Knowledge Based System Development as an Engineering Process

by

M. Davoodi

A Thesis submitted for the degree of Doctorate of Philosophy

Brunel University, London

1989

Abstract

Knowledge Based System (KBS) development is a difficult and challenging task, in particular in knowledge intensive domains. The traditional view of knowledge engineering is one of mining experts' knowledge and *somehow* transforming it into a machine usable form. This process, in general, suffers from insufficient or misconstrued representation of experts' problem solving behaviour. It is also unstructured and unduly biased at an early stage by design and implementation issues - normally in the form of incremental prototyping.

We believe that both knowledge acquisition and KBS development for real life applications will require a 'structured' approach. This approach should harness a KBS developer's ability in extracting knowledge and developing systems. The structure should also be sufficiently flexible to allow the knowledge engineer to use his sense of creativity in developing a KBS. This thesis puts forward such a structured approach, in which KBS development is carried out in an engineering fashion. A process in which the worker is provided with an environment for developing knowledge based systems as an engineering process, as opposed to that of an artform or crafting.

The main emphasis of this work is that part of the process which deals with the analysis and design phases in developing KBS. The analysis is performed at an 'epistemological' level, not coloured by design or implementation issues. The output of this phase captures both an expert's problem solving capability, and the business constraints placed upon the intended system. This is then used by the design process in order to create an optimal, workable, and elegant design architecture for the ultimate system.

Acknowledgement

I am indebted to my supervisor Dr. Michael Elstob for his learned objectiveness, and his high human qualities which have been most inspiring.

This research has been funded partially by the Commission for the European Communities' ESPRIT programme under project number 1098. The partners in this project are STC plc. (UK), University of Amsterdam (UvA, The Netherlands), Cap Sogeti Innovation (France), SCICON ltd. (UK), SCS GmbH (W. Germany), and KBSC (UK).

Within the project 1098 (P1098), I gratefully acknowledge the useful discussions and project work I have carried out with Simon Hayward (STC plc.), Prof. Bob Wielinga (UvA), and Dr. Joost Breuker (UvA) in the early part of P1098. In the latter part of P1098 I have had the good fortune of leading the *design methodology* team in P1098. Within this period I have benefited intellectually from my project work and discussions with the other members of the design team, namely: Guus Schreiber (UvA), Prof. Bob Wielinga (UvA), and Bert Bredeweg (UvA).

In a collaborative research work of this type it is difficult to identify clearly the extent of the contribution of each individual involved. To the extent that this is possible, I shall claim chapters one, six, eight, nine, and the appendices of this thesis as essentially my work. The other chapters are largely influenced by the work of the other researchers within P1098, most of whose names appear above. For this I am greatly indebted to these people. The arguments put forward throughout the thesis nevertheless reflect my own intellectual stance, which may not necessarily coincide with those of fellow researches within P1098.

I would also like to acknowledge the moral support and encouragement that I was given by William Clancey (KSL, Stanford Uni., USA) in my analysis of his work on

"NEOMYCIN" (see chapter 8, below). Our discussion convinced me that my study of "NEOMYCIN" was pertinent to the development of part of the design methodology, although Bill might not have been aware of this at the time.

Massoud Davoodi

Brunel University, 1989.

Table of Contents

3. REQUIREMENTS ANALYSIS

10. References

Appendix A: a KADS Prototype for COMPARE Knowledge Source:

11. A Brief Description

Appendix B: KBS, from Requirements to Design:

12. BOTAID

1. Introduction

In the course of this thesis, we shall set out to describe an emerging methodology called "Knowledge Acquisition and Documents Structuring support" (referred to as KADS throughout this work). The main concern of KADS is the conduct of KBS development as an *engineering process*. An engineering process, in our view, is one in which a framework containing the appropriate tools and techniques for developing systems is provided. The framework for KBS should support a knowledge engineer from "knowledge acquisition" to "system implementation". In this context, KADS will contain:

- A number of 'models' for interpreting knowledge and information acquired from experts. These models will also provide the means for classifying different types of expertise, as well as identifying the components of knowledge within each type. The models will therefore provide the tact and focus which a knowledge engineer will require in conducting interview sessions with an expert. We have called these models "interpretation models" (also referred to as IM throughout this thesis; see chapter 5).

- A number of phases (currently two of "analysis" and "design") for developing a KBS.

- A number of 'models' for capturing knowledge and system architecture components at analysis and design phases respectively.

