dse-d2-ws].
end instance.
```

Relationships

The relationships on the ER model have been described in the above sections which have defined the frames and instances required for the system.

Inheritance

The instances in this example clearly inherit properties from the frames defined, for example, the instance 'sick-plant' of type 'plant' inherits the slots 'parts' and 'states'. The former slot is already instantiated in the frame definition for 'plant' and therefore all members inherit the same list of parts. The latter slot is not, however, defined in the frame definition but is defined for each instance. The instance is thus only inheriting the slot type, but not the default value.

5.4 Interpreting the ELC diagram

Whereas the ER model is interpreted mainly as the static parts of the knowledge base, the ELC diagrams model primarily, the control of the knowledge base, and are thus dynamic in nature.

Of the two knowledge representation schemes discussed, it is only production systems which allow any change in status of the knowledge base. Frame systems define static structures which cannot change or be created without the use of another knowledge representation scheme such as production systems. In this section therefore, only the interpretation of ELC diagrams to production systems is investigated.

Taking the ELC for factor-status (fig. 5.1) as an example, the effect of user input on the status of the knowledge base can be represented.

The events which cause a change in status of the knowledge base are :

E1) User enters a correct and allowed value for the factor.

user-entered(f,x) & possible-value(f,x) & factor(f,unknown) => set-factor(f,known)

E2) User enters a 'don't know' answer.

user-entered(f,'don't know') => set-factor(f, unknowable)

E3) User enters an incorrect or disallowed value for the factor.

user-entered(f,x) & ~(possible-value(f,x)) & factor(f,unknown) => set-factor(f,unknown)

The above constraints are in fact defining part of the control mechanism of the knowledge base. This becomes clearer if we look at the ELC for rate-status (fig. 4.20).

Once all assessment areas have been evaluated, their area-status values will be set to 'known', and the following 'rule' would be triggered :

area-status(a,known) => set rate-status(r,known,)

Similarly, if there are any areas which cannot be evaluated (for example if factors are 'unknowable'), the following 'rule' would be fired :

area-status(a,unknowable) => set rate-status(r,unknowable)

5.5 Conclusion

In this chapter it has been shown that the use of a logical model involving both ER models and ELC diagrams is advantageous for the design of knowledge based systems. The ER model, which defines both static knowledge about objects or concepts and dynamic

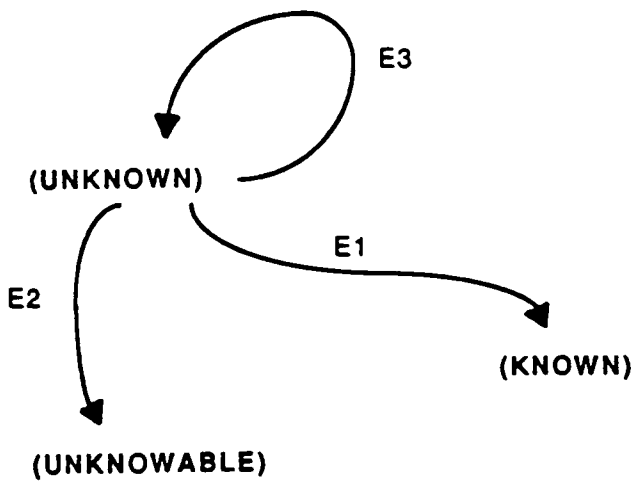


Fig. 5.1 Entity Life Cycle for status attribute 'factor-status'

constraints, maps closely onto both Prolog type systems where a 'fact' database is set up, and onto frame systems, where the frames define the static objects.

Dynamic relationships as included on the ER model for Fire Risk assessment in section 4.31 have been represented in this chapter by the use of production rules. Production rules are used both on their own in e.g. a Prolog knowledge base, and as part of a mixed representation system, for example with frames or Bayesian networks. The dynamic constraints are thus not represented within frames or instances.

Entity Life Cycle (ELC) diagrams model the status of the knowledge base according to specific status attributes linked to key entity types. As such they are concerned with the control aspects of processing and therefore the knowledge held within them cannot be represented in frames or, say, semantic nets. Although when developing a knowledge base, the emphasis is not on control, but on the declarative aspects of the knowledge, it is important to know how rules or rule sets will be fired, i.e. the different states in which the knowledge base will be, and the events which will cause these states to change. Production systems offer a convenient way of representing this process activation control knowledge, as shown in 5.4 by use of the Fire Risk Assessment ELCs.

CHAPTER 6 UTILITY OF A DATA-ORIENTED METHOD IN PRACTICE

6.1 Introduction

In this chapter a practical example of the use of a data-oriented approach to elicitation is presented. The method used, domain analysis, had the following specific aims :

* to analyse the expert domain thoroughly thus forming a precursor to functional elicitation,

* to involve users in the development process with the view to reducing user dependency,

* to take a data oriented approach to the problem of elicitation and subsequent formalisation of the knowledge,

* to restrict the input data of the system to limited data sets.

The practicality of such an approach to knowledge elicitation was tested by application to the ARIES (Alvey Research into Insurance Expert Systems) project carried out by Logica (UK) Ltd [Butler, 1987]. The aims of the involvement with this project were to test the ideas behind, and thus improve, the method and in particular, to test the importance of involving users with respect to the issue of user independence.

In the subsequent sections, the ideas on which the method of domain analysis is based are discussed i.e. where in knowledge elicitation domain analysis fits, the relevance of an intermediate representation, and the importance of involving users. Details of issues such as user independence can be found in other chapters. The two main phases of domain analysis, i.e. definition and testing are then described with reference to their application to the ARIES Fire Risk system [Logica, 1986].

6.2 Domain Analysis

If we look at the whole process of knowledge elicitation (the acquisition and formalisation of expert knowledge), analysis of the domain forms one part of this process (fig. 6.1) falling between the initial high-level analysis of the domain which pinpoints and/or evaluates the expert system application, and the functional knowledge elicitation which defines the functions of the knowledge base which will emulate the expert reasoning process.

The first stage is equivalent to a functional specification where the boundaries and goals/decisions of the system are defined and the specific purpose (i.e. what not how) identified. The final stage, that of functional elicitation uses the model developed in the middle stage (detailed domain analysis) as a basis on which to build the functions.

The proposed method of domain analysis is a structured method (based on the idea of IPO (input, output, processing) methods) which results in part of an implementation independent Intermediate Representation (IR) or model of the experts problem solving domain. This part of the model is here called the 'domain model'.

The approach focuses not solely on how the expert solves the problem (functionally or otherwise) but on how he would define for a user (with less knowledge/expertise) the factors within the domain. The domain model is therefore developed at the user level of understanding and knowledge, and could be viewed as the expert's method of communication with this level of user. The method therefore requires a high level of user participation in order to achieve this.

The domain model forms the definition of the input to the problem solving task, with each factor defined in terms of a limited data set on which the users can discriminate as well as the experts. This model will also therefore form the input to the resulting expert system and will be redefined until both i) factors are defined at the correct (i.e. user) level of knowledge/understanding of the problem, and ii) data set values are defined at the correct

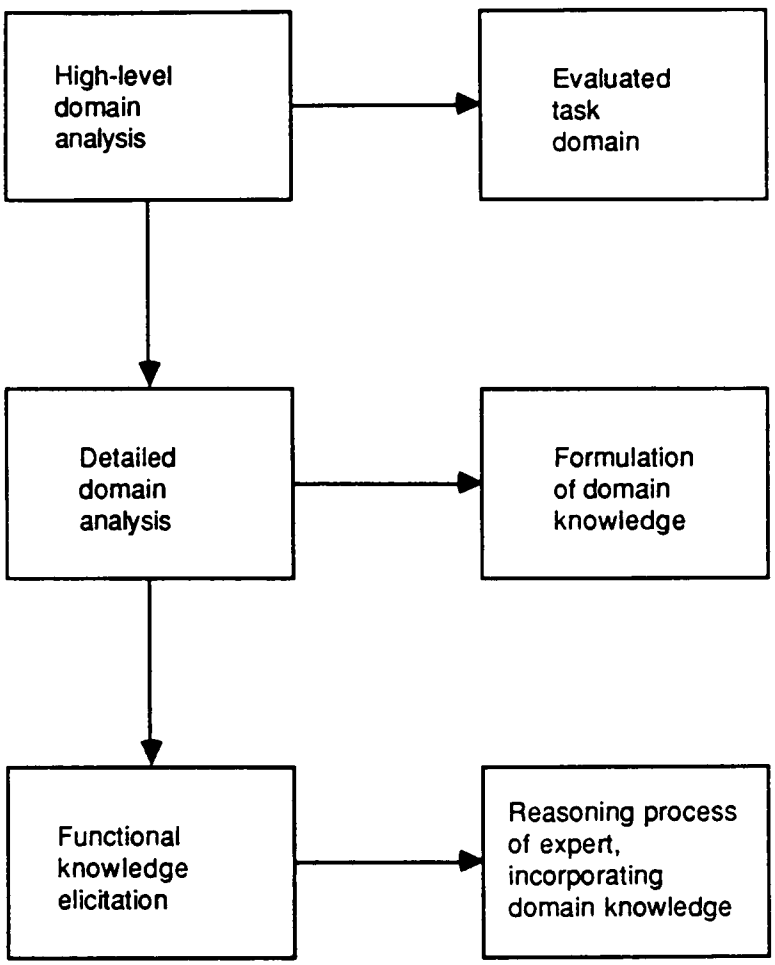


Fig. 6.1 Stages of knowledge elicitation

"grain size" (i.e. the number of points on the scale which the user must be able to discriminate).

Domain analysis as the precursor to the functional stage of knowledge elicitation, has the major objectives of:

i) ensuring that the output of this analysis i.e. the domain model is defined at the user level of knowledge and understanding in order that the functions built using this model will result in a usable and acceptable system with which the users can reliably perform the expert task.

ii) approaching the task of knowledge elicitation from the same angle as do conventional IPO methods for systems design i.e. defining the input and output of the system first, before modelling the processes which emulate the problem solving tasks of the expert(s),

iii) ensuring that the knowledge engineer understands the concepts he is working with, thereby making the functional knowledge elicitation easier,

iv) formalising the domain knowledge.

Fig. 6.2 outlines the detailed domain analysis process, each phase of which will be discussed in a separate section. As the diagram shows, the process will almost certainly be an iterative one, due to the involvement of users as well as experts.

