Task Models for Knowledge Elicitation

Michael Wilson
Rutherford Appleton Laboratory
Chilton, Oxon, U.K.

ABSTRACT

Knowledge elicitation cannot be lead solely by the expert. Elicited knowledge must be analysed and represented by the knowledge engineer. When a knowledge engineering project is started some framework must exist of what is likely to be encountered and the representation to be used for it. Given our present knowledge of the psychology of expertise the top-down influence on knowledge elicitation must be derived from previous analyses of the tasks likely to be performed by the expert.

As knowledge engineers become more experienced they will build up there own library of previously analysed tasks. Before then, they must rely on general task analyses and their model libraries. This chapter reviews developments in task models which can be used to guide decisions during knowledge acquisition.

Introduction

When we start knowledge acquisition we have a series of techniques at our disposal. We are aware that we should not alienate the expert, or appear as though we are going to replace him with a computer. A general methodology suggests that we gain background information and then approach an accessible, willing expert. But what do we actually do when we are faced with the background information or expert; which technique do we use; if we default to an interview, what do we ask about; when we analyse the tapes and data, what do we select; which terms do we use in sorting tasks ?

The earlier chapters in this book have described the battery of techniques available to the knowledge engineer when faced with an expert. But the discipline of knowledge engineering has very little to say about which technique to use when, and what to look for. For example, if you are to develop a KBS for choosing periodicals from a news-stand, you may look around the news-stand and interview the stall-holder. Your first reaction may be to classify the periodicals. For this, various techniques have been described in earlier chapters. But is this the best thing to do? If you choose to do this through an interview, which questions do you ask after general ones designed to gain a rapport with the expert? To answer these questions you must have a pre-conception of what is required out of the knowledge acquisition process.

This chapter will describe attempts to develop models of what pre-conceptions of the output from knowledge acquisition are useful and how they can be used to motivate the decisions made by the knowledge engineer during knowledge acquisition. If knowledge engineering were a better developed discipline it might to be possible to provide a cookbook which listed problem types, the techniques that should be used to elicit knowledge for them, and the order in which they should be used. Unfortunately, it is not that well developed so this chapter will describe several research efforts to reach this state in order that individual knowledge engineers can develop their own cookbooks as their experience increases, without making the mistakes which others have made before them.

In a recent article Hoffman (1987) criticises many articles whose titles indicate that they concern knowledge acquisition but actually contain discussions of abstract inference strategies and representations rather than methods to extract knowledge from experts. This paper is consistent with that body of literature he criticises. This is because a knowledge of abstract inference strategies and representations is necessary in order to make the decisions about which technique to use and which questions to ask during knowledge acquisition.

What should guide the knowledge engineer ?

The pre-conceptions a knowledge engineer brings to a knowledge aquisition task may have been stated, or not. It is always good experimental practice to make all pre-conceptions overt and to structure them so that it is clear where they are influencing decisions, otherwise decisions will be made which cannot be justified, and may be contradictory. If the pre-conceptions of what a knowledge engineer is looking for are going to be stated, what will they be?

One argument suggests that when knowledge is expected to be verbalisable, the expert should lead all interviews and all the knowledge engineer should do is record it and repeat it to the expert to make sure it is understood the same way by both parties. This will intimidate the expert least and will prevent the knowledge engineer's pre-conceptions from influencing the acquisition process. There are two problems with this argument on its own. Firstly, the documentation used for background study during knowledge acquisition and experts themselves can provide vast amounts of knowledge which is irrelevant to the development of a knowledge based system. We want to access knowledge which is salient to the purpose of the KBS. Therefore we do not want either too much or too little knowledge. The time and effort of both the knowledge engineer and expert are valuable and this irrelevant material should not be brought into knowledge acquisition, as it can be if the knowledge engineer does not provide some direction. Secondly, it is obvious that material gained from the expert must be analysed and in this analysis, decisions will be made on basis of the knowledge engineer's pre-conceptions. Consequently, the argument can only apply to interview sessions and not to the whole knowledge acquisition process. Bottom up methods must be used during an interview, but feedback or teachback are local techniques. Similarly, asking whether there are any more items at the same level of a hierarchy as an item identified or requesting the name of the class these items fall into are simple questions. There must be some model or set of pre-conceptions to provide

more complex questions.

A second argument states that knowledge acquisition and KBS development are processes of mapping one system onto another. They try to map the parts of a system which includes a human who is expert at a task onto a computer program that will perform a task. If one starts this process with a pre-conceived notion that the computer program will be built around a simple shell, then techniques can be applied to extract knowledge consistent with the inference mechanism and representations employed by that simple shell (e.g. facts, and rules, driven by an inference procedure using backward chaining search and Bayesian logic). Most texts that describe interviewing techniques for other purposes that knowledge acquisition expect the interviewer to have a prepared list of questions. The required representation can be used to motivate these questions. Unfortunately, although simple shells provide an introduction to expert system development, they have proven inadequate for developing large systems, and systems which try to address problem types for which they were not developed. Consequently, trying to bend the knowledge in interview transcripts and documents to fit an implementation representation is unlikely to result in a record of the expert's knowledge.

A third argument states that in developing systems the knowledge engineer's initial view of the target system must be broader than the facilities offered by such shells, or tool-kits. The knowledge engineer should aim at producing a representation of the knowledge intermediate between the expert's and that of the target system (e.g. a paper model) which is not confounded by the trade-offs necessary to implement it, but more purely represents the knowledge of the expert. In this situation the choice of representation cannot be guided by the target implementation, but other criteria must be used. Neither can the elicitation techniques be chosen to produce a particular machine representation. Ultimately, the choice of representation and elicitation technique should be based on a theory of the psychology of expertise. To this end, much research in knowledge acquisition is attempting to determine the appropriate techniques to be used to acquire specified types of knowledge. Until this work has progressed further we require an intermediate method.

Knowledge Types, Representations and Elicitation Techniques

Attempts to specify knowledge types and the techniques suitable for extracting them are becoming increasingly common in the knowledge acquisition literature (e.g. Gammack and Young, 1985; Bainbridge, 1986; Slater, 1987; Welbank, 1987a, 1987b, 1987c). Matrixes of the suitability of techniques for knowledge types can be derived from experimental study based on a thorough analysis of tasks and the knowledge they draw upon. Such studies are in progress although the complete range of testing required is not complete (see Schweickert, *et al* 1987). However, they can also be derived by surveying the research literature for previous analysis and combining the different authors' descriptions. This is a laborious task which provides a starting point for more detailed analysis and I am not criticising it *per se.* However, it often yields a set of terms derived from different sources which are hard to understand as a coherent whole. An example after Welbank (1987c) is shown in Table 1 which lists a series of techniques, and the knowledge types for which they have been shown to be suitable or inappropriate by various authors. A consistent model is required onto which these terms may be mapped for their relationships to be comprehensible.