- A set of layers within each phase, each of which will correspond to aspects of 'conceptual models' (see chapter 4) in the analysis phase, and system architecture (see chapter 7) in the design phase.

- A set of vocabularies within each of the above layers, in order to describe the corresponding aspects of each phase.

- A method of describing and incorporating business, user community, and environmental constraints placed upon the intended system. We refer to these constraints as the 'external view' (see chapter 3). This is in contrast to the problem solving component of the system known, in KADS, as the 'internal view' (see chapter 3).

- A set of Computer Aided System Engineering (CASE) tools for automating the creation and maintenance of different phases of KADS. This is known as 'KADS System' which is only partially developed, and will not be described to any great depth in this thesis. The use of a CASE tool is of assistance to a developer, but it is not essential to the conduct of a methodology.

An engineering process, as we regard it, will admit both of a 'structured' approach and a sense of 'creativity' in developing systems. We argue that a developer's sense of creativity in developing systems can, in general, be enhanced and focused by having a framework of the type we have just described (above). We also hold that the framework should be a descriptive one, as opposed to one which *prescribes* every step of development. The latter, we believe, will place unnecessary burden upon a knowledge engineer or a system developer, thus detracting from their sense of creativity in applying their previous experiences to new application domains.

We have, therefore, attempted to develop KADS as a *language* for developing KBS. It is a language to the extent that a number of development phases and layers of description are provided within each layer, both within analysis and design models (see chapters 4 and 7). Each layer provides a syntax for combining KADS vocabularies (such as 'concepts', 'relations', etc.; see chapters 4, 5, and 7). The semantics of the

language can be observed in fully defined and instantiated 'conceptual models' (see chapter 6), and 'system architectures' (see chapter 8). These will capture, respectively, the implications of a particular domain of expertise at 'knowledge', and 'system architecture' levels.

KADS, as can be observed from our earlier discussion (above), has grown beyond being simply a knowledge acquisition tool, with the future intent of supporting and harnessing all stages of KBS development and maintenance. Our emphasis in this work will be on the two phases of 'analysis', and 'design' (see below). The rest of this chapter is intended to provide an overview of the whole thesis, and a summary of KADS contribution to the field of KBS.

1.1. KADS Contribution to the field of KBS

The major contribution of KADS methodology to advancing the field of KBS is two-fold:

- The application of a number of abstract models known as "Interpretation Models" for understanding the nature and relevance of knowledge and information, acquired from experts, to problem solving in a particular domain of expertise. A knowledge engineer will use a library of "Interpretation Models" in order to fit the newly acquired knowledge against the components of an IM which appears to be appropriate to the domain at hand (for instance a 'classification' IM for a diagnostic problem). The next stage would be to use the same IM to consolidate and focus the line of interviewing with an expert, by concentrating on asking questions about those components of the IM for which little or no knowledge has yet been attained. The use of abstract problem solving models in this way is a novel one in the field of KBS. We propose that the use of IM in this manner, should help to eliminate the classic

knowledge acquisition bottleneck. A situation which is almost entirely due to a lack of a top-down model driven enquiry method; IM should provide just the models a knowledge engineer will seek.

- KADS provides a language (in the sense which we have described above) for modelling expertise and deciding the underlying system architecture for the intended KBS. This is different to other methodologies, and approaches for developing KBS, in that KADS is not constrained by a number of working systems which are the property of a limited number of researchers belonging in centres of excellence. This open approach to KBS development will allow KADS to cover a large number of models at the analysis and architecture spaces (see chapter 2), with the mapping between the two levels clearly being defined (see chapter 7). It will also turn the methodology into an environment for developing systems in academic and commercial environments alike. KADS is not constrained by computer formalisms (computer languages and shells, etc.), since it is possible to devise KADS analysis and architecture models independently of such formalisms. It will however make use of the appropriate computer languages, shells, or environments, once the architecture of a system is decided. KADS will also use prototyping as an aid to understanding the features of a system during design, rather than as a methodology for developing systems.