Envisaged advantages of such a method are outlined below :-

i) In domains where user dependence is likely to be a problem i.e. where

- the level of knowledge/expertise of the intended users is significantly different to that of the experts or
- the discriminatory ability at these levels is significantly different or

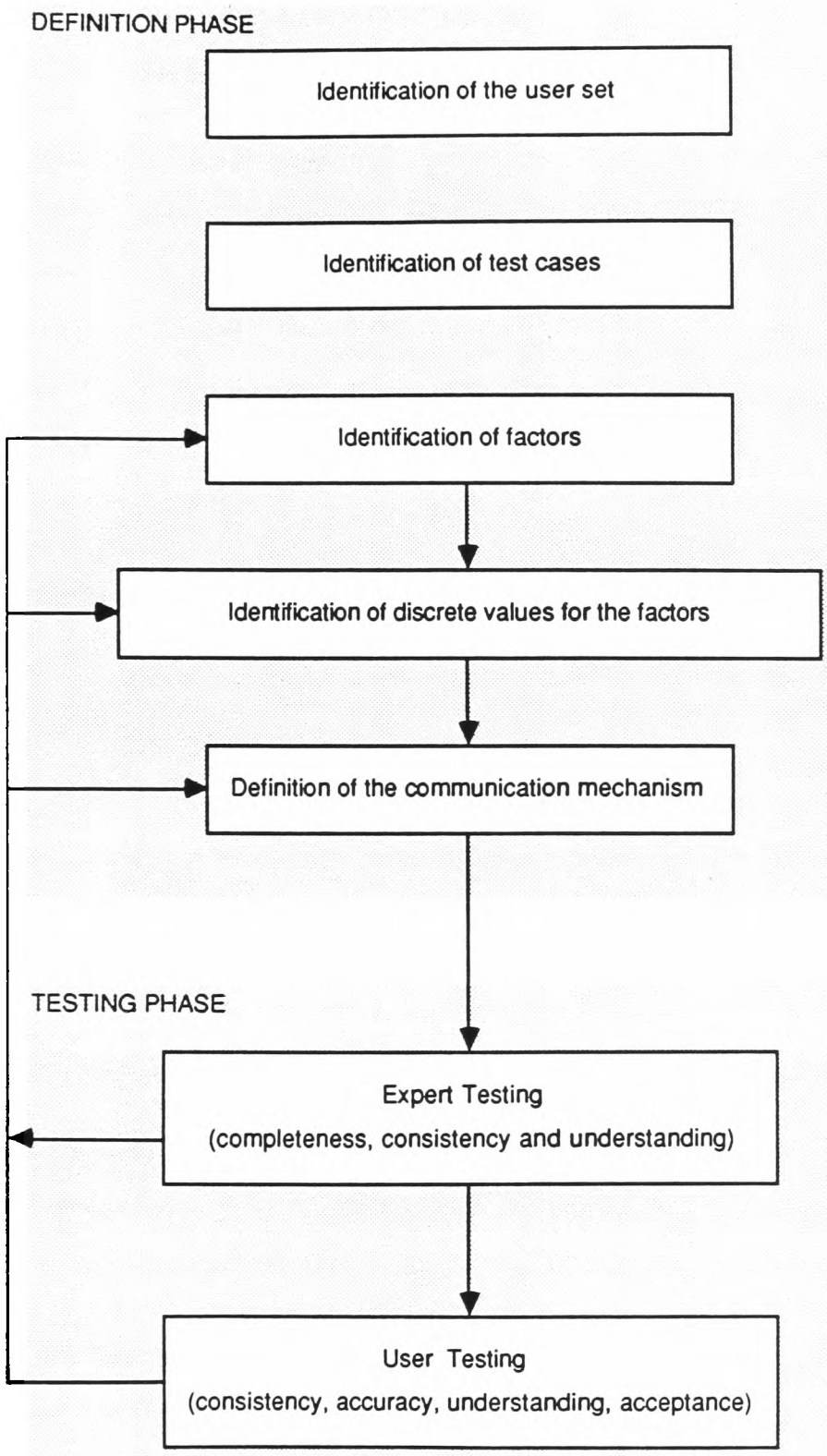


Fig. 6.2 Phases of Domain Analysis

- the input to the problem task requires judgement or is open to some interpretation,

the method aims to reduce or eliminate this dependence by defining the user input to the system in terms which the users understand and can discriminate. The method concentrates on input rather than functions because, if we consider the reasoning process or functions to be constant (like an algorithm), then the only factor which can affect the output of the system is the input data. The conclusion which the system will reach will therefore rely heavily on how the users interpret the input data, unless it is defined at their knowledge level.

ii) The use of limited data sets for input as opposed to continuous scales,

- enables achievement of the above as the values are defined according to the discriminatory ability of the users, and

- results in a potential simplification of the reasoning process and therefore greater ease in the functional elicitation stage of knowledge elicitation.

iii) The method provides a structured approach to analysing an expert's domain.

6.21 Definition Phase

Following the initial stage of knowledge elicitation i.e. the high-level domain analysis where the task domain is evaluated for suitability for an expert system, the definition phase is the first of the two main phases of detailed domain analysis. The definition phase can be carried out independently of any functional elicitation, and likewise of implementation.

Involved in this phase will be the knowledge engineer(s) and the expert(s). For the Fire Risk (FR) system, the major expert role i.e. supplying the bulk of the expertise, was taken by one expert in the appropriate field, with "contributory" experts being involved for review sessions where they could offer criticism and suggestions.

In summary, the task of the definition phase is to identify and define :-

- the user group
- representative test cases
- factors/concepts
- limited data sets for the factors
- expert communication mechanism

The output resulting from this phase will be a "paper" (rather than computer implemented) model of the knowledge within the domain, expressed in terms of what the expert thinks about but not how he uses this knowledge specifically. No functions will therefore be represented explicitly, although they may be implied by the model.

This model should be expressed at the knowledge level of the potential user group of the system (where this differs from that of the experts) i.e. both the factors and the limited data sets associated with them should represent how the expert would communicate his understanding of the problem task to a less expert "user".

In accordance with current ideas in systems analysis for conventional data processing applications, there are strong reasons for analysing the problem domain thoroughly before looking at the expert functions and trying to emulate them in a computer system.

De Marco [1978] identified three principle reasons for carrying out such a thorough analysis :

- projects do not get started because there is no agreement as to what to do,
- projects do not get finished,
- projects deliver unusable products.

The latter of these problems was taken up by Mumford [1978] who advocated a participative approach to system design where the users are involved almost to the degree of designing the system themselves. The main emphasis of the approach is on the user interface of the

final system and the way in which it will fit into the users environment - both clearly important for the acceptance of any computer system.

The data analysis approach to systems analysis was based on the idea that the data elements (entities and their attributes) within a particular problem domain would describe the nature of that problem. Whereas this data is static, the functions which use it are more likely to change with time and are therefore not as good a basis for modelling the problem.

Once the basis of the "system" in which the problem exists has been defined, the task of designing the necessary functions to solve the required problem is more simple. A thorough understanding of the problem area is therefore gained by analysing the data and the relationships between the data elements.

These structured approaches concentrate heavily on the use of tools and techniques for analysis, the use of graphical documentation techniques, data dictionaries etc. for documentation and communication. De Marco's data flow diagrams describe the flow of data in a system, only elaborating on functions once this flow has been defined. Similarly, entity modelling models the problem environment by mapping entities (objects) and their relationships before defining any functions.

Lessons learned from the early years of systems analysis can be used in developing expert systems, in particular in the knowledge elicitation stage of system development. Structured, data oriented methods for analysing the domain, in which users are involved should lead to systems which are both usable by the intended users, and robust due to the thorough analysis of the problem domain.

In addition, the result of a thorough analysis of the domain from a data-oriented point of view, should be to make functional elicitation an easier, and quicker task. The resulting knowledge base should be more complete and concise due to the formalisation and structuring of the domain knowledge before functional elicitation.

In summary, the aims of the definition phase are :-

- to thoroughly analyse the domain and formulate the knowledge in a computer-independent and expert/user understandable fashion, before functional elicitation
- to simplify and speed-up the process of eliciting functions
- to help towards the development of a knowledge-base which is complete and concise
- to deliver a usable expert system, designed at the correct level of understanding for the specified users.

The form and purpose of the domain model which results from the definition phase is shown in fig. 6.3. The model forms part of the Intermediate Representation (IR) which lies between the experts knowledge and the knowledge as represented in the expert system. At this stage, the IR is however incomplete as no functions are represented, and is thus called the domain model.

6.21.1 User Involvement

i) Importance of Users

The importance of involving the potential users in the development of a computer system has been a subject of study for many years. During the 1960's many unsuccessful projects resulted from bad user-analyst relationships, where the users were not involved in the development and design of the systems, but were forced to participate and comply with the analysts [De Marco, 1978].

Conclusions drawn from this period were that the user must be interested in the project and willing to participate rather than forcefully be made to use systems they have not been involved in the development of. It is therefore important that all potential levels of users

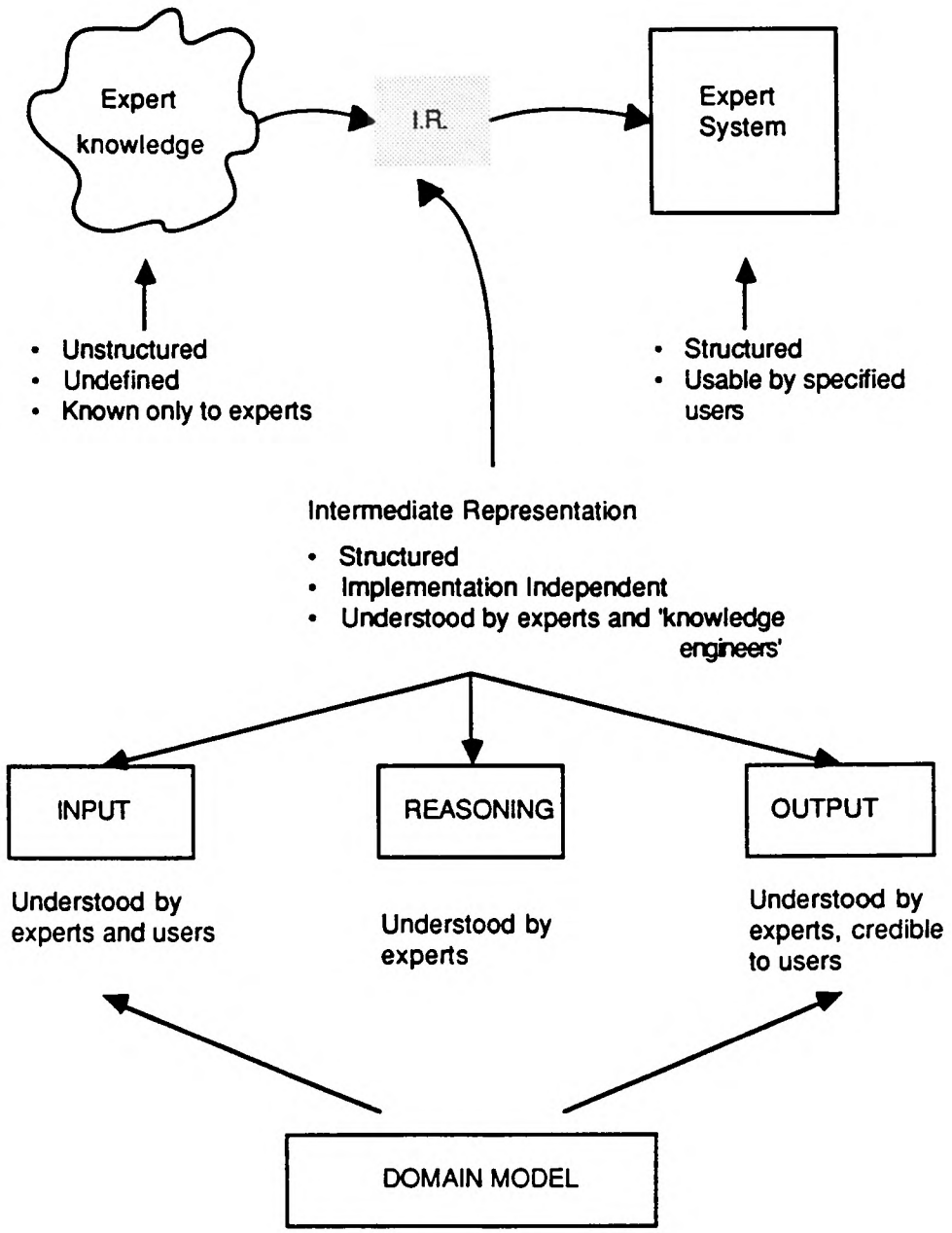


Fig. 6.3 Form and purpose of the domain model resulting from the Definition Phase of Domain Analysis

should be involved in the development of any computer system in order that the resulting system is usable and accepted by those users.