Table 1: Types of Knowledge with appropriate Acquisition Methods (after Welbank 1987c). Key: Y - suitable; N - not suitable; ? - difficult, should be used with care.

There are various ways of deriving knowledge types to look for, some of which have been used to derive the terms used in this table.

1) The implementation language - The outcome of the KBS development process will be a running system coded in an implementation language. Therefore, since the information in the knowledge sources must be recoded into this implementation language, the classes of knowledge available in it can be used to select the appropriate technique. Unless the classes of knowledge used in the acquisition can be mapped onto the implementation there is no point in using them, so they may as well be used at the outset to avoid unimplementability. In Table 1, examples of terms from this source are facts, rules, weight of evidence, and context of rules.

2) A formal grammar - There has been research in linguistics and human sciences for many years which has produced a wide range of formal grammars for representing the information which may exist in a population to support its culture and language. Since these are the product of many years work the classes they utilise can be used to classes of knowledge. This approach would be better than using the implementation language as a constraint since that describes computer scientists' views of the "correct" way to classify information. At least formal grammars have been developed for the human sciences and will be more likely to apply to the humans involved in the knowledge acquisition without being polluted by the arbitrary constraints of current computers.

3) Psychological Theory - Psychologists have complex human information processing models of human reasoning which provide categories for knowledge. These should be used to categorise the information and would be better than formal grammars since they will allow the information used by an individual during performance to be captured rather than the "competence" of a population. In Table 1, an example of a term from this source is conceptual structure.

4) Tasks Previously Analysed -

Once a series of tasks have been analysed there is already a description of them and it is not necessary to resort to psychological theory every time. The analyses of previous tasks may require slight modification, but they apply to the individual and are not constrained by the machine implementation. It would be good to base the classification on psychological theory since the analysis of the expert's task should be in terms of the objects and actions that exist in the expert's world, which includes the expert's mental world. However, psychological theory is not yet advanced sufficiently to allow that, so analyses based on descriptions of tasks using psychological constraints where possible are the best that can be achieved. In Table 1, examples of terms from this source include expert's strategy and system's strategy.

Task Analysis

Task analysis is a technique where a task is divided into into pre-specified units. These often take the form of sub-tasks where the actions taken and the classes of objects they affect are specified. The pre-specified sub-tasks, actions and object classes are usually derived from previous analyses, using the available theoretical base. Methods of task analysis have a long history of use in the social/behavioural sciences for breaking tasks into behaviourally (e.g. Miller, 1962) or cognitively (e.g. Rasmussen, 1986; Barnard, 1987) salient units.

The suggestion that task analysis should be used for knowledge acquisition requires the development of a suitable taxonomy of tasks, and for each task, the units into which it can be divided. In general, the analysis method would enable known tasks to be identified in problems facing the knowledge engineer. These would provide initial models of the knowledge types which should be found, and the inference structure in which they would be used. Since the knowledge types would be known, the appropriate techniques could be selected for knowledge elicitation, and appropriate questions asked to determine the problem solving method and the completeness of the knowledge elicited.

A variety of knowledge acquisition regimes have been presented in the literature (e.g. Frieling et al, 1985) although most are still informal and unproven. In a generalised regime of knowledge acquisition, such a task analysis technique would take place early in the method to direct later domain knowledge gathering:

1) Initial Problem assessment to determine if KBS offers a cost effective solution ?

2) Initial study of documents and first interviews.

3) Task Analysis

4) Full Interviews/Performance Protocols/Indirect Measures/Simulation

5) Develop the Intermediate Representation.

6) Re-formulate conceptual structure of the intermediate representation for a particular tool.

After the third stage, further sub-tasks may be identified by task analysis.

To develop an appropriate task analysis it is necessary to specify what a task or problem type is for the purpose of knowledge acquisition and to generate a taxonomy of them to allow them to be identified. "Attempts to make knowledge engineering a systematic discipline often begin with a listing of problem types" (Clancey, 1985:313). Indeed, as Clancey suggests, this kind of analysis is prone to category errors. I will continue with his example of a naive list of "problems" or task types which includes "design", "constraint satisfaction" and "model-based reasoning" which can also be classed as a type of task, an inference method and a kind of knowledge. He neatly exemplifies the combination of these in a single task as a VLSI chip design problem using constraint satisfaction to reason about models of circuit components. To avoid this kind of confusion it is necessary to develop a systematic description of task types at the same level of description, and then specify the classes of knowledge they incorporate before deciding which techniques are suitable for eliciting particular knowledge types. In knowledge acquisition and KBS research the appropriate task type is often termed the "Generic Task".

The Generic Tasks for Knowledge Acquisition

The first problem is one of describing the "appropriate" level of task. For example, in the task of finding the number of a chemist in a yellow pages directory, one analysis might say that the task is "to find the number" and that it has an input which is an ill defined description of the role of the agent whose number is being searched for, and the output is the number itself; consequently, the search is a unitary operation: that is, a single task. A second description might say that the task can be further sub-divided into taking the description and performing an action of comparing it with the headings in a directory until one of them is an acceptable match, then looking down the list of entries under this heading to find one which matches secondary criteria (such as geographical procsimity to the searcher) and the result of this match is the output. A third description may subdivide the search task even further and describe the processes which take place during the match in great detail. Such an analysis would describe the perception of the characters on the page and the formulation of words from them, then the matching of these words with memory. This level of description is more detailed but also has a has a superficial appearance of being more psychologically salient since it incorporates stores such as memory and operations such as human perception. How can we determine in this hierarchy of possible descriptions of a task which is the one that is the most appropriate for the building of KBS's? The answer must be influenced not only by a desire to capture the psychological reality of the description, but also by the computational and functional constraints imposed by the purpose of the desired KBS itself.