The features, we have just described will also form the central hypotheses this thesis will test. In the sense that in developing KBS, we shall need a number of models (i.e. IM) for acquiring knowledge, and a descriptive language for developing the acquired knowledge into a working system. The hypotheses have been tested on fourteen domains of expertise, most of which have resulted in working systems. The domains cover areas such as 'network management', 'process control', 'statistical modelling',

'credit assessment', 'underwriting', 'medical diagnostics', 'hardware configuration', etc. For domains which are relatively easy to understand and develop, KADS will carry too much overhead, and whilst elements of it can be used, its full application is not recommended. On the other, no technique or methodology to this date, can deal with developing KBS for domains of expertise which are too complex. This is both in terms of the type of knowledge involved (eg. tacit knowledge), and the complex nature of the reasoning processes applied to them. In KADS, like most other approaches, we shall need to apply a 'feasibility' check list to a domain, before deciding whether it is possible to develop a KBS for that domain based on today's methodologies, and technology.

1.2. Model Driven KBS Development

Because it takes experimentation to achieve high performance, an expert system evolves gradually. This evolutionary or incremental development technique has emerged as the dominant methodology in the expert system area. The procedure of extracting knowledge from an expert and encoding it in program form is called **knowledge acquisition**.....*The burden of uncovering and formalising the expert's knowledge falls on the shoulders of the knowledge engineer....., at present knowledge engineers must rely on their own skill and insight to guide the knowledge acquisition activity. (Hayes-Roth, et. al., 1983)*

We find the sentiments expressed in this passage most relevant to expert system development to date. We should, however, like to propose that any reasonably complex domain of expertise will benefit from having a problem solving model and architecture devised for it, before prototyping can be used as a support activity. Our use of prototyping, in this context, will be to understand some of the aspects of the underlying system architecture, as well as users' reactions to the emerging system. The proto-

types at this stage will enable us to modify or enhance certain features of the intended system, in order to make a reasonable use of available technology, whilst complying with some of users' requirements. We do not propose that prototyping should be used to bring about the conceptual and initial architectural understandings required. These, we find, to be the concern of modelling techniques, such as those in KADS. We hold that:

- Most KBS which are purely based on an incremental prototyping approach, will address relatively simple domains of expertise,

Or that

- they are likely to suffer from the shortcomings of the computer formalism (eg. expert system shell) which has been used in developing them. An example of this would be to force a 'production-rule based' shell upon a domain which will require a large degree of 'object oriented' knowledge representation. The reason behind the wrong choice of a shell can mostly be attributed to a lack of understanding of the elements of knowledge of the domain prior to implementation. This understanding is made possible through the metaphor of modelling, as provided within KADS.

Figure 1-1 is a simple illustration of the stages through which a knowledge based system development will have to go. A methodology, proper, should cover and support the stages described here so as to deliver 'knowledge acquisition (engineering)', and system development from the realm of simple prototyping into that of an engineering process. The engineering process (as argued earlier) will ensure that the creative sense of the knowledge engineer, and his use of prototyping is conducted within the framework of a methodology.