The field of expert systems is no different in this respect from that of conventional data processing systems. An expert system is designed to incorporate the expertise of one or more experts, but it will not necessarily be used by people with the same level of expertise.

As in systems analysis therefore, the potential users of the expert system should be involved in the development of the system in order that a usable and acceptable system is developed. If such an approach is not taken the result will almost invariably be a system where "... questions may be difficult to understand, ambiguous or poorly stated for the users of the expert system" [Hayes-Roth *et al*, 1983]. A lack of understanding of the system may also lead to a significant drop in accuracy when the system is used in the user environment. This is due not necessarily to the system being incorrect but to the users answering questions wrongly due to a lack of understanding at the required level.

It is not expected that the users (if of a different level of ability than the experts) should be able to carry out the expert task (unaided) but that they should be able to understand the concepts involved and jargon used etc. within the problem domain (and therefore the system itself). The users understanding should therefore be of prime interest and importance when eliciting knowledge from the expert for incorporation in the knowledge base of the expert system.

In addition to taking into account the level of user understanding, the user-interface of the resulting system must be specifically designed for ease of interaction between the specified level of users and the expert system. The users should also therefore be involved at this stage of system design. The design of the user interface should in fact be made easier if the users are involved at earlier stages of system development, as the system itself will have been built upon concepts understood by these users.

ii) Definition of User Groups

Most organizations or professions have an inherently hierarchical structure, which may be representative of many factors such as :-

- age
- knowledge
- skill
- ability
- experience
- salary

  etc.

At each level within the hierarchical structure there will be a group of people, with each group increasing in size from the top of the hierarchy to the bottom, thus forming a pyramid. As an example, a hospital will have a very small number of specialist doctors with very specific skills, a larger number of house surgeons and possibly an even larger number of student doctors with less experience, skill etc.

In order to form a group, the individuals in the group must have a common objective and possess certain characteristics in order to belong to that group. In the case of groups belonging to specific professions such as the medical profession, the criteria for membership would be qualifications, level of knowledge and experience, ability, skill etc.

The diagram of fig. 6.4 illustrates the structure of such groups.

The top level of the hierarchy represents a few people with a high level of skill and experience, who will probably be referred to as the experts in the field, e.g. specialist doctors.

At each level of the pyramid below this top level, there will be an increased number of people with a lower level of experience, skill etc., e.g. house surgeons, general practitioners, student doctors etc.

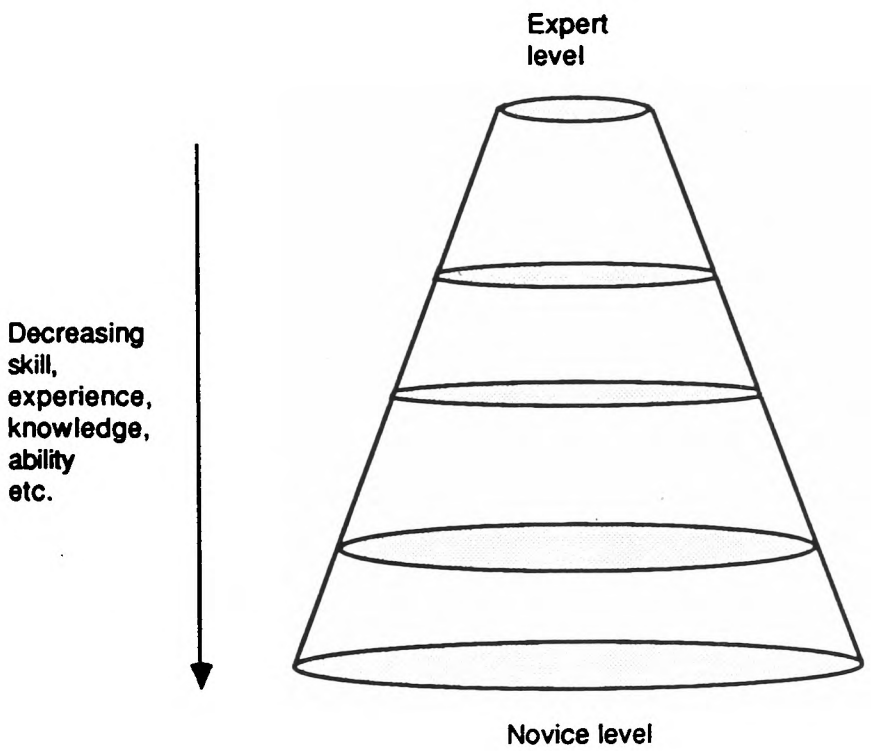


Fig. 6.4 Expert-novice hierarchy

Finally, at the bottom of the pyramid another level can be added which represents people outside the organization or profession who have little or no experience or skill in the given area e.g. the general public.

When developing an expert system the expertise which forms the core of the system will invariably be elicited from the top level of the pyramid i.e.from the experts in the field. (The exception to this is machine induction of rules from examples, where the expert role is one of verification and refinement rather than provision of rules.)

Early on in the development of the system the level within the pyramid of the target users must be selected. A representative group of these people will then be required to act as the user group for the development and testing of the expert system. All members of this user group must possess similar characteristics in terms of level of skill, experience, knowledge etc. and should be interested in taking part in the system development.

The expert system can then be aimed at this level of skill resulting in a system which

- has a degree of user-independence due to the involvement of a representative group of users of the same level of expertise
- is understandable to and accepted by the users in terms of concepts, jargon etc.,
- has a suitable user-interface.

It is important to stress that the involvement of the users in knowledge elicitation is restricted to the domain analysis phase only, and does not stretch to the functional elicitation stage. The latter involves eliciting the functions or rules which will form the basis of how the system solves a problem and should therefore be elicited from those people with the relevant expertise i.e. the experts. These functions will be elicited using the domain model obtained from carrying out the domain analysis on the experts task domain. This process should in fact be greatly aided by the domain analysis phase, in which the task domain is analysed in terms of factors or concepts which the expert uses to solve a problem.

The users will be involved in testing the domain model in order to produce a model of the expert domain at their knowledge level. As the elicitation of functions for the system will be based on this model, the resulting system should incorporate the expert knowledge and expertise, but in a form i.e.level which the users can understand and use to their advantage.

iii) Identification of the User Group

The identification of the user group is a task which should be carried out primarily by the expert. When the boundaries and purpose of the system have been defined, the level of users (i.e.novice, semi-professional, professional, expert etc) at which the system is targeted should also be defined.

In the definition phase however, the identification of the users must be more precise and the characteristics of the group defined by the expert. The criteria of membership to this group are likely to be such characteristics as mentioned before i.e.

- knowledge
- qualifications
- experience
- age (possibly)
- enthusiasm

  etc.

A representative group of such users should then be selected to take part in the testing phase of domain analysis, and possibly in later stages of system evaluation (i.e.after functional elicitation and implementation). If a prototyping method is being used, then the users will be involved in this way for every prototype, ideally, but should at least be involved in testing one of the early prototypes.

During the early stages of the Aries Fire Risk project, the type and level of potential user was defined by the experts. The users, some of whom would take part in the later stages of

testing the 'paper model' developed, were selected to have the same set of characteristics i.e.

- ACII qualified,
- experienced in underwriting fire risks,
- professionals at branch level (as opposed to experts or novices).

6.21.2 Discriminating Factors

i) Definition

A discriminating factor is an object or concept within the problem domain which the expert uses in his problem solving process. Using these factors the expert can make judgments about problem cases in the domain in order to make decisions. The factors are described as discriminatory because they are the factors or "data" which make up the patterns which the expert recognises and discriminates on in order to make judgments. It has long been recognised that in addition to using some form of functions e.g. rules, experts recognise particular patterns, which spark off specific actions or decisions [Waterman & Hayes-Roth, 1978]. Identifying the discriminating factors is an attempt to capture the essence of these patterns and to formulate the resulting domain data in some way.

When an expert sees patterns representing different problem cases he is often seeing the same factors occurring, but these factors possess different values in each case. The expert must therefore have some form of "measurement" for each of these factors and they must be able to possess only a certain number of values in order for the expert to meaningfully see patterns, and use the factors within these patterns for discriminating between different problem cases in order to reach different conclusions.

An example of a discriminating factor used by e.g. a recruitment expert may be "ability". The expert will look at this factor when assessing candidates for a course or a job and will place each candidate at some point on a scale for "ability". From the assessment of many such

factors i.e. of the pattern the expert has recognised, a decision can then be reached about the candidate e.g. to accept, re-interview, recommend for another course or job, reject etc.

ii) Relevance for Expert Systems

The relevance of identifying these discriminating factors becomes clearer when we start testing the user understanding of the problem domain. The very fact however, that these factors have been labelled as "discriminating factors" is important. For the purposes of an expert system where we do not want to ask irrelevant questions of the user, one method by which we can cut down the amount of domain data or the size of the "domain space" is to ask questions and thus make inferences about, only factors which play a relevant and specific part in the decision making activity (i.e. those factors which form part of the pattern recognition phase of problem solving and thus differentiate problem cases in the domain).

If for example a doctor takes no notice of the colour of a patients eyes, or of his blood group when making a certain diagnosis, then there is no reason to ask for this information as it will have no effect on the resultant answer to the problem. On the surface however, both pieces of data would seem sensible in the context of diagnosis.

If we are aiming at a system which is usable by the defined users of that system we need to ensure that the factors used within the system are of the correct discriminatory level for those users. Fig. 6.5 shows the relationship between the two hierarchies of users and decisions/actions/factors etc. The discriminatory hierarchy on the right highlights the different levels of understanding and ability to discriminate. Users lower down in the user set hierarchy may need to look at sub-sets of higher level factors in order to discriminate between problem cases, whereas the expert or more experienced user may be able to reach a decision using the higher-level factors.