The first confusion which arises at this point is in determining which task is being described. The expert performs a task in his natural environment. The KBS when built will also interact with its user to perform a task in a new environment. The expert may be a consultant neurologist working in a hospital diagnosing patients who have been referred to him. In contrast the user may be a general practitioner diagnosing patients who come to him with symptoms. Two very different tasks are being performed here, and the task the KBS is performing is only a part of the second task. What is required is an analysis which will map the task performed by the expert (e.g. the neurologist) into the task performed by the KBS (e.g. the practitioners tool). It is accepted that there are various different roles such a tool can play with respect to the user: advisor, imperative guru, criticising colleauge or teacher. Each of these different roles will require different knowledge to support them, and consequently different knowledge to be aquired by the knowledge engineer. For example, a system in the role of an imperative guru does not require a deep model of the problem to use as the basis for persuasive explanation whereas a critiquing system does. It would be preferable if the task analysis technique would account for the final role of the system in its environment. However, before we analyse the task the system is to perform we must look at the expert's task, although it may

be different.

For task analyses in other areas of research the problem of selecting the appropriate "grain" of task has been resolved by performing analyses at levels of description which appear to possess too little detail, intuitively the right level of detail and too little detail. The results of these different levels of analysis of example tasks have then been instanciated in the available technology and the consequences of using that technology on them have been compared with a view of what the desired output should be. This form of sensitivity analysis has shown in previous cases what the appropriate practical level of description is for the available technology (e.g. Card, Moran and Newell, 1983). We can attempt to apply the same method to the technology of KBS's. What we are looking for is the level of description of a task which is appropriate for both the technology available to implement it and the human expert who has to describe it to the knowledge engineer. Therefore we must look at the analyses of tasks previously performed to derive a taxonomy of generic tasks at the appropriate level for knowledge acquisition.

One source of previously analysed tasks that can be used are previously developed KBS. To illustrate the grain of tasks such an analysis will provide (after Clancey, 1985), the structure of the task of configuring the components of VAX computers performed by R1 (McDermott, 1982) with its order processing front end XSEL could be described as:

SPECIFY + DESIGN { + ASSEMBLE }

while that for the task which the diagnostic medical expert system MYCIN (Shortliffe and Buchanan, 1975) performs could be characterised as:

MONITOR (patient state) + DIAGNOSE (disease category) + IDENTIFY (bacteria) + MODIFY (body system or organism)

This appears a possible source of data to establish a task taxonomy, but before developing such an analysis, a view of what description is required for each task to be useful for knowledge acquisition.

In order to assess the necessary components of an analysis of generic tasks for knowledge acquisition we must look to the sensitivity of currently applicable psychological theory. A recent review of task analysis techniques applicable to the knowledge relevant to user interface design (Wilson, Barnard, Green and Maclean, 1988) assesses techniques under four headings:

Knowledge: The breadth of knowledge classes addressed by the technique and the depth of description offered of those classes.

Task dynamics: The goals of the task performer, the short term changes that take place during task performance (in both processing and actions on external objects) and the long term changes that take place across successive performances of a task (e.g. learning).

The Cognitive Limitations on Processing: The extent to which a functional cognitive model is embodied in the technique and any constraints on the size and speed of memory and processing which are incorporated in the technique.

Use of the Technique: How well the technique is specified to be used unambiguously and how much skill and knowledge must be brought to the technique to use it.

When these criteria are applied to task analyses for knowledge acquisition, the desirable technique would include a taxonomy of generic task types, a set of models for each generic task and guidance as to how to elicit the knowledge relevant to each task.

Each model for a generic task should address each of these points. It will need to describe the knowledge required to perform the task. This will require an indexing of the generic task model into the taxonomy of task models and some rules as to there combination. Otherwise, it will take the form of the type of knowledge output by the task (the solution), the classes of objects expected, and the classes of attributes that those objects should have. At a different level, the knowledge available from different knowledge sources (e.g. guidelines, manuals, databases) should be indicated.

The task dynamics would include the goals of the task, and to map the model onto the system representation, a description of the inference strategy that is expected for the task to account for the short term

changes. The inference strategy will of course interact with and constrain the knowledge of the objects in the task. The long term changes in the task performance should be described as requirements for explanation of the task and requirements for maintaining a model of the task as information changes.

The technique which employs task models should be stated so that it is easy to use to analyse tasks without the knowledge engineer having to bring a large amount of psychological or other specialist knowledge to it.

Consequently, the model of a Generic Task should contain:

The type of problem the generic task addresses

The inference mechanism/strategy expected

The form of knowledge used in the solution

The (meta) classes of objects expected

The (meta)classes of attributes expected

The type of explanation mechanism which is possible

Requirements and methods for maintaining task knowledge

The generic task itself does not contain user goals since the task is for use in acquiring knowledge of the expert's or source task and not in designing the interface to the task the user of a KBS developed to perform it will himself perform. Similarly it is unlikely to contain real time changes in task performance since these will vary at too low a level of description for the task.

The task analysis technique will therefore consist of a taxonomy of task models each of which contain the information in the above form and a technique for using them to analyse tasks. Such a technique does not yet exist. We do not yet have a standard set of task models which can be fitted to all the tasks which the knowledge engineer is likely to encounter. The goal of several research groups is to develop these, but at present the partial analyses which they have put forward are the best available. These analyses vary as to the task types addressed and the content of the generic task models proposed. The types of task to which KBS's have been applied for the longest are classification and diagnostic tasks, so that group of tasks has been most thoroughly described. The types of design task have been studied less and the analysis of these is less well developed. As more task types are identified, the structure of the set of generic tasks changes.

The sets of generic tasks proposed by different groups of researchers will be described here to illustrate the methods used for developing task taxonomies and generic task models, as well as the direction of development that current research in this area is taking. Any knowledge engineer can expand the set of task models themselves with their own experience of developing systems in new task areas. Those presented here are to be used as examples on which to base such development, they do not constitute the only or the best models possible. However, they do show some of the flaws in taxonomies which have been developed so that knowledge engineers should not incorporate them into their own structures.

The progression illustrated by these examples starts by looking at expert systems which have already been developed and attempts to categorise these objects into systematic lists. Then, the tasks which are performed by these various systems are extracted from the whole system and described. This list of tasks is then structured with respect to a theory of systems, and what tasks can be performed on them. Further analyses of tasks performed outside the KBS domain are then introduced to add to this categorisation. Finally, an attempt is made to map the taxonomy that exists onto a cognitive rather than system, oriented view of tasks.

Classification and diagnostic tasks will be described in less detail than design tasks since the knowledge engineer who has to tackle design tasks will find less information on design in the literature and much of that is less accessible than information on other task types.

Generic Categories of Knowledge Engineering Applications.