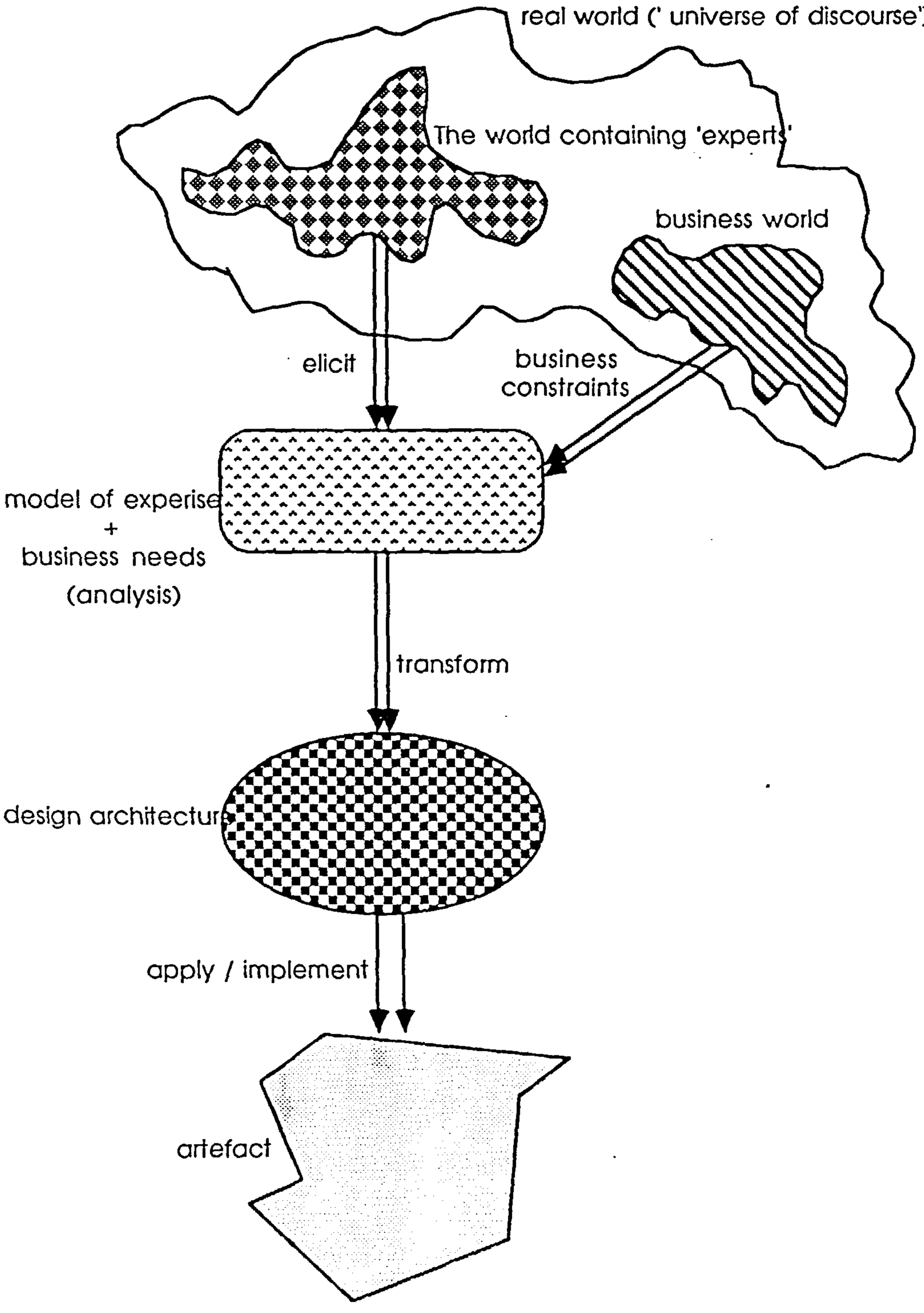

Figure 1-1: *Development Stages of a KBS*

The more the methodology is put to test, and systems and support tools built around it, the clearer and more concrete will become the 'engineering process'.

1.3. KADS Methodology

KADS should ultimately support all stages of expert system development (fig. 1-1). The current version of KADS has been extended to support the design process, yielding very interesting results (see Davoodi, et. al., 1987). It is intended that KADS should be extended to support generic design models, in the same sense that it has been applied to modelling at an earlier stage (fig. 1-1). We intend to concentrate in this work on the 'design' and 'analysis' part of the methodology. This should make it possible to provide a forum for a detailed and useful examination of the said phases.

1.3.1. KBS Philosophy

... to enhance the performance of AI programs, **knowledge is power** *[emphasis added]. The power does not reside in the inference procedure.' The power resides in the specific knowledge of the problem domain.* (Feigenbaum, 1983)

Expert system development has entered a new era in which domain knowledge has assumed the central role in developing knowledge intensive domains, thus power lies in knowledge. The increasingly prominent and real-life applications of expert systems in knowledge intensive domains has made non-domain specific techniques such as 'general problem solving' (GPS) unworkable.

The current belief is that the knowledge engineer should make extensive use of the body of domain knowledge made available to him through reading texts, and interviews with experts. This knowledge should then be consulted both to establish a vocabulary about domain 'facts' (entities known to be true irrespective of the problem solv-

ing behaviour), and the reasoning processes applied to those 'facts'. The ultimate aim of this approach is to bring about a separation between domain 'facts', and (heuristic) reasoning in using them. The origins of the term "knowledge based system" lies in this transition, highlighting the role of domain knowledge in KBS development.

1.3.2. KADS Modelling

We are aware of these developments, and also concerned about the fact that little has been done in the way of understanding expert knowledge at an epistemological level. That is, a knowledge level understanding of all of the interesting domain entities, relations and structures amongst them, and different ways of exploiting them in problem solving. Such an understanding should constitute a 'theory' about a particular expert knowledge domain (thus, the use of the term epistemological), without being unduly biased by design and implementation issues. Such an understanding is a prerequisite to any proper system development, at worst reducing backtracking, and at best avoiding expensive development disasters.