In this respect, the Fire Risk system is fairly simplistic as there are only three levels i.e. the final rate, the assessment areas and the factors to be considered. Only in a few situations are

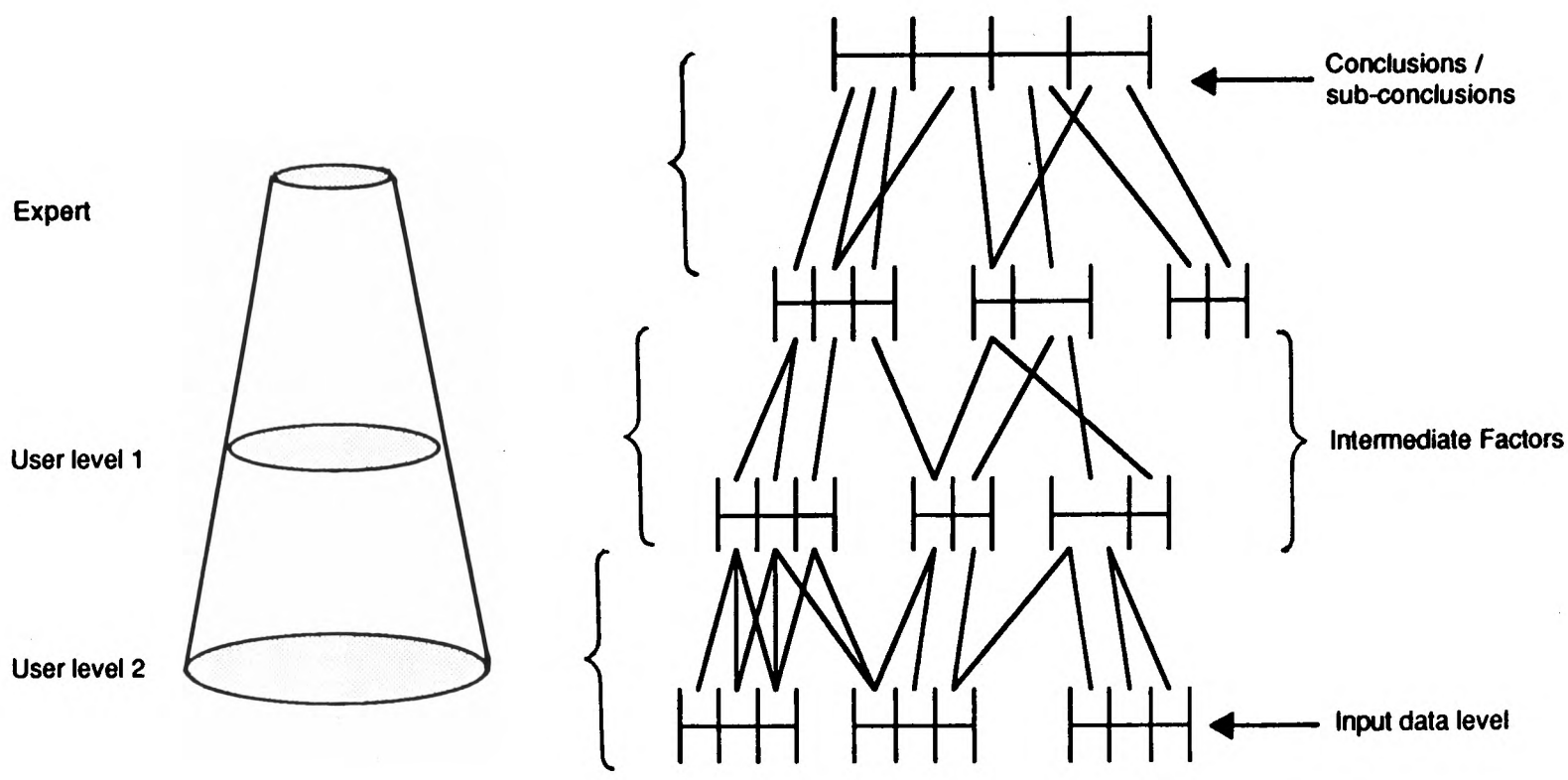


Fig. 6.5 Relationship between user hierarchy and factor / decision hierarchy

factors combined to produce an intermediate result. The tree/network structure for this system is therefore relatively simple.

A head office 'expert' at the top of the hierarchy may be able to judge the risk from the higher level areas e.g. construction, heating etc., by making overall judgments from the survey report.

A branch office professional or novice is more likely to need to look closely at each factor which constitutes these 'area judgments' in order to reach the same decision about the risk.

iii) Identification

The identification of discriminating factors can be carried out in many ways but should be done

i) in the most concise way possible

ii) in a way in which the expert feels most at ease.

The actual method of elicitation used at this point is therefore to a certain degree dependent upon both the nature of the domain, and on the expert and his ability to articulate his knowledge.

For a problem which is fairly routine, and for which test cases exist, example-led interviews can be a good method. In this way, the knowledge engineer only elicits factors which the expert actually uses in solving the particular problem in front of him. The examples used will give direction to the interviews, but the expert will be able to discuss the problem in his own way. The use of further examples in other sessions with the expert will enable the knowledge engineer to both verify and expand upon the set of discriminating factors elicited during the first session.

The above method was used successfully for the initial elicitation of factors for the Fire Risk system. The first few (detailed) knowledge elicitation sessions took the form of the expert

'walking through' a survey report. Whilst talking about the example and how he would reach a decision about a rate to charge for the risk, the structure of the factors used in the decision-making process became clear.

The domain naturally fell into six sub-domains or 'assessment areas', each of which was then studied in detail in order to define the factors being used in reaching a decision about the hazard of each assessment area.

After the first few sessions with the expert, the examples were only referred to for clarity or where confusion arose, or for use as 'analogous descriptions' which may help to clarify points. In the remaining sessions therefore the elicitation of factors was a case of refining the model developed during the first few sessions by means of discussion around the subject.

The knowledge engineer should however keep in mind the fact that it is the user view of the domain which is required for the domain model. The expert must therefore think in terms of these less expert users when identifying and defining discriminating factors. Early definition of the target user set is therefore essential in order to define factors at the correct level.

These elicitation sessions may appear to be functional i.e. the expert may talk about the problem in terms of how he is solving the problem. It is the task of the knowledge engineer however, to abstract from this, the factors which he uses rather than the functions themselves. This too is dependent however upon both the expert and the problem - the expert should not be forced into talking in a specific mode.

In the case of ARIES, the expert was very aware of the need to define factors initially rather than functions, and so the task of developing the 'model' became to a large degree, a joint effort between the expert(s) and knowledge engineers.

Another method could be to try to elicit the factors used in decision-making directly from the expert, without using test cases. The experts mind will not however be focused on a particular problem so he is more likely to overlook factors. There is also a danger, if this approach is used, that the result will be a theoretical rather than a practical model of the

domain. On the other hand, the expert may feel that a limited set of examples could not sufficiently cover the domain, and that the model would be more complete by not using examples.

A method of eliciting factors which would ensure that they were defined at the users level of understanding would be for the knowledge engineer to be present at an expert-user session where the expert explains the problem to the user. In order for the user to understand, the expert must explain the problem using factors at the user level. These factors are then the discriminating factors required for a system which will be usable by this level of user.

Other methods of eliciting factors such as the use of Kelly grids (personal construct psychology) and other psychological techniques are discussed in the section on knowledge elicitation techniques.

The result of eliciting discriminating factors will therefore be a model of the domain knowledge, most probably in the form of a tree or network. The highest level factors, or maybe even decisions will be at the top of this model, with any contributory factors below. Fig. 6.6 shows an example of such a model for one 'assessment area' of the Fire Risk system, 'construction'.

At the point where the expert has reached what he considers to be the lowest level of factors the user needs in order to make decisions, the users must be involved. The expert needs to think about whether the users would be able to assess test cases and discriminate using these lowest level factors in the same way as he does. What does the user need to know i.e.what level of knowledge, experience and understanding is assumed in order to discriminate between the test cases ? As an example, if we imagine an insurance expert rating a building for fire risk, he may be able to classify the combustibility of the walls in terms of e.g. wholly combustible, partly combustible etc. In order for the user to carry out the same task, an understanding of building materials, fire resistance times etc. is assumed. Note that we are not at all concerned with the reasoning process at this point, but purely with the factors which will form the input to the system. If these are at the level of understanding of the

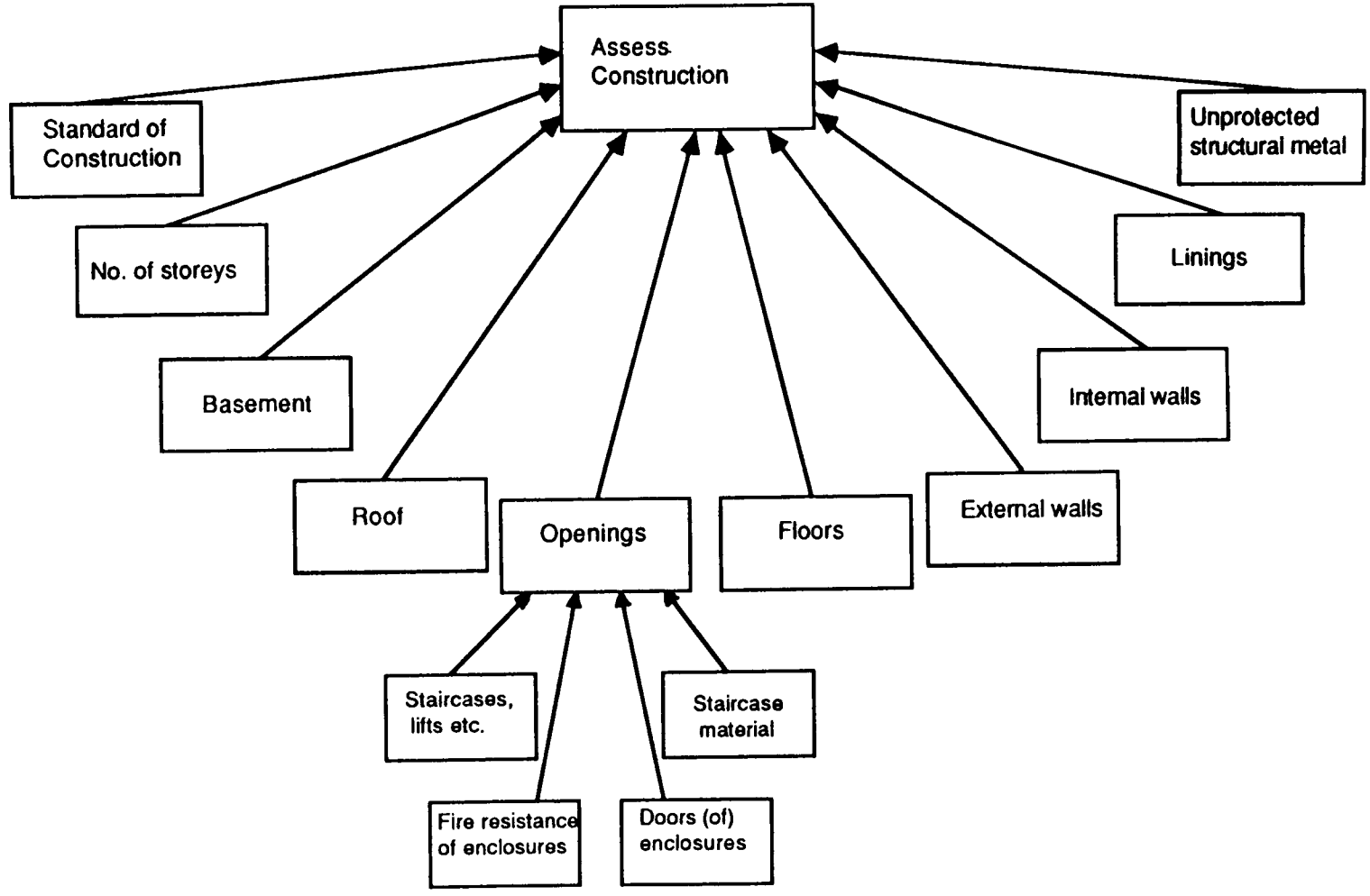


Fig. 6.6 Factorisation of one module from the ARIES Fire Risk system

users then there is a reduced chance of incorrect decisions being made due to questions being answered incorrectly.