The first major attempt to structure a set of task models for KBS development was that of Hayes-Roth, Waterman and Lenet (1983) when they presented a list of generic categories of knowledge engineering applications (see Table 2). These were put forward as distinct types of knowledge-engineering applications rather than as generic task models to aid the analysis of knowledge or knowledge acquisition. However, the objective of the list was to aid the extraction, articulation and computerisation of experts' knowledge so they serve as a starting point to show the direction in which research is progressing.

center tab (%) allbox ; c c l l. Category%Problem Addressed Interpretation%inferring situations from sensor data Prediction%inferring likely consequences from a given situation Diagnosis%inferring system malfunctions from observables Design%configuring objects under constraints Planning%designing actions Monitoring%comparing observations to plan vulnerabilities Debugging%prescribing remedies for malfunctions Repair%Executing a plan to administer a prescribed remedy Instruction%Diagnosis, Debugging and repairing student behaviour Control%Interpreting, predicting, repairing, and monitoring system behaviour

Table 2: Generic Categories of Knowledge Engineering Applications (after Hayes-Roth *et al*, 1983).

In their analysis, interpretation systems infer situation descriptions from observable data about the situation. They are used to explain observed data by assigning symbols to them which describe a situation that accounts for the data in systems such as those used for surveillance or speech understanding. Prediction systems infer the likely consequences of described situations in systems such as those used for whether forecasting. Diagnosis systems infer system malfunctions from described data for the purpose of medical diagnosis or electronic fault finding. Design systems develop configurations of objects that satisfy the constraints of a design problem for the purposes of circuit layout or building design. Planning systems design actions which take place over time. Monitoring systems compare observations of system behaviour to features that are important to the successful outcome of plans and normally correspond to potential flaws in plans. Debugging systems prescribe remedies for failures while repair systems develop and execute plans to administer remedies to diagnosed problems. Instruction systems diagnose student behaviour and attempt to debug it. Control systems adaptively govern the overall behaviour of a system.

Since this is one of the earliest classifications of KBS task and system types, it is one of the most discussed and criticised. It has been criticised (e.g. Reichgelt and Van Harmalen, 1986) since some of the classes used are more general types of other classes. The criticism is clearly appropriate for several cases: debugging systems may monitor a system, diagnose a malfunction, then plan a remedy, and may execute the plan; repair systems always both plan and execute the plan; instruction systems monitor, diagnose, plan a remedy and execute the plan; Control systems contain most of the above primative acts. It is therefore clear that this is not a list of either primative KBS types or of primative tasks which should be combined together to produce the task performed by KBS's or experts. This is a list of system types that have been developed and classified by the way the developers or users see their role.

A set of primative tasks has been produced from this list by Reichgelt and Van Harmalen. They argue that the difference between *diagnosis* and *interpretation* tasks is that while both infer a situation description from data, the situation description in a diagnosis system must be a malfunction while interpretation systems have no constraint on the type of state they infer. Consequently they view diagnosis systems as a sub-class of interpretation system. Similarly, they argue that planning systems are a sub-class of design systems where the design is a temporally executable plan, and that debugging systems are a sub-class of planning system where the plan is always directed at correcting a malfunction. This argument results in four primative types of task: interpretation (or classification), monitoring, prediction (or simulation), design.

As with Hayes-Roth *et al*, Reichgelt and Van Harmalen were looking for a set of tasks which can be used as the building blocks for expert system control structures rather than tasks which could be identified during a knowledge acquisition task analysis. Although their list is therefore guided by constraints from logic and efficiency rather than psychological saliency, it offers a further step on route to a psychologically salient set of task models.

Reichgelt and Van Harmalen draw a distinction between tasks which recursively *monitor* a system's outputs over time and compare them with a plan, and tasks which take a system's outputs (possibly over time) and *classify* them onto a high level description. The distinctions between several tasks are confounded

here. The first task takes input data into internal form. A second task classifies sets of internal data (which may be directly re-coded external data or inferred information) against a set of higher level descriptions. There is a third task which outputs higher level descriptions to the user, and a fourth which outputs higher level descriptions into internal form (stored inferences). There is also a distinction as to whether the high level description is a temporal plan, or an atemporal object description. The first three tasks performed once each in succession using atemporal object descriptions make up the *classification* task. The *monitoring* task consists of a cycle of taking input data, classifying it against a temporal plan and storing inferences, with the task of outputting inferences only being introduced when a particular classification (usually an error) is reached. This confusion has serious consequences for Reichgelt and Van Harmalen's purpose of developing machine representations, but may not appear to for knowledge acquisition since this description appears too detailed to be cognitively salient. However, it is a confounding which will effect later arguments in this chapter.

The *simulation* task starts with a system and a requirement for change, and the dependent changes must be inferred. It is usually impossible to enumerate all the possible solutions to such problems in advance. The structure of the task involves a bottom up control regime whereby changes in the state of the system are interpreted as changes in the behaviour of sub-systems. This information can then be used to predict the changes in the behaviour of the overall system. Consequently, the entities which will exist in this task will include a system, its component sub-systems, features of these which will change, and the mechanisms by which the sub-systems interact.

Since the planning and design tasks of Hayes-Roth *et al* appeared to differ only in as far as the product was temporally constrained or not, the fourth task Reichgelt and Van Harmalen propose is a general *design* task which involves the construction of a complex entity to meet certain constraints. These constraints will specify both the target or end state and the present state. The present state may be a set of simple component entities which will be specified with constraints as to their combination that will have to be met when constructing the complex entity. In this analysis, the entity under construction may be a spatial system such as a building, or a temporal configuration of actions such as a plan. The major difference between this and the previous tasks is that they were analytical whereas design is synthetic and the solution cannot be found in a pre-determined set of possible solutions. There may be conditions under which the solution space is enumerable, although usually it is too large. Reichgelt and Van Harmalen do not further analyse the sub types of design task or describe the cases where the solution space may be enumerable.

A second criticism of Hayes-Roth *et al* has been provided by Clancey (1985). He notes the progressive subsumption in the list that Reichgelt and Van Harmalen did, but attempts to overcome it by addressing the concept of a *system* in greater detail and trying to avoid the confusions that arise from viewing the types of system as *objects* where it is unclear whether programs or procedures are objects or processes. This distinction should not be dismissed as only being relevant to machine representational issues since it will bear on the way the tasks involved in knowledge acquisition are viewed and analysed.