KADS analysis is a process in which such an epistemological study is undertaken resulting in models of problem solving behaviour in different domains. KADS analysis has also been extended to deal with business needs required of the ultimate problem solver (the intended KBS). This issue is particularly important in commercial applications of KBS. Models of problem solving behaviour, and business needs have come to be known as the internal and external views, respectively (Barthelemy, et. al., 1987). The former is concerned with what constitutes the inner (problem solving) aspects, and the latter with the external constraints placed upon it.

1.3.3. KADS Analysis

The analysis stage is concerned with capturing the internal and external views. A clear understanding of these two views are essential to a model-driven, methodological development process.

1.3.3.1 The Internal View

The expert knowledge in solving problems falls into two broad categories of domain 'facts' and reasoning procedures. KADS will represent this knowledge in a four layer model, purely concerned with expert problem solving behaviour at an epistemological level. The layers depend very closely upon one another. Their organisation allows for a good understanding of the domain facts, the interplay amongst them, and ways of exploiting them in problem solving. The four layers together form (our view of) a conceptual model. We shall provide a brief description of these layers.

Domain Layer: All of the facts pertaining to a domain of expertise gathered through reading texts, and interviewing the expert are collected in this layer. The domain entities, in general, consist in concepts, relations between concepts, and structures made up of these relations.

Inference Layer: This layer is concerned with interesting inferences which can be made in the domain. The inferences are assembled in a structure containing primitive operations, with objects as input and output to those operations. The primitive operations are what we call 'knowledge sources' which use domain relations in order to arrive from a given input to a desired output. The i/o objects are termed as 'meta-class objects', these are classification of domain concepts according to the role those concepts play in the reasoning process.

Task Layer: In solving a problem, the expert will consider a goal toward which he sequences his inference (reasoning) steps. This is usually done by breaking down the overall goal (task) into a number of sub-goals the achievement of which performs the

expert task.

The behaviour described here is achieved in this layer by devising a task structure containing statements expressing goals and sub-goals. The sub-goals (sub-tasks) are ultimately satisfied by the application of knowledge sources (operations), which, in turn, use meta class objects as arguments.

Flexible Strategy Layer: This is the most embryonic and difficult of the four layers. It is the least developed of the four layers, since it has to deal with *dynamic planning* and monitoring of tasks. This type of planning has also been a perennial problem in the field of artificial intelligence in general. This layer is concerned with ensuring that a KBS can respond to new and unforeseen situations in an 'intelligent' and interesting fashion, taking actions which would avoid redundant behaviour.

The four layers, put together, describe a system's internal view at an epistemological level, and form the conceptual model. The conceptual model can be used to guide 'knowledge elicitation' in various stages of a development. This is one of the significant aspects which is missing from rapid and incremental prototyping - viz. a lack of model-driven data enquiry.

1.3.3.2 Interpretation Models

These are generic problem solving models applied to a class (or classes) of domains sharing some common, interesting feature captured by the model. For instance, an interpretation model for heuristic classification could apply to any domain requiring that particular problem solving behaviour.

Interpretation models are arrived at by studying domains sharing some common feature we want to capture. One may then generate a conceptual model for the most general of those domains. The conceptual model is then transformed into a more generic

problem solving model, by deleting the domain layer, and replacing meta class names with ones which are generic, to form the interpretation model. The process of transforming a conceptual model into an interpretation model could turn out to be quite challenging and demanding.

Interpretation models once devised can be instantiated to form conceptual models, to be used within different domains for which they have been intended. We place particular importance on interpretation models for general problem solving in KBS.

1.3.3.3 The External View

Business demands placed upon a KBS, need to be considered at the same time as we form our problem solving four layer model. If Knowledge Based expert systems are to assume their proper role in solving problems in commercial environments, they need to accommodate for the external requirements placed on them by those environments.

We consider analysis of internal and external views as part of the analysis global lifecycle model, in which the views will have to be negotiated to reach an acceptable compromise. The compromise should bring about an acceptable trade off between the business needs, and the possibility of capturing the internal view within the current system development technology.

1.4. Design

Design in KADS appears in the methodological spectrum after analysis. A continuum along which the output of analysis stage should be used as input to design (cf. Davoodi, 1987[b]). The input to design is defined in terms of the conceptual model, and a number of statements describing the external requirements.