If the expert thinks that the assumed level of understanding is too high, i.e.that the users will not be able to make judgments at this level, then the factors should be broken down into lower level or contributory factors, or some 'help' information prepared for the users. Examples of the latter were found during the first testing stage of the fire risk 'paper model' where terms such as 'standard of roof' were not completely clear or understood. In most of these cases however, it was the range of answers provided which caused the mis-understanding not necessarily the question itself. This is discussed further in the next section.

The level of factors which forms the input to the expert system should correspond to the correct level on the hierarchy in fig. 6.5, for the level of user specified.

6.21.3 Limited Data Sets

i) Definition

In order for discriminating factors as defined in the previous section to be of any use in an expert system (i.e.for pattern recognition, differentiating between problem cases etc.), the expert must have classification criteria for these factors. This idea of 'discrete' versus 'continuous' processing is discussed in more detail in chapter 3. Each factor must be divisible into a number of distinct "bands" or classes consisting of either a single (discrete) value, or a series or range of values, where each of these classes is bound by two significant values.

An example of the former (taken from the ARIES Fire Risk Assessment System) may be "incombustibility of external walls". This factor can be represented by a data set consisting of four values, as in fig. 6.7.

Completely	Mostly	Partly	Completely
Incombustible		Combustible	

Fig. 6.7 Limited data set for 'incombustibility of external walls'

< 3	4-12	13-30	31-60	61-100	> 100

Fig. 6.8 Limited data set for 'number of employees'

An example of a data set which consists of a series of ranges of values occurs when considering e.g."number of employees", (fig. 6.8).

Each class is bound by an upper and lower value, decided upon by the expert. In this case, no further explanation of the classes is required, whereas in the previous example, the expert may need to provide some form of communication mechanism i.e.a description of each band or value the factor can take.

ii) Significance of limited data sets

Bayesian updating which relies on the propagation of probabilities, and the use of statistical likelihood factors has been used to represent uncertainty in many expert systems (e.g. Prospector, Expert Edge, R1). The input to these systems takes the form of Yes/No answers together with certainty scores to qualify these answers. There are several problems with this form of input, mostly associated with the robustness and user-independence of the system.

Firstly, the user has to state his certainty of his yes/no answer, and in some cases (e.g. Expert Edge) may have the option of entering his answer to the question on a sliding scale instead of answering with Yes or No. In the latter case the user is faced with two types of uncertainty making the input to the system both ambiguous and confusing. Using a colour scale from red through to green as an example, the user may have to state both where on the red-green scale the colour he has observed lies, and his certainty of the answer he has given.

It is unlikely that a group of users even of the same ability and experience at the given task would give the same values for both of these scales and also that the same user would answer in the same way, on different occasions. We therefore have a problem of user dependence resulting in a system which is highly dependent upon the user input, no matter how much effort has been put into ensuring that the system emulates the expert in terms of functions.

Secondly, once the user has entered these opinions he is then faced with a certainty score reflecting the strength of the decision reached by the system. In his eyes, what is the difference between e.g. 2.8 and 3.2, and would another user interpret these certainty scores in the same way ? There is also therefore a problem in that the output of such systems can be to some degree, meaningless.

Thirdly, it has been suggested, that in many cases, humans do not and in fact cannot, think in terms of numeric certainties, percentages etc. [Fhaner, 1977]. They tend instead to use critical values and cut-off points when describing situations or test cases. If expert systems are to be used by anyone but the experts designing them therefore, it seems more useful and meaningful to use the latter method of entering data, opinions etc. These points are discussed in more detail in chapter 3, in relation to Michie's human window, the issue of user independence, communicability etc.

As is also shown in chapter 3, from a collection of user data (i.e. answers to the same set of questions), classes of answers/opinions can be found, using blockmodelling techniques. As it is the ability of the users to answer questions which form the input to the reasoning process of the system, it is important to represent their understanding in the data set values defined. The blockmodelling method of obtaining classes from user data is obviously one method, another is to involve the users in testing the expert-defined values, as described with reference to the ARIES Fire Risk project in section 6.22.

iii) Identification

For each discriminating factor identified, the knowledge engineer must then elicit more information about that factor. As stated previously, in order to discriminate on these factors, there must be significant values for each one, which the experts and users can identify. It is therefore these values which the knowledge engineer is most interested in eliciting at this stage of domain analysis, in order to complete the domain picture. There are two major sources of this knowledge, as detailed below.

i) The designing expert

The knowledge engineer and designing expert must methodically work through the list of factors, so that the expert can identify the critical values for each factor. These critical values are values of the factor identified, which the expert uses to discriminate between cases in the problem domain. The result of this exercise will be a list of factors divided into bands or classes, representing the domain (fig. 6.9). Each band or class identified may represent either a specific value of a factor, or a range of values. In the latter case, it is the boundary values which are important, as these will be the discriminating values of the factors.

From this exercise we have obtained a picture of the domain, and from this we can see not only what factors the expert uses, but to a certain degree, how he uses them to discriminate, due to the bands or classes identified for each factor in the domain.

ii) User Data

This method of identifying classes can be used either if there are no particular experts in the field, or as a check after eliciting this knowledge from the designing experts. An example of the former may be e.g. medical genetics, where there are many experts, but the field is rapidly developing and new genetical syndromes are constantly being identified (Riaille, 1985).

The method of analysing user data consists of collecting user rankings of test cases, on certain discriminatory factors. All users are asked to discriminate using the same factors, on the same set of test cases, using the same scales for ranking. It is then possible, using blockmodelling techniques, to find the 'natural' classes, on which the users most agree. The technique is discussed in more detail in chapter 3.

6.21.4 Communication Mechanism

The next important stage of domain analysis is to obtain from the designing expert a method of communicating the domain picture to other experts and in particular to users. This involves

Standard of building construction

Essentially Grade I	Essentially Grade II	Neither	Not known

Number of storeys

1	2	3	> 3

Incombustibility of external walls

Completely	Mostly	Partly	Completely
Incombustible		Combustible	

Standard of roof

Satisfactory	Unsatisfactory			
	Slightly	Partly	Mostly	Completely

Combustible floors

Half	> Half

Staircases, lifts and openings

Not aplicable	Fully enclosed	Essentially enclosed	Partially enclosed	Not enclosed

Fig. 6.9 'Domain picture' of all factors and limited data sets for one module of Fire Risk

Fire resistance of enclosures

Essentially fire resisting	Not fire resisting

Doors (of enclosures)

Essentially to fire break standard	Not to fire break standard

Staircase material

Generally incombustible	Combustible

Fire resistance of internal walls

Walls and openings	Just walls	Just openings	Neither	Not applicable

Quality of combustible linings

None	Insignificant amount	Significant amount

Unprotected structural metal

None	Insignificant amount	Significant amount

Fig. 6.9 (cont.) 'Domain picture' of all factors and limited data sets for one module of Fire Risk

describing classes or data set values identified for each factor, as described in the previous section, such that all users concerned can understand the meaning, and use the values reliably for discriminating between test cases, and thus performing the expert task.

If we look at the examples identified by the designing expert, he knew what he meant by the classes he identified, but when users are consulted, they may not be able to discriminate cases into these classes, due to not knowing exactly what each class means. An expert uses years of experience and judgement to perform his task, and will know in his mind what e.g. warm and cold mean to him in terms of temperatures, but will probably not actually think in terms of numeric temperatures. A less experienced expert, or user, may on the other hand, need to have these classes defined in terms of numeric temperatures so that he can reliably place test cases into the correct classes.

There are several ways in which classes can be described by the designing expert, some of which will be discussed below.

i) Textual descriptions

The expert gives a simple explanation of the class without necessarily detailing examples which fall into this class. An example of this type of description is given in fig. 6.9a, where each entry corresponds to a discrete value from the 'heating' assessment area of Fire Risk.

ii) Identify Most Typical Case

If the designing expert is able to identify the most typical test case for each class, the users then have a guide to follow when discriminating between cases. When faced with a test case, and the choice of which class this case should fall into, the user can look at the typical case for each class, and decide which his case is nearest to in character, and place the test case into the appropriate class.

Using the student example of chapter 3, there should be a student in each class who lies in the middle of that class, and can be said to be typical of the standard for that class. Looking

Discrete Value	Help Text
Portable	A portable or transportable heater is a heater which is not securely fixed in a permanent position or which though fixed is not designed for such use.
Central Heating	Central Heating Systems are those where the heated medium is delivered to the space or spaces to be heated via pipes or ducts.
Segregated	The heater is segragated from the rest of the building by separating elements providing not less than two hours fire resistance (including self-closing doors) in accordance with the Committees's "Rules for the Construction of Buildings, Grades 1 & 2", or by being in the open.
Remote	A remote fuel source is where gas or oil is fed to the heater through fixed, rigid pipework, from either an external gas supply or oil tanks as specified in the Committee's Recommendations for Oil Fired Installations.

Fig. 6.9a Help Text for Discrete Values

at the blocked matrix in fig. 3.15 it can be seen that this is in fact true, so other lecturers should be able to use these examples as a guide for placing other students into classes.

Once these most typical classes have been identified, another exercise could be carried out to see what measure of agreement can be attained between the users, when using these guides. This would give us an idea as to whether the users agree with, or can discriminate using this kind of guide.

iii) Identify Boundary Cases

The above method of communication may suffer due to users not knowing how to deal with borderline cases which could fall into either of two adjacent classes.

In such a case, it would be helpful for the user to have an idea of the best and worst, i.e. the borderline cases for each class. These could be identified instead of, or as well as, the most typical cases for each class as described above. The designing expert therefore needs to identify the borderline cases, and describe why he attributes each case to a particular class.

Taking the student example again, the factors which the lecturers were ranking on were ability and motivation. The latter cannot be broken down into numerical type attributes, but a lecturer may well be able to pinpoint certain characteristics for each class, which determine whether a test case falls into that class or the next. For the factor 'ability' however, the lecturers are more likely to be able to identify the borderline cases, or indeed the most typical cases for each class, by means of average marks gained etc.

It is most likely that a combination of the above methods of communication and description of the domain would be used. It is however very important that all users understand the meaning of the classes identified by the designing expert, in as much depth as is necessary for them to use the system and arrive at the same conclusions as would the expert. This is covered further in the next section which is concerned with the testing phase of domain analysis.