Hayes-Roth *et al* describe knowledge as consisting of descriptions, relations, relationships and procedures in some domain of interest (after Bernstien, 1977), although they accept that in practice that knowledge does not appear in some precipitated form that neatly fits such abstract categories. Although they differentiate between knowledge and skill, they do not describe a method for analysing a particular problem into generic units during knowledge acquisition. Although Reichgelt and Van Harmalen do not describe the content of their generic tasks in the detail requested above, Clancey takes both the generic task description and the taxonomy itself much further.

System Based Generic Operations

Clancey (1985) argues that a system can be characterised simply in terms of inputs and outputs. He assumes that tasks will either interpret what exists in a system or construct a new system. This results in the tasks of *interpretation* which is similar Reichgelt and Van Harmalen's *classification* (or interpretation) and *construction* which is similar to their *design.* At this level he does not appear to account for their other two tasks. However, Clancey describes both *monitoring* and *simulation* (or prediction) as sub-types of *interpretation* since he provides a hierarchy of task types under each of his two major headings rather than limiting himself to a single dimensional list. His complete hierarchies are shown in Table 3.

Interpret		working system
|	identify	take input and map it onto system knowledge
|	| monitor	detect discrepancy
|	| diagnose	explain monitored discrepancy
|	predict	describe outputs for given inputs
|	control	describe inputs to produce given outputs
Construct		Non-working system
|	specify	construct a system description
|	design	conceptual construction
|	| configure	organise objects as a structure
|	| plan	organise objects within a process
|	assemble	physical construction of a system
|	| modify	change the structure of an existing system

Table 3: Generic Operations (after Clancey, 1985).

The *interpret* tasks all operate on an existing system. Consequently, that existing system can give outputs which can be taken in and *identified* by comparing them with a knowledge base of a design and either simply categorise the current state in the system by *monitoring,* or explain that state in terms of discrepancies between the actual system and a standard system. The existing system could be given inputs and the *predict* task would be used to determine what the outputs of that system would be. Alternatively, a known system may be required to produce known outputs, in which case the *control* task can be used to generate the inputs that would be required.

All tasks which operate on a system may be classified by the binary choice of analysis (interpretation) or synthesis (construction) at the top level. The analysis tasks can be broken down into the three categories of recognition (identification), prediction and control on the basis of whether the single unknown is the input, output or system itself.

There is a conflict between the *monitoring* and *classifying* tasks proposed by Reichgelt and Van Harmalen and the *monitoring* and *diagnosing* tasks suggested by Clancey which requires further elucidation. A confounding was shown above in the distinction between Reichgelt and Van Harmalen's two tasks. This was due to the way in which they accounted for time in both the performance of the task and the type of descriptions being used for the classification. The monitoring may occur at one time or recursively, and the data structure could be temporal or not. Clancey's tasks both take place on a 'running' system and detect a deviation from a standard. He does not distinguish between whether the description is temporal or not, but between whether the task involves the 'detection' of discrepancies or 'explains' the monitored behaviour in terms of discrepancies. That is, the output from *monitoring* is a less rich description than that from *diagnosis.* As with Reichgelt and Van Harmalen's distinction, both tasks include the same initial input of data, and the distinction between them arises with the process which produces a different output from each task.

For Clancey, the *simulation* task is subsummed as a special type of *prediction* where a computational model which is complete at some level of detail is used to predict the outputs resulting from known inputs. This is not the same as Reichgelt and Van Harmalen's *simulation* where a system itself is modified in a simulation and then both its outputs can be tested for known inputs and its inputs tested for known outputs. These two tests correspond to Clancey's *simulation* and *control* tasks, although the modification prior to the testing is a *construction* task.

Whereas Reichgelt and Van Harmalen only described one task which covered all forms of design, modification, construction or planning, Clancey provides a richer analysis of these tasks. The first task of the *construct* type is *specify* which refers to the operation of constraining a system specification in respect of both other defined systems and the real world. *Assemble* is a general task referring to the physical construction of a system. This task would could require robot assembly in a computer system performing the task. The one sub-task *modify* is included to cover the transformation of a system to effect a redesign by 're-assembly' given a required design modification.

Design is characterised as a general conceptual operation describing the spatial and temporal interactions of components including a characterisation of both of structure and process. Having united them under the

same task of *design* Clancey then draws a distinction between the two tasks of *configuration* and *planning* which had existed in Hayes-Roth *et al's* analysis. Reichgelt and Van Harmalen's arguments for the computational equivalence of the two tasks are not being denied here. However, the use of a hierarchical structure of tasks permits both the computational generality and the instinctive distinction to be expressed which their list would not. However, Clancey does not use the temporal nature of a plan the basis of his distinction. The *configure* task pieces together components into a whole so that the function whole will show desired properties. A typical example of this would be VLSI design where physical objects are pieced together so that the behaviour of the parts interact to produce the desired system behaviour. In contrast the *planning* task does not operate on well structured systems but on a general system (e.g. the world) which surrounds and transforms an entity (e.g. a person).

A potential generic task structure for the *configure* task is provided in the work of Chandrasekaran and his co-workers (Brown and Chandrasekaran, 1986; Bylander and Chandrasekaran, 1987:236,237):

> *Problem Type:* object synthesis by plan selection and refinement.
>
> *Problem:* Design an object satisfying specifications. An object can be an abstract device, e.g. a plan or program.
>
> *Representation:* The object is represented in a component hierarchy in which the children of a node represent components of the parent. For each node, there are plans that can be used to set parameters of the component and to specify additional constraints to be satisfied. There is additional knowledge for selecting the most appropriate plan and to recover from failed constraints.
>
> *Important Concepts:* The object and its components.
>
> *Inference strategy:* To design an object, *plan selection and refinement* selects an appropriate plan, which, in turn requires the design of sub-objects in a specified order. When failure occurs, failure handling knowledge is applied to make appropriate changes.
>
> *Examples:* Routine design of devices and the synthesis of everyday plans can be performed using the generic task; e.g. MOLGEN (Friedland, 1979), R1 (McDermott, 1982).

This generic task does not provide information on the explanation mechanism required for the task, nor any guidance in maintaining the system. Also, the description of the representation is very implementation oriented, and would be of little use in guiding questions during knowledge acquisition, but it illustrates a step closer to the generic task description required above than those offered by earlier analyses.

However, these general descriptions do still not offer a specification of the structure of knowledge required which is detailed enough to motivate knowledge acquisition.