The input is then transformed (fig. 1-1) into a description capturing WHAT functional-

ity is expected of the ultimate system. This description is then transformed into a design architecture providing a transparent support for the artefact. Ideally, the only decisions to be made at the implementation level should be those to do with house-keeping, and system tidy-up issues (such as garbage collection).

The design language and methodology as we have come to establish, share some aspects of conventional software design, as well as having major points of departure from it. The major distinguishing factor in the design of knowledge based systems is the use of AI problem solving techniques (methods), which will have a significant effect on the architecture of a system.

1.5. KADS Power Tools

KADS uses a development environment based on a number of tools specifically designed and implemented for this purpose. The tools are created in response to requirements within different stages of development within the methodology, and they are integrated together to provide a support environment.

Some of these these tools are:

- Protocol Editor (Anjewierden, 1987), enabling an extensive cross-referencing facility within transcripts (protocols) of interviews with experts. This makes it possible to cross-reference amongst domain concepts, and large segments of text, as well as annotating parts of text, and so forth.

- Tools providing for editing a glossary of domain terms, building domain lexicons, selecting and editing interpretation models, and editing concept hierarchies. For further references, see "KADS Power Tools: User Interface Specification" (Allen and Anjewierden, 1987), and "KADS Power Tools: User Guide" (Anjewierden and Allen, 1987), amongst other possible documents in

this respect.

We shall not concern ourselves with KADS Power Tools within the course of this work beyond the above description.

1.6. Global Remarks

We do not intend to use prototyping as the main activity for developing KBS. Prototyping can be applied in testing development stages within the methodology; this subject deserves a chapter to itself. We shall say more on this in the concluding chapter of this work.

We intend to extend KADS in the future to contain KBS shells. These shells are intended to contain generic analysis and design models for different classes of expertise. This is an extension of IM, so that a useful library of generic problem solving models can be established for the users of KADS methodology. In this respect we agree with Clancey (1985), that there are a number of general applications which exhaust most of useful expert activities (planning, monitoring, diagnosis, so on).

KADS intended KBS shells are based on existing general expert problem areas, providing a flexible environment whose every stage is explicit and documented. These, we believe, are the important tools the knowledge engineer will require in developing large and complex knowledge based systems.

1.7. Organisation of Chapters

The chapters will appear in the same order in which we have introduced various sections in this introductory chapter. We shall, however, precede the work with a literature survey, so as to compare our work within KADS with attempts of a similar

nature. Chapter 2 will provide this comparative study within a spectrum of methodologies or approaches which have been devised in the field of KBS of late.

We shall provide a detailed description of KADS analysis within the analysis lifecycle model in chapters 3 and 4, starting with the 'external view'. Chapter 5 will round off the work in analysis by introducing the notion of 'Interpretation Models' in some detail. These models are the cornerstone of the KADS methodology. In chapter 6 we shall provide an account of the use of conceptual models and interpretation models in the context of a real life application used as a case study in that chapter.

In chapter 7 we shall provide an account of KADS design, which uses the output from the analysis phase. The modelling within design does not as yet benefit from the same degree of completeness as that of analysis. Despite this we are witnessing a number of successes using our design approach, and we are encouraged that further research should yield a design description language of the same power as KADS analysis language. In chapter 8 we shall apply our design approach to a system called NEOMYCIN (see, for instance, Clancey 1985[b]) in a post-hoc study, in which the power of KADS analysis and design description languages are illustrated. This study is particularly useful, since NEOMYCIN shares some of MYCIN's background. MYCIN, in itself, is behind an approach to a KBS methodology in the shape of ROGET (see chapter 2), which has been a point of inspiration in developing KADS. NEOMYCIN also has a wide scientific appeal in the KBS community due to its large degree of system modularity, and its reusability within HERACLES which is a knowledge acquisition shell based on NEOMYCIN. The culmination of these factors presents NEOMYCIN as a suitable test-bed for KADS design approach.

In the concluding chapter, we shall provide a summary of our work, as well as indicating our plans for future work in continuation of the methodology. This chapter will have a significant part which deals with the use of prototyping, the inclusion of which

we find particularly apt. The use of prototyping within KADS is at a seminal stage; a description of it will thus be well placed in a chapter addressing future developments within KADS.

This thesis is not about producing an extensive system implementation, the length and breadth of which would have made up the content of our work here. An attempt of this type would have been futile, since it would have had little to bear on the substance and the reasons for introducing KADS as a methodology for KBS development. By concentrating on one large implementation, we should have been unable to show why KADS is applicable in general.