6.21.5 Conclusion of the Definition Phase

The principal output from this first phase of domain analysis is a computer-independent domain model. Associated with this model should be some or all of the following :-

- textual details of interviews with the expert
- recordings of interviews
- a formalisation of the top level decisions and (contributory) factors within the problem domain
- definitions of all limited data sets for factors,
- descriptions of the data set values or classes defined by the expert in terms of - text, typical cases, borderline cases
- test cases

The first two of the above would be used for back-up and confirmation of the model. The third is useful when talking to experts about the domain, and shows to some degree, the relationships between the factors and decisions in the domain.

The data set definitions, descriptions and test cases are of most use in the next phase, that of testing the user understanding of the domain model which will form the basis on which the system is developed.

6.22 Testing Phase

The area of testing expert systems is one which is not widely discussed in expert systems literature, except from the functional angle.

Testing the functionality of an expert system does not differ significantly from that of testing conventional data processing systems. Thus modular testing, the use of extreme input data

etc. are all methods which can be used to test whether the system is functioning as specified.

In many expert systems however, there is a facet of testing which does not occur in conventional systems; that of testing the performance of the system in the hands of users of a different level of ability. Like conventional systems, for any set of input the output will always be the same (assuming the functionality of the system is correct), but due to the degree of interpretation required at the input stage, we have to test the consistency i.e.for any one test case, how consistently will the users answer a series of questions posed by the system. Only when consistency (and accuracy) has been reached at this input stage, can we expect such consistency in the output of the expert system.

In addition to consistency there must also be a degree of accuracy in the output i.e.the system must produce accurate (according to the experts) answers consistently, in the hands of the specified set of users.

Prior to testing, these performance measurements must be determined i.e.what degree of consistency and accuracy are required from the system ? From a system where the input is clear-cut, well-defined and not open to any subjective judgement and interpretation it would be reasonable, if not essential, to expect 100% accuracy (and therefore 100% consistency) from all users. Most expert systems however are developed in domains which are towards the other end of the subjectivity scale and thus lower performance criteria would be expected and acceptable e.g.70% accuracy (from users) as compared to the experts.

This form of testing which is not functional, but more concerned with acceptance, "usefulness" and performance needs to be carried out at various stages in the system development cycle. It therefore acts very much in a positive way, giving pointers to areas within the system where performance is below the accepted levels.

If such areas can be highlighted early on in development i.e.before implementation, then further elicitation can be carried out in these areas, and performance improved. As this area

of testing is primarily concerned with the input to the system, it is perfectly possible to carry out tests with the users of the system at the end of the modelling (definition) stage, before any implementation and even before any functional elicitation has taken place.

Although we are talking about the performance of the the system, testing at this stage is concerned only with the performance of one part of the system, the input, not the output or processes. If we come out of the first cycle of analysis, modelling and testing with sufficiently high performance in terms of consistency and accuracy in the hands of the users, then we can enter the second cycle which is concerned with the functionality of the system.

6.22.1 Expert Testing

The designing expert, who has been involved from the beginning of the elicitation process, and who has provided the bulk of the knowledge should be involved in this testing phase, in addition to a group of contributory or moderating experts, whose task it is to see that the expert system does not comply only with one expert in the field.

The main aim of testing at this stage of knowledge elicitation is to check for completeness of the domain model, in terms of factors and their possible values. The domain model should be a complete specification of the domain in terms of the factors recognised by the experts and communicable to users of a lower level of expertise. If the model is complete in this way, the experts should be able to reach decisions using data provided by the model, together with their functional expertise.

In order to carry out the testing phase, a representative set of example cases from the domain needs to be collected, preferably by the expert(s). In addition to "real" examples, random sets of data can also be used for testing, which will represent unseen examples (which may or may not realistically exist in the domain). Both sets of examples will enable the domain model to be tested for completeness whilst the latter set will also highlight illegal combinations of factors or values of these factors. For testing the Fire Risk system, a set of 9 survey reports were used both for testing the experts interpretation of the reports and the

ability of the model to handle this interpretation, and for testing the ability of the users to use the model with consistency and accuracy.

The experts should all be given the same sets of examples, and asked to carry out two tasks for each example :-

i) for all "real" examples, select the correct (i.e.most appropriate) value for each factor, thus creating a domain data model of this example,

ii) for all examples, including those for which domain data models were randomly specified, make a decision about the example, based only on the data represented by this domain data model.

If the experts cannot carry out either of these tasks for any reason, the reasons should be stated. The difficulties envisaged include:

- the factors provided in the domain model do not represent all the factors in the domain, and hence the expert(s) cannot make decisions based only on the domain data models produced at the end of i) above,

- the values given for a factor are not representative of the values which occur in the domain for that factor (e.g. not complete, too numerous, incorrectly defined, etc.),

- explanations are required for factors or values of factors,

- the combinations of values given for a random example may be illegal or impossible.

The latter reason for not completing the testing successfully will result in a void test case, which in itself is of no consequence. It will however provide valuable information for the later stage of knowledge elicitation, that of functional elicitation.

The results of the above tests can then be analysed for consistency between the experts (if this is a required performance measurement). The necessity for this analysis is dependent upon both the number of experts in the field and the expected difference in their treatment of

the same examples from the domain. The consistency when using the domain model should be as good as that expected under normal working conditions.

The major use of the expert test results is however for comparison purposes when user testing is carried out, as described in the next section.

If only one expert has taken part in the tests, then there is no need to carry out tests for consistency, and the set of results (i.e. domain data models) for this expert will be used as the correct results in the user tests. If however more than one expert takes part in the tests, then where disagreement occurs, either this disagreement must be resolved, or a consensus or agreed result must be put forward for the user tests. In either case therefore (i.e. single or multiple expert(s)) the result of expert testing and analysis of results will be one agreed domain data model for each example used in the tests.

6.22.2 User Testing

A representative group of users from the user set as defined in 6.21.1 need to be involved in user testing of the domain model. The aim is to check whether the model has been defined in terms which the users understand and can discriminate on, as can the experts. The outcome of this stage of testing will therefore be an idea of

i) the performance of the domain model in the hands of the users (i.e.the degree to which the users can produce domain data models as did the experts in expert testing),

ii) the areas in the domain model which need further elicitation due to performance in i) above being low.

The process of user testing is the same as that of expert testing, but the analysis of the results is the more important and time consuming part of this stage of testing. The same sets of examples as were used in expert testing must be used so that the results can be realistically compared with those of the experts.

The users are asked to carry out only one of the tests which the experts carried out;

- to produce a domain data model for each of the (real) examples given.

The users will not however be asked to make any decisions based either on these data models, or on random data models. The reason for this is that the emphasis is on testing the performance of the domain model in the users hands by testing the ability of the users to classify examples into the correct categories (values of factors), rather than testing the users ability to make decisions. The degree to which the users make correct classifications indicates the degree to which they understand the model and therefore the input to the potential system. The resulting performance can then be described as the expected performance level of the system in the hands of such users (i.e. of the same level) assuming that the functions and the output of the implemented system are also correct with respect to the expert.

The result of carrying out user tests as above will therefore be a series of domain data models for the same set of examples as the experts. If there is only one expert, then the user results must be compared with this set of results. If however there is more then one expert then the agreed expert result, as described above must be used for comparison purposes.

In comparing the results, the aim is to identify areas of the model where agreement is low, and therefore where further elicitation is required in order to improve the user understanding.

The problems which are likely to occur are:-

i) The factors which the expert has identified are at the wrong level of understanding for the users, and possibly need redefining or breaking down into lower level (sub-)factors,

ii) The users cannot discriminate at the required level i.e.the grain size of the discrete values defined for each factor is incorrect for the users, and needs either increasing or decreasing.

iii) Explanations of factors or data set values are required for clarity. Such explanations would be most useful as a training aid and would not necessarily be used after initial contact with the system.

6.22.3 Analysing the Results

In this section one specific method of analysing the results of expert and user testing (as used on the ARIES Fire Risk system) is discussed although the principles of testing for consistency and accuracy would be the same for any other method chosen.

i) Expert Test Results

The results of expert and user testing can be described as 'domain data models' as defined in 6.22.2. These needed to be analysed for

i) completeness and accuracy of the 'domain model' (IR) i.e. missing or wrongly defined factors, data set values etc.,

ii) consistency i.e. agreement between the experts - a high degree of disagreement between experts in the same domain may indicate a fault in the model.

For ease of analysis, each set of answers on the model was numbered as on the example of fig. 6.10 for the factor 'staircases, lifts and service openings'.

The values selected by each expert for each factor could then be conveniently recorded in table form, and compared. The table of fig. 6.11 shows the method of recording the results which enabled the degree of agreement between the experts to be calculated.

In the table, for the factor 'external walls' the experts A B and C all selected the same value i.e. '1' for test case 1, but selected different values (i.e. '1', '4', and '1' respectively) for test case 4. Agreement between the experts on this factor is calculated as the percentage of test cases for which the experts agree fully. 'Full' agreement in this sense is achieved on the following two conditions :

Not applicable	Fully enclosed	Essentially enclosed	Partially enclosed	Not enclosed
1	2	3	4	5

Fig. 6.10 Numbering limited data sets for testing analysis

FACTOR	Number of fields	Expert	Value selected per test case				% full agreement
			1	2	3	4	
External walls	5	A B C	1 1 1	1 2 1	1 4 1	3 1 2	50%

Numbers refer to numbers associated with data set values for factor 'external walls'

Fig. 6.11 Method of recording testing results for analysis

Module	% agreement between experts
Construction	68
Management	94
Trade Processes	79
Average	80

Fig. 6.12 Expert consistency over three modules for 'paper model' testing

a) The number of fields is <= 3 and all experts select the same field for one test case e.g.

A	2
B	2
C	2

b) The number of fields is > 3 and the difference between the highest and lowest field selected is not greater than 1 for one test case e.g.

A	1	but not	A	1
B	2		B	4
C	1		C	1

In the table of fig. 6.11, full agreement according to rule b) above is reached for test cases 1 and 2, but not for test cases 3 and 4. The percentage full agreement is therefore 50% for the factor 'external walls'.

The results of this analysis were used to highlight areas of the model which needed further elicitation. The rules used to calculate 'full' agreement allowed for small differences in opinion, but at this stage of translating the survey report it was not envisaged that large differences in opinion would result. Where agreement was found to be low therefore, this was taken as an indication that there was some error in the definition of the factor or the discrete value of this factor.

In the case of ARIES, due to the small set of test cases used, this analysis could be carried out by eye to a large degree, but were a larger test set to be used, this process of analysis could be automated. The actual method used, although not rigorous was considered as appropriate at this stage of testing, as a means to highlighting problem areas in the model. An obvious problem with the methods is that when calculating the 'full' agreement it could be argued that if the experts have selected values '3', '1' and '2' as for test case 4 above, there is never a difference of greater than 1 i.e. experts A and B have selected the values either side

of that selected by expert C. However, the fact that the experts have all selected different values is sufficient reason to highlight this as an area of the model for further elicitation.

An example of a factor where low agreement was reached between the experts was 'combustibility of linings' in the 'construction' assessment area. Over the nine test cases, an agreement of 33% was reached, which on further discussion was found to be due to the wording of the question. Consequently the factor was changed to 'quality of combustible linings' and the discrete values altered accordingly. The next stage of testing (in this case implementation testing) showed an increased percentage agreement of 60% for this factor.