Generic Task Models in KADS

The most complete set of generic task models proposed both in breadth and in detail are presented in the KADS knowledge acquisition methodology. This is currently under development (see Breuker and Wielinga, 1987; Haywood, 1987) and a complete task taxonomy and library of task models will be available in the "KADS Handbook" with supporting software tools (see Anjewierden, 1987). Table 4 shows an initial taxonomy of problem types from this methodology which is a development of that presented by Clancey. These tasks are intended to be accepted by knowledge engineers as the appropriate categories. However I will continue to assume that knowledge engineers will modify any taxonomy of tasks on the basis of their own experience and will therefore describe the reasons why the distinctions have been made to provide a basis for such personalisation.

Analysis
| identify
| | classify
| | | simple classify
| | | diagnosis
| | | | single fault diagnosis
| | | | | heuristic classification
| | | | | systematic classification
| | | | | | causal tracing
| | | | | | localisation
| | | | multiple fault diagnosis
| | | assessment
| | monitor
| predict
| | prediction of behaviour
| | prediction of values
Modification
| repair
| remedy
| control
| | maintain
Synthesis
| transformation
| design
| | transformational design
| | incremental design
| | | single stream incremental design
| | | multiple stream incremental design
| | configuration
| | decomposition
| planning
| modelling

Table 4: Taxonomy of Problem Types (after Breuker *et al*, 1987)

The top level categories of *analysis* and *synthesis* are similar to Clancey's *interpret* and *construct* tasks. In Clancey's analysis there was some confusion about whether systems should exist or merely that designs for systems be known. Similarly, it was not clear that *modify* was a valid form of assembly since a system had to exist for the *modification* to take place, whereas it could not for *assembly*. To overcome these doubts a third category of modification has been introduced to illustrate that a continuum and not a discrete cut off exists between *analysis* tasks, where a solution has to be identified, and *synthesis* tasks, where a solution has to be constructed.

The analysis tasks here are an expansion of Clancey's *interpretation* tasks. The use of a hierarchy has permitted the further differentiation of these on the basis of principles already mentioned.

The *synthesis* tasks however are somewhat different to Clancey's. The top level classification is made on the type of input and output that the task has. Design tasks take functional requirements as their input and produce descriptions of an artifact as output. The distinction from Hayes-Roth *et al* between *design* operating on artifacts and *planning* operating on temporal events has been revived here despite the criticisms made of it earlier. Otherwise these two tasks are the same. The *modelling* task takes the same inputs as design, but in addition it requires a set of "data" of which the solution must be an abstraction. In many cases descriptions of component elements must be abstracted from the data. The output of the task is a description of an artifact as for design. The *transformation* task takes as input a description of an artifact and a known transformation to apply to it. The output is a description of a new artifact after the transformation has been applied. This is a task which could be classified as either design or modification, illustrating

the continuum that exists between them.

The set of task models proposed for design tasks in KADS is richer than any of the previous sets. The task model for the general design task itself can be used to illustrate the greater depth of description of generic tasks that KADS aims at than the earlier attempts.

It is assumed that design is a process whereby an entity must be synthesised to conform to a collection of specified requirements or constraints. The requirements are usually ill-specified, therefore the first stage of the design task is to analyse the informal requirements and to produce a formal specification. The second stage of the task will be to synthesise a detailed design from the formal specification. There are therefore five concepts involved at this level of description: two processes - *analysis* and *synthesis* - and three objects - *informal requirements, formal specification* and *detailed design.* The process of analysis can be performed on the informal specification by either its *expansion* or its *transformation.*

This model is further elaborated in KADS to take into account the conceptual model of the final product which specifies the structure of the product entity. The conceptual model is required for design tasks that involve configuration since the global structure of the product object is not specified in these. This conceptual model will be developed prior to the detailed design from the formal specification. Therefore the single process of *synthesis* must be decomposed since a process is required to *select* or *aggregate* elements of the formal specification into the conceptual model. A second process is required to *transform, expand* or *refine* this conceptual model into the final detailed design. The resulting structure for the design process is shown in Figure 1