Appendices A and B will, however, provide examples of using KADS to support system implementation. Appendix A is a prototype based on an earlier case study (chapter 6). Whilst, appendix B will provide an account of how to go from 'requirements statements' to full 'system design' using KADS. The two appendices will complement the case studies (chapters 6 and 8) in showing how KADS can be used in developing knowledge based systems.

2. A Comparative Study

2.1. Introduction

In this chapter we shall compare KADS with a number of knowledge acquisition approaches. The candidate approaches are identified as using a number of techniques and methodologies which are comparable to parts of KADS. The aim of the study is as follows:

- To establish the position of KADS amongst methodologies for developing KBS.
- To show our reasons for believing that KADS is the most appropriate methodology for KBS development amongst those discussed in this chapter.
- To provide a framework for a contextual understanding of our work against the conventional view of KBS development.

We shall start by providing a brief history of transferring experiences gained in developing a system called MYCIN into a general purpose knowledge acquisition tool in the shape of ROGET (Bennett, 1983). The next stage will be to pursue this line of development within other application and non-application based knowledge acquisition tools. This line of enquiry should provide both the history and the tradition of such tools, as well as creating the appropriate context for a comparative study in the spirit that we have alluded to earlier.

2.2. Knowledge Acquisition - a brief history

There are a few examples of knowledge acquisition tools dating from the early days of expert system research. Teiresies (Davis, 1979) is probably the most notable of these tools, which is used to define MYCIN's knowledge base. Another tool derived from the MYCIN project is ROGET (cf. ibid.); ROGET can be seen as precursor of interpretation models in KADS. It has only one model which is derived from the structure of MYCIN. The model has, however, helped to recognise that there is a level of description of system / domain which is more abstract than the underlying system itself. The abstraction provides an important insight into the structure of the problem solving within the system. The structure might, then, be used to help to devise other systems showing similar problem solving behaviour.

The notion of generalising a problem solving structure by applying it to a number of domains is gaining acceptance increasingly within a large section of KBS community. Researchers within the community have shifted their focus of attention from building individual systems, to capturing generic systems applicable to categories of domains. The remainder of this chapter will address these systems with a view of comparing them against KADS.

Given the range of systems and structures in existence, it is of little use simply to list them. We shall require a general framework as a basis for comparison. The development of this framework is also part of achieving a better understanding of the KBS development process.

The framework provided here is very much in the spirit of our own work. The framework certainly is not intended to reflect the perspective of many of the researchers whose works are referenced.

2.3. Paradigms for Knowledge Acquisition

The conventional view of expert system development is one of extracting knowledge from a source of expertise and transferring it into a format which can be used in a computer system representation. The resulting computer program is known as an expert system, which is the artificial counterpart representation of some real life expertise. The source of expertise resides in an expert (or a small group of experts) and, possibly, documents pertaining to the associated domain of expertise. The inextricability and tacitness of certain elements of knowledge make them unsuitable for representation on a computer system. This is because what cannot be expressed clearly in the real world can certainly not be represented and used in computer programs. In such cases if the tacit elements are essential to problem solving, then a graceful degradation will take place in automating the expertise. When tacit knowledge elements are not central to problem solving, they may be side stepped at the risk of lower system functionality or performance.

A more general view is that regardless of the exemplification of the expertise in the functioning of an individual, knowledge about a domain exists in the world (perhaps in a distributed form). It can be said further that this knowledge may be codified and the associated behaviour produced in a system. Thus the paradigm is one of modelling an aspect of the 'real' world and reproducing certain behaviour or synthesising the desired behaviour (which may not be actually found at the present) in an 'artificial' world.

The use of 'real' and 'artificial' here has much in common with Simon's view (Simon, 1969) and his characterisation of "enginering the artificial" seems consistent with our suggestion about KBS development, albeit his focus of attention is elsewhere. We believe, in the same way, that the pairing between the engineered artifact and its real life counterpart does not need to maintain all of the features of the latter in the former. We are interested in capturing those features of the 'real world' which are seen as

essential to solving problems within a computer system. This may also result in synthesising certain features in the artifact which cannot be observed in the 'real world' counterpart, in order to achieve the required problem solving behaviour. The use of a modelling language such as KADS is to facilitate a mapping from the 'real world' onto the domain of 'the artificial'. We are not, however, advocating modelling to the exclusion of elicitation from experts. Experts should be used whenever required and possible, but the use of data should be conceived as a foundation for modelling.