In addition to calculating the agreement per factor, the consistency within each assessment area was also calculated as the average of the agreement over all factors in that module. The table in fig. 6.12 shows the consistency at the first stage of testing the paper model of each module.

This table shows the consistency which could be expected at the data input i.e. translation of the survey report stage of the Fire Risk system. The consistency at this stage must have an effect on the overall performance of the system i.e. the higher the consistency of the input the higher the consistency of the output. The processes in the middle can for this purpose be considered a 'black box'.

ii) User Test Results

Two types of analysis were carried out on the results of user testing

i) consistency between users i.e. comparing users results with those of other users,

ii) accuracy i.e. comparing users results with those of the experts.

The method used to record the results and to calculate 'agreement' figures for consistency between users was the same as that for analysing the expert test results, and as before, several problem areas were highlighted from this analysis. The table of fig. 6.13 summarises the consistency i.e. 'user agreement' reached.

Module	% agreement between users
Construction	70
Management	87
Trade Processes	71
Average	76

Fig. 6.13 User agreement over three modules for 'paper model' testing

Test case : 1 Module : Construction						
Factor	Main expert answer	User answers				% agreement
		1	2	3	4	
1	3	2	2	3	3	50
2	4	4	4	4	4	100
3	1	1	1	1	1	100
4	1	1	1	1	1	100
5	2	1	2	1	5	25
6	2	1	1	1	1	0
7	2	2	2	2	2	100
8	2	2	1	1	1	25
9	2	1	1	2	2	50
10	2	2	2	2	2	100
11	2	1	1	2	2	50
12	1	2	1	2	1	50
13	1	1	1	1	1	100
				Average agreement		65

Fig. 6.14 Consistency results betwen 4 users for one module and one test case

These results show a high consistency between the users, but do not tell us whether the users were in agreement with the experts i.e. they may have all agreed on the wrong answer. A more realistic test of the performance of the model in the hands of the users, and of the suitability as regards discriminant ability, is therefore to compare the results of the users with those of the experts.

Where there was disagreement between the experts, the value selected by the main expert for a particular factor was used for the comparison. This was considered reasonable as the main expert had spent the largest amount of time working on the model. The table of fig. 6.14 shows the agreement for one module 'construction' which has thirteen factors, for one test case.

Results from tables such as that discussed above were then collated to find the agreement for each module, over all nine test cases. The averages per module at this stage were 69%, 65% and 51%, i.e. an average of 61%. The experts had envisaged that the users should agree with the experts to at least the 70% level, therefore more elicitation was required to ensure that the level of user understanding could be increased.

As for the analysis of expert test results, the above figures show the consistency and accuracy which could be expected at the input stage of using the developed system. Even if the functions were 100% correct (with respect to the experts method of working), such high accuracy could not be expected due to the inaccuracy at the data input stage.

Although this particular method of analysis has its drawbacks, and is not a statistically proven technique, its use in this case highlighted problem areas, and generally proved that there is a need for this kind of testing to be carried out at the early stages of system development when it is easier to make changes.

6.3 Conclusion

The work carried out during the ARIES Fire Risk Assessment project was invaluable for the continuation of this research in terms of the conclusions which were reached, and also constituted a practical and successful approach to knowledge elicitation.

The principles of domain analysis as defined at the beginning of the chapter were satisfied as detailed below.

To analyse the expert domain thoroughly thus forming a precursor to functional elicitation.

Analysis of the expert domain was carried out in a strictly top-down fashion in the sense that the domain was first split into modules, each of which was subsequently 'factorised' in order to arrive at factor hierarchies of the type in fig. 6.6. Each factor was then in turn analysed in terms of the values which it could take (e.g. fig. 6.9) - these being based on the linguistic terms which were used naturally in the domain of fire risk assessment.

The ease of functional elicitation from this starting point can be likened to any process where the input and output is known. The expert was asked to enumerate the combinations of values of factors which lead to certain decisions for each module. This formed the lowest level of reasoning i.e. reaching a decision regarding a specific module of the system such as building construction.

The functional elicitation at the next level was then a repetition in terms of the process applied in that the output decisions from the lowest level of reasoning formed the input 'data' to the higher level of reasoning.

In addition to easing the process of functional elicitation, it is clear that such an approach also ensures both consistency and conciseness in the resulting knowledge base. If, for example, we imagine a system with only one level of factors and one level of decisions, the reasoning process can only have as input, factor values, and as output, decision values.

To involve users in the development process with the view to reducing user dependency.

Users were involved in all stages of development of the knowledge base, but most importantly, during the domain analysis. As soon as a 'paper model' of the domain had been developed, the users tested this model. They did not have to know the expert reasoning process in order to carry out this testing, and were only presented with the same information as would a system developed at that stage i.e. the allowed values for each question.

The results of these tests and the conclusions of the analysis were described fully in sections 6.22.2 and 6.22.3. The main advantage of carrying out these tests was that any problems with user dependency could be identified and solved before implementation.

To take a data oriented approach to the problem of elicitation and subsequent formalisation of the knowledge.

As outlined at the beginning of this thesis, one of the aims of this research is to investigate the degree to which methods of systems analysis can be applied to knowledge elicitation and analysis. Due to the success of data oriented methods for systems analysis, it is therefore reasonable to take similar approaches to knowledge elicitation. Methods from systems analysis were not strictly applied during this project, however, experiences of taking a data oriented approach as described above did lead to the following conclusion:

A data oriented approach is useful and successful for knowledge elicitation, but a more structured and rigorous approach is required which provides standard graphical techniques as opposed to informal 'ad hoc' techniques.

The work described in chapter 4 of this thesis takes up this issue by the application of ER modelling to knowledge engineering.

To restrict the input data of the system to limited data sets.

Several advantages of taking such an approach are immediately obvious i.e. that the user interface is significantly simplified; that testing of the 'paper model' or system for consistency

and accuracy (i.e user independence issues) of input is made possible; and that functional elicitation is made more straightforward, as discussed above.

The application of this constraint and the advantages as outlined above led to the investigation of the notion of explicitness of knowledge bases and the need to first define an explicitness boundary as discussed in chapter 3.

CHAPTER 7 CONCLUSIONS AND PROGRESS AGAINST OBJECTIVES

Conclusions reached about specific parts of the research carried out for this thesis e.g. the significance of the explicitness boundary, and the use of specific techniques within the knowledge elicitation process, are drawn at the end of the corresponding chapters. The aims of this concluding chapter are twofold. The first is to outline the approach to knowledge elicitation, highlighting the key features, and the second is to assess the success of the research against the objectives at the start.

7.1 Structuring the Task of Knowledge Elicitation

Knowledge elicitation can be thought of as consisting of two main tasks - gaining knowledge from human experts, and subsequently analysing this knowledge. The process is an iterative one, which is equivalent in terms of its position in the development life cycle of a computer system, as the analysis phase in the development of a traditional data processing system.

The tools and techniques discussed in this thesis are most directly concerned with the second of the above two tasks i.e. analysing the knowledge. However, due to the iterative nature of the whole process, the analysis will to a large degree, drive the elicitation sessions.

One of the first sub-tasks of knowledge elicitation is to place a high level structure on the experts knowledge. It is at this stage that data flow diagrams can be useful, both for partitioning the problem, as described in chapter 3, and also for putting it into context with its environment [Thomas, 1988].

Once a high level structure has been achieved, then detailed elicitation is required to 'flesh out' each of the resulting 'portions' of knowledge. It is at this point that the notion of an explicitness boundary first becomes important, as does the involvement of potential users of the expert system.

The explicitness boundary is in practice made up of a series of limited data sets which define the input to and output from the processes which will constitute the knowledge base.

The practical importance of defining such a boundary between quantitative and qualitative data and processes, which is discussed in detail in chapter 3, can be summarised as follows:

i) Any task, whether expert or not, can be considered as having three elements - input, output and processing. The definition of the first and last of these provides considerable help with eliciting the third.

ii) The limited data sets form the building blocks of the experts knowledge, just as does data in a conventional data processing problem. This allows for the use of traditional analysis and modelling techniques i.e. entity modelling and entity life cycle analysis for modelling the expert knowledge.

iii) The limited data sets which are defined at the boundary form the interface *via* which the user will communicate with the expert system. The users can therefore be involved in evaluating and testing this interface before the system is developed.

The process of detailed elicitation and analysis of the experts knowledge is thus concerned with the production of a logical model in graphical form.

The resulting model of the expert knowledge has then to be converted into a physical knowledge base. Although not the central theme of this thesis, it is shown in chapter 5 that the modelling techniques proposed do in fact map onto common knowledge representation schemes, namely production rules and frame systems.

7.2 Progress

In chapter 1, the aim of this thesis was stated broadly, as

"to develop a method of knowledge elicitation".

In addition, motivation for the research was discussed, and a number of problems inherent in the knowledge elicitation process were outlined. In the first part of this chapter an assessment is made of the success of the research in terms of achieving the above aim, satisfying the motivations outlined in 1.1, and overcoming the problems of knowledge elicitation as stated in 1.22.

7.21 Development of a Method for Knowledge Elicitation

The main question to be answered here, is whether or not the set of techniques discussed in the preceding chapters constitute a method. This depends largely on the view taken of the role of a method in developing systems. The view taken in undertaking this research is that a method should encourage a structured and rigorous approach to the process of knowledge elicitation, but at the same time, allow flexibility.

The set of techniques prescribed are compatible with each other, thus allowing continuity from one stage of elicitation to another and therefore providing consistency, but can each be used independently, thus satisfying the need for flexibility.

In section 1.3, a structured method is defined, in the context of systems analysis, as providing graphical tools for communication, forcing a logical / physical design split, and encouraging a high degree of user involvement during analysis for logical design. The techniques proposed in chapters 3 and 4 provide graphical tools in the form of data flow diagrams, ER models and ELC diagrams. The resulting 'intermediate' representation of the expert knowledge serves both as a communication medium, and also as a basis for logical design.

The issue of user involvement is discussed mostly with regard to user independence of the resulting systems - the proposed method encourages the involvement of users of the expert system in testing the limited data sets which will form part of the user interface.

7.22 Motivations

As outlined in chapter 2, a number of techniques exist to aid specific tasks within the knowledge elicitation process. There are very few methods, however, which cover the process from start to finish, and which provide the advantages of a structured method as discussed above.

The proposed set of techniques provide a framework within which to work, and result in a formalism which can be translated into a physical design, but which remains independent of that design. Techniques such as memory probing, teachback interviewing etc., as described in chapter 2 could fit into this framework providing specific ways of eliciting, say, concepts (entity types) or events which trigger changes in the status of the knowledge.

Due to the emphasis on analysing the expert domain from a data-oriented point of view, the method proposed allows scope for the involvement of users who may be of a different level of ability to the experts. Whereas it is necessary to elicit the knowledge initially from the expert, it is critical to test the limited data sets which will form (part of) the user interface for their suitability for and understandability by, the designated users. Although it has been stated (Hart, 1986) that knowledge elicitation is a more complex task than systems analysis, the application of techniques from systems analysis in this thesis has shown that, in principle, the same methods can be applied. A data oriented approach is certainly advantageous, not only in terms of the resulting consistency and conciseness of the resulting knowledge base, but also due to the help it provides in eliciting the functional knowledge associated with the task.