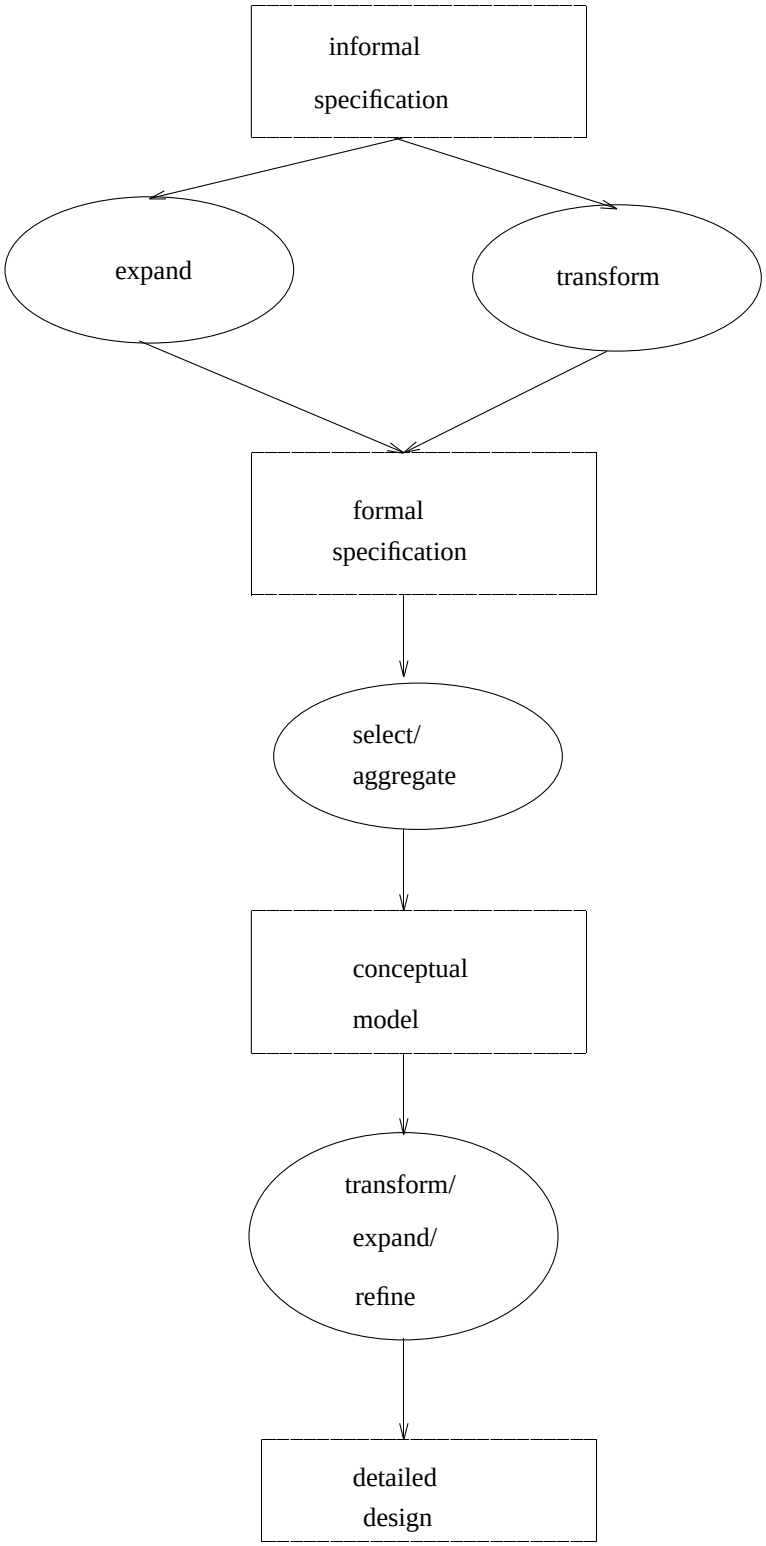

Figure 1: Global Structure of the design process (after Breuker *et al*, 1987).

Earlier in this chapter it was suggested that a generic task description should contain not only the type of problem the task would address and the inference mechanism, but also the form of knowledge used in the

solution. The KADS system has presented a taxonomy of problem types, including sub-types of design. It has included in the description of design various objects and processes. For the first time in this series of task descriptions these are further expanded to describe the type of knowledge expected in each. For example, the object *informal problem statement* is described as (after Breuker *et al* 1987):

> *meta-class* **informal problem statement** This is the input of the design process, an informally formulated specification of the structure to be designed. Examples: input/output specification, specification by analogy, description, component/function specification.
>
> *consists-of:* **informal_system_specification** constraints, requirements.

Similarly, the other object types are described as meta-classes of objects, and have types within the meta-class further defined and exemplified. In the same way, the processes are described, although in the terminology of KADS, these are termed *knowledge sources.* For example, the process which builds a formal specification from an informal specification of a problem is described as (after Breuker *et al* 1987):

> *knowledge source:* **transformation and expansion** These knowledge sources build a complete formal specification of the system to be built.
>
> *input:* **informal problem statement** This is a description of the problem that does not necessarily include all parameters, function and constraints, but just the major ones.
>
> *output:* **formal specification** The formal specification is a description of the system to be designed which holds all constraints, functions and parameters. This statement is given in the language of the domain.
>
> *methods:* Currently the methods are not specified by KADS although they are assumed to be domain dependent.
>
> *domain knowledge:* This probably includes knowledge about the functions required in systems the expert knows about, knowledge about what is achievable etc..

Currently these illustrate that although the KADS project attempts to develop knowledge type descriptions acknowledging that they are a desired feature of a generic task model, it has not yet been completed for design tasks. Much further research is required in analysing design situations for these to be completed but they illustrate for the first time the depth of description required.

There are several options as to how the sub-types of the design task can be structured. Those shown above are only tentative within the KADS method, but they can be used to illustrate the alternatives and the grounds for assessing them. It is generally agreed that design involves objects in states and that those objects have attributes which can be set to values. It is also agreed that constraints can be applied to determine acceptable values. There are three major distinctions possible.

The first distinction to be drawn is whether all the constraints can be stated initially as requirements or whether some can only be generated in response to a design. The first option provides the general KADS design model. The second requires a loop to exist which allows the analysis (or diagnosis) of a design to produce new constraints which can be fed into the requirements phase. This process would combine the KADS *design* and *analysis* tasks together into a more complex design task.

The second distinction relates to this and determines when the loop should terminate, by setting the standards of acceptable diagnosis in design. The first option is to accept a design which meets the set of constraints provided. The second option is to determine all the designs that meet the provided constraints and then look for constraints which will select amoung these, until only one optimal design remains. When a non-reducable set of designs remains a random selection will be required. The first option is usually associated with engineering design (e.g. the generic design model suggested by Chandrasekaran above) and the second with more creative artistic design.

The third distinction relates to how the constraints are applied to the whole design and sub-parts of it. It may appear that design components would have to be altered, but even in engineering design the range of options can be broad (after Chekayeb, Niedzweeki, Connor; 1987):

1) Routine Design - Select a known entity and transform it by modifying attribute values, e.g. a bigger bottle. When the entity and the attribute values are known this task becomes the *transformation* type of synthesis task in KADS. The process of determining the transformation and applying it constitute the *transformational design* task.

2) Re-design - Select an old design for an entity, then modify values of attributes of its components. Modifying one component may effect other relationships to other components and further constraints will be brought to bear. This will require the *decomposition* of an old design followed by *transformational design.* in an overall process often called hierarchical design.

3) Innovative Design - Synthesise a new configuration from old primative objects. This becomes the *configuration* task in KADS.

4) Creative Design - Create New Primatives - then apply innovative design to these primatives. New primatives can be created by converting the primatives that already exist in one relation lattice to another lattice where they are not primative.

The fourth distinction is whether all the options in the design space are generated before the constraints are applied or whether the constraints are applied to each step in the generation of the design solution space in order to restrict the solution search. Gero and his colleuges (e.g. Rosenman, Coyne, and Gero, 1987; Oxman and Gero, 1987) have developed design synthesis systems which generate all possible design solutions that can result from a present state before applying constraints to them. This may be computationally effective, but psychological evidence (Johnson-Laird, 1987) suggests that it is not a method used in human creative reasoning and therefore should not provide a generic task model for knowledge acquisition.

Within this set of options a large number of design tasks can exist. The correct set of "Generic Tasks" for knowledge aquisition can only be derived through further research and analysis. This statement of the options however, should allow the selection of possible interpretations for any design task facing a knowledge engineer.

The State of Research into Task Models.