A number of extensions to the techniques of entity relationship and data flow modelling have been suggested which make the techniques more suited to knowledge elicitation. This type of enhancement is necessary partly due to basic differences between conventional data processing and human data processing, (e.g. the idea of sequential processing), and also due to the complexities of the knowledge to be modelled.

7.23 Problems of Knowledge Elicitation

7.23.1 Form of the Knowledge

The two characteristics of human knowledge which make knowledge elicitation most difficult i.e. subjectivity and uncertainty, are also the most difficult to overcome with any method. However, using a specific set of techniques as proposed, gives the knowledge engineer or analyst a starting point, and tools to aid them in analysing the knowledge. For example, in order to draw a set of data flow diagrams, the type of questions which must be put to the expert are almost pre-defined. By having a definite goal to achieve from elicitation, the task of the knowledge engineer is simplified, allowing more time to concentrate on the knowledge to be formalised as opposed to the method of formalisation.

Similarly, the ability of the expert to articulate his knowledge cannot be helped directly by the use of a method, but indirectly by increased focussing due to the defined end products. These end products also form a communication medium between the expert and knowledge engineer thus helping both progression of the elicitation itself, and the ability of the expert to articulate.

Lastly, the techniques proposed do take into account different types of knowledge - data flow diagrams deal with two aspects of knowledge processing - the 'data elements' passed between processes, and the control knowledge required. Entity relationship models help to formalise the concepts and relations which form part of the expert knowledge, and entity life cycle diagrams (together with the results of attribute analysis) deal with the control aspects of processing i.e. the functional elicitation.

7.23.2 Formalisation and Representation Techniques

As discussed above, the techniques proposed from this research provide a method of representing the expert knowledge in an intermediate form. The resulting models may not be the only form of representation used, but they do force a logical / physical split, necessary to ensure a rigorous approach to the design of any computer system.

7.23.3 Usability of the Resulting System

In chapter 3, the issue of user independence of the resulting system was discussed, and in chapter 6, the need to test the user interface before implementation. These are both crucial to the success of the system in terms of its usability by the target users. The sooner the users are involved, the more likely is the system to be understandable to them and therefore usable by them.

The method of testing described in chapter 6 is not definitive, but serves to illustrate the usefulness of testing at an early stage of system development i.e. during knowledge elicitation. In the context of the ARIES project, the testing procedure was found to be useful both for finding areas of the knowledge which were not clearly expressed and areas for which the users did not have a sufficient level of understanding as compared with the experts.

References

Addis T.R. (1985), Designing Knowledge Based Systems, Kogan Page, London.

Arabie P, Boorman S.A. and Levitt P.R. (1978), Constructing Blockmodels: How and Why, Journal of Mathematical Psychology Vol 17, No. 1.

Bachman C.W. (1969), Data Structure Diagrams, Data Base 1, 2, pp4-10.

Berry D.C. and Broadbent D.E. (1984), On the relationship between task performance and associated verbalisable knowledge, Quarterly Journal of Experimental Psychology Vol. 36a, 2, pp209-231.

Black W.J. (1986), Intelligent Knowledge Based Systems, Van Nostrand Reinhold (UK).

Boose J.H. (1985), A knowledge acquisition program for expert systems based on personal construct psychology, International Journal of Man Machine Studies 23, pp495-525.

Boose J.H. (1986), Expertise Transfer for Expert System Design, Elsevier Science Publishers, Amsterdam.

Braune R. and Foshay W.R. (1983), Towards a practical model of cognitive/information processing task analysis and schema acquisition for complex problem solving situations, Instructional Science 12, pp121-145.

Breuker J.A. and Weilinga B.J. (1983), Analysis Techniques for Knowledge Based Systems, Reports 1.1, 1.2, Esprit Project 12.

Broughton R.K. and Cox P.R. (1983), Micro-Expert Users Manual.

Buchanan B.G. and Shortliffe E.H. (1985), (eds.), Rule Based Expert Systems, The MYCIN Experiments at Stanford, Addison-Wesley.

Buchanan B.G. and Shortliffe E.H. (1985a), Knowledge Engineering, (In, Buchanan B.G. and Shortliffe E.H. (1985), pp149-158), Addison-Wesley.

Buchanan B.G. and Shortliffe E.H. (1985b), Human Engineering of Medical Expert Systems, (In, Buchanan B.G. and Shortliffe E.H. (1985), pp599-612), Addison-Wesley.

Bunt H. (1985), The formal representation of (quasi-)continuous concepts, (In, Hobbs J.R. and Moore R.C. (eds.), Formal Theories of the Commonsense World), Ablex Publishing Corporation.

Butler A. (1987), Expert Systems for Insurance and Investment, (In, Proceedings of the Third International Expert Systems Conference), Learned Information.

Chen P.S. (1976), The Entity-Relationship Model - towards a unified view of data, ACM Transactions On Database Systems 1,1, pp9-36.

Chi M.T.H., Feltovich P.J. and Glaser R. (1981), Categorization and Represenation of Physics Problems by Experts and Novices, Cognitive Science 5, pp121-152.

Crittenden J. (1988), Escar: An Expert System for the control of aluminium reduction, (In, Giorgi B (ed), Personal Computers for Industrial Control), Butterworths.

Date C.J. (1981), An Introduction to Database Systems, Addison-Wesley.

De Marco T. (1978), Structured Analysis and System Specification, Yourdon.

Duda R., Gaschnig J. and Hart P. (1979), Model Design in the PROSPECTOR consultant system for mineral exploration, (In, Michie (ed), Expert Systems in the Micro-Electronic Age), Edinburgh Press.

Expert Systems International (1985), ESP Frame Engine User Guide, Expert Systems International Ltd.

Fhaner S. (1977), Subjective Probability and everyday life, Scandinavian Journal of Psychology, 18, pp81-84.

Gaines B.R. and Shaw M.L.G. (1986), Induction of inference rules for expert systems, Fuzzy Sets and Systems 18(3).

Gammack J.G. and Young R.M. (1984), Psychological Techniques for eliciting expert knowledge, (In, Bramer M. (ed.) Research and Development in Expert Systems), Cambridge University Press, pp 105-112.

Gane C. and Sarson T. (1977), Structured Systems Analysis: Tools and Techniques, McDonnel Douglas.

Gane C. and Sarson T. (1980), Structured Methodology. What have we learned?, Computerworld (extra), 16, 38, pp52-57.

Gilbert T.F. (1961), The Technology of Education, Journal of Mathetics, Vols 1 & 2.

Graham I. and Jones P. (1988), Expert Systems - Knowledge, Uncertainty and Decision, Chapman and Hall.

Greenacre M.J. (1984), Theory and Applications of Correspondence Analysis, Academic Press, London.

Grover M.D. (1983), A Pragmatic Knowledge Acquisition Methodology, Proceedings - AI (8th), Kalsruhe, Germany, Vol.1, pp436-438.

Hart A. (1984), Experience in the use of an inductive system in knowledge engineering, (In Bramer M. (ed.) Research and Development in Expert Systems), Cambridge University Press.

Hart A. (1986), Knowledge Acquisition for Expert Systems, Kogan Page.

Hayes-Roth F., Waterman D. and Lenat D. (1983), Building Expert Systems, Addison Wesley.

Howe D.R. (1983), Data Analysis for Database Design, Arnold.

Johnson L. (1985), The need for competence models in the design of expert consultant systems, International Journal of Research and Information Science, Vol.1, pp23-36.

Johnson L. and Johnson N. (1985), Knowledge Elicitation involving teachback interviewing, man Computer Studies Group, Brunel University.

Johnson P., Diaper D. and Long J. (1984), Tasks, skills and knowledge: Task analysis for knowledge based descriptions, Manuscript, Ergonomics Unit, University College London.

Logica (1986), ARIES Reports - Stages 3, 4 and 5, Logica Financial Systems Ltd.

Mealy G.H. (1967), Another look at data, Proceedings - Fall Joint Conputer Conference, pp525-534.

Michie D. (1982a), The State of the Art in Machine Learning, (In, Michie D. (ed.) Introductory Readings in Expert Systems), Gordon & Breach.

Michie D. (1982b), Game playing and the human window, Computer Journal 25(1), pp105-113.

Minsky M. (1975), A framework for representing knowledge, (In, Winston P. (ed.), The Psychology of Computer Vision), McGraw Hill.

Mumford E. (1978), Participative Systems Design, Computer Weekly, Nov.

Mumford E. (1985), Expert Systems and Organizational change, Manchester Business School.

Myers C.D., Fox J., Pegram S.M. and Greaves M.F. (1983), Knowledge Acquisition for Expert Systems: experience using Emycin for leukemia diagnosis, (In, Proceedings - BCS Expert Systems Conference, 1983).

Newell A. and Simon H.A. (1972), Human Problem Solving, Prentice-Hall.

Peters L.J. (1981), Software Design: Methods and Techniques, Yourdon.

Quine W.O. (1961), From a Logical Point of View, Harper and Row.

Quinlan J.R. (1979), Discovering Rules by Induction from Large Collections of Examples, (In Michie D. (ed.) Expert Systems in the Micro-Electronic Age), Edinburgh University Press.

Quinlan J.R. (1982), Semi-autonomous acquisition of pattern-based knowledge, (In Michie D. (ed.) Introductory Readings in Expert Systems), Gordon & Breach.

Rock-Evans R. (1981), Data Analysis, IPC Press.

Romiszowski A.J. (1981), Designing Instructional Systems (Decision Making in Course Planning and Curriculum Design), Kogan Page, London.

Shaw M.L.G. (1980), On Becoming A Personal Scientist, Academic Press.

Shaw M.L.G. (1981), (ed.), Recent Advances in Personal Construct Technology, Academic Press.

Suwa M., Scott A.C. and Shortliffe E.H. (1985), Completeness and Consistency in a Rule-Based System, (In Buchanan and Shortliffe (eds.) Rule Based Expert Systems, The MYCIN Experiments at Stanford), Addison-Wesley.

Teory T.J., Yang D. and Fry J. (1986), A Logical Methodology for Relational Databases using the Extended Entity-Relationship Model, Computing Surveys, 18, 2.

Thomas M. (1988), Structured techniques for the development of expert systems, (In, Moralee S. (ed.) Research and development in expert systems IV), Cambridge University Press.

Tsichritzis D. and Lochovsky F. (1982), Data Models, Prentice-Hall.

Tulving E. (1972), Episodic and Semantic Memory, (In Tulving E. and Donaldson (eds.) Organisation of Memory).

Waterman D.A. and Hayes-Roth F. (1978), (eds.), Pattern-directed inference systems, Academic Press, New York.

Welbank M (1983), A Review of Knowledge Acquisition Techniques for Expert Systems, Martlesham Consultancy Services, Ipswich.

Yourdon E. and Constantine L. (1979), Structured Design, Prentice Hall.

Zimmerman R. (1983), Phases, Methods and Tools - A Triad of Systems Development, (In Entity-Relationship Approach to Software Engineering), Elsevier Science Publishers, North Holland.