The current state of task taxonomies has been reviewed here at length, although it will be necessary to refer to the KADS Handbook to use the full range of the detailed models put forward their. It is clear from this description that current models do not meet the criteria laid out for them in the introduction. The taxonomy of models has been reasonably developed for analytic tasks, although it is still incomplete for synthesis and modification tasks. Consequently the identification of generic tasks in any real task is not supported by an easy to use method. However, the examples given show that it is possible to describe some real tasks in terms of generic tasks. When the generic tasks have been identified in a real task they should provide the knowledge engineer with sufficient information to know what types of knowledge to acquire, and motivate the acquisition strategy. The present set of models do describe the inference strategy expected and begin to describe the classes of knowledge expected, although these are not linked to elicitation strategies. Perhaps the biggest problem with the present models is that they do not address the issues of explanation in the task the KBS user will perform or maintenance by the user of the KBS. These two aspects of task models are still very much research issues.

Although task models are still under development knowledge engineers should see the advantages offered by their use during the analysis of aquired information from this review. They should also be in a better position to develop their own models as their experience progresses. It should of course be remembered that the task models described here are to be used by the knowledge engineer in analysis and to structure elicitation sessions and not presented to the experts during knowledge elicitation.

References

Anjewierden, A. (1987) Knowledge Acquisition Tools. AICOM 0(1).

Bainbridge, L. (1986) Asking Questions and Accessing Knowledge **Future Computing Systems, 1** (2), 143-149.

Barnard, P.J. (1987) Cognitive Resources and the Learning of Human-Computer Dialogues. In J.M. Carroll (ed.) **Interfacing Thought: Cognitive Aspects of Human-Computer Interaction** Cambridge, Mass.: MIT Press.

Breuker, J. and Wielinga, B. 1987 Use of Models in the Interpretation of Verbal Data. In A. Kidd (ed.) **Knowledge Acquisition for Expert Systems.** New York, NY: Plenum.

Breuker, J, Wielinga, B, van Someren, M., de Hoog, R., Schreiber, G, de Greef, P., Bredeweg, B., Wielemaker, J., Billeaut, J.-P., Davoodi, M., and Hayward, S. (1987) **Model-Driven Knowledge Acquisition Interpretation Models.** Deliverable Task A1, Esprit Project 1098, Commission of the European Community.

Brown, D.C. and Chandrasekaran, B. (1986) Knowledge and Control for a mechanical design expert system. **Computer, 19** 92-100.

Bylander, T. and Chandrasekaran, B. (1987) Generic tasks for knowledge-based reasoning: the "right"level of abstraction for knowledge acquisition. **Int. J. Man-Machine Studies, 26,** 231-243.

Card, S., Moran, T. and Newell, A. (1984) **The Psychology of Human-Computer Interaction** Hillside, N.J.: Lawrence Erlbaum Associates

Chekayeb, Niedzweeki, Conner (1987) Paper presented at Second International Conference on Engineering Applications of AI.

Clancey, (1985) Heuristic Classification **Artificial Intelligence, 27,** 289-350.

Friedland, P. (1979) Knowledge based experiment design in molecular genetics. **Ph.D. thesis** Computer Science Dept. Stanford University.

Frieling, M., Alexander, J., Messick, S., Rehfuss, S. and Shulman, S. (1985) Starting a Knowledge Engineering Project: A Step-by-step Approach. **AI Magazine 6** (3), 150-165.

Gammack, J.G. and Young, R.M. (1985) Psychological techniques for eliciting expert knowledge. In M.A. Bramer (ed.) **Research and Development in Expert Systems,** Cambridge: Cambridge University Press.

Hayward (1987) How to build knowledge based systems: techniques, tools, and case studies. In **ESPRIT '87: proceedings of the Esprit Conference,** pp 665-687. Commission of the European Community: Brussels.

Hoffman, R.R. (1987) The problem of extracting the knowledge of experts from the perspective of experimental psychology. **AI Magazine,** Summer 1987, 53-67.

Hayes-Roth, F., Waterman, D.A., and Lenet, D.B. (1983) An overview of expert systems. In F. Hayes-Roth, D.A. Waterman, and D.B. Lenet **Building Expert Systems,** Addison-Wesley: Reading, Mass..

Johnson-Laird, P.N. (1987) Reasoning, Imagining and Creating. **Bulletin of the British Psychological Society, 40,** 121-129.

McDermott, J. (1982) R1: a rule based configurer for computer systems, **Artificial Intelligence, 19,** 39-88.

Miller, R.B. (1962) Task Description and Analysis. In R.M. Gagne (ed.) **Psychological principles in system development.** New York, NY: Holt, Rinehart & Winston.

Oxman, R and Gero, J.S. (1987) Using an expert system for design diagnosis and design synthesis. **Expert Systems, 4** (1), 4-15.

Rasmussen, J. (1986) **Information Processing and Human-Machine Interaction: an approach to cognitive engineering.** Amsterdam: North Holland.

Reichgelt, H. and van Harmelen, F. (1986) Criteria for choosing representational languages and control regimes for expert systems. **The Knowledge Engineering Review,** 2-17.

Rosenman, M.A., Coyne, R.D. and Gero, J.S. (1987) Expert systems for design applications In **Applications of KBS's: based on the proceedings of the second Australian conference.** Turing Institute in association with Addison-Wesley, 66-84.

Schweickert, R., Burton, A.M., Taylor, N.K., Corlett, E.N., Shadbolt, N.R. and Hedgecock, A.P. (1987) Comparing knowledge elicitation techniques: a case study **Artificial Intelligence Review, 1,** 245-253.

Shortliffe, E.H. and Buchanan, B.G. (1975) A model of inexact reasoning in medicine. **Mathematical Biosciences, 23,** 351-379.

Slater, P.E. (1987). **Building Expert Systems: cognitive emulation.** Chichester, U.K.: Ellis Horwood.

Welbank, M. (1987a) A Survey of Knowledge acquisition techniques. **SD Insight Report,** System Designers: Camberley U.K.

Welbank, M. (1987b) Perspectives on Knowledge Acquisition. In C.J. Pavelin and M.D. Wilson (eds.) Proceedings of the SERC Workshop on Knowledge Acquisition for Engineering Applications. **Rutherford Appleton Laboratory Report, RAL-87-055,** 14-20.

Welbank, M. (1987c) Knowledge Acquisition: A survey and British telecom experience. In T. Addis, J. Boose and B. Gaines (eds.) **Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems.** Reading University: Reading, U.K.

Wilson, M.D., Barnard, P.J., Green, T.R.G., and Maclean, A. (1988) Knowledge-Based Task Analysis for Human-Computer Systems, in G.C. van deer Veer, T.R.G. Green, J.-M. Hoc and D. Murray (eds.) **Working with Computers: theory versus outcome.** London, U.K.: Academic Press