An important consequence of making this generalisation is that knowledge acquisition has to be viewed with a *modelling* metaphor, rather than a metaphor of extraction or mining. The modelling, as pointed out earlier, will provide the intermediate stage of mapping from a 'real' into an 'artificial' world. This will also remove many of the psychological and practical problems of knowledge acquisition if seen as extracting something from the expert. The emphasis on expert systems as critically involved with modelling has been forcefully argued by Clancey; although, his emphasis is on the systems produced and ours is rather on the production process.

The distinction is in the ability of using KADS modelling independently of any one system or developer to create a problem solving model. Clancey's approach is based on applying abstraction of systems already developed as models for creating new systems. This can be observed in the use of HERACLES which is a diagnostic shell abstracted from NEOMYCIN. HERACLES is intended to be used for modelling diagnostics in various domains, and not just in medicine as is the case for NEOMYCIN. KADS, on the other hand, will provide the vocabulary (see chapters 5 and 6) for a user to create her own generic model for different domains. Whilst KADS will learn from works such as HERACLES project, it will not contain the creation of such models to centres of excellence. KADS will also provide a forum for criticising aspects of a model within a methodology, incorporating a consensus view as far as

practically possible.

If one accepts this view of KBS creation at the most general level as a mapping from an understanding of behaviour in the real world to the description of the form of an artifact (i.e. as a category of engineering), then there are essentially two ways in which such a mapping may be created. One may start with known components of techniques in the artificial world and experiment with composing them to create the desired behaviour. Examples of this are not common but include chemical synthesis on an exploratory basis and exploratory architecture. The alternative approach is to devise a modelling language within which one can describe the required functionality and transform this description into the desired artifact. There may of course be a number of languages and thus transformations involved in this process. This approach is very common, obvious examples are in civil engineering and electronic circuit design. A very clear abstract description of this view of software development is provided by Maibaum(1986).

Supporting the former process lies outside our scope but can be seen as the function of tools such as KEE and LOOPS. Very few of conceptual tools fall into this category with the notable exception of KREME (Abbret & Burstein, 1986). These development tools will have within them a number of primitives, such as 'objects', 'rules', and inference mechanisms for developing and combining aspects of the artifact which may result in the required behaviour expected of a computer program. The question which then arises is how can the diversity of other works be located within the paradigm of development via model building.

One categorisation may be based on the distinction between the top-down approach to analysis and a bottom-up approach to it. Thus approaches which presuppose a model structure (or several alternatives) will then attempt to fit observed data within such a structure. On the other hand, tools which support one or more analytical techniques

make no commitment to structure in advance. Worse still, they will provide less support for problem understanding, as well as generating weaker structures. The top-down versus bottom-up categorisation, which is well known for all types of software activity, hides a more fundamental and more interesting distinction between the research programmes behind the various pieces of work. In a bottom-up approach the tools are of a general nature, and their use is guided by their capability to represent and manipulate aspects of knowledge or expertise. The wider, and more heterogeneous the population of such tools, the greater expertise domains they are likely to support. At the same time, there is no guarantee that they would provide the analytical tools for a new domain, since there is no a priori commitment , on their part, to the structure (or model) of that domain.

We have already outlined the notion of KBS development as mapping the real world onto the artificial world via a model. This perspective can be extended across KBS development as a whole by considering a description space which includes all potential real world models, and a second space containing all system models. A specific system development process will, then, consist of constructing a description in the first space, and mapping it onto a description in the second space, and from there to an implementation. We shall refer to the first as the *analysis* space, and to the second as the *design* space. The major issue in knowledge acquisition research *from an engineering perspective* can then be seen as attempting to understand the structure of these spaces and how points within them may be defined. The definition will require both appropriate languages, and effective processes within those languages; the two issues whilst closely related are nevertheless distinct. Research programmes tackling these issues may be seen in three categories.

The first provides techniques for concentrating on a very limited number of dimensions but is reasonably precise in handling those dimensions. A good example is repretory

grid analysis (cf. Gains & Shaw, 1986). Such techniques may provide relatively simple and well defined methods of analysis. Their drawback is that there are no clear criteria for when they are applicable and how to use the subsequent results. This can be seen as a consequence of their failure to shed light on the structure of the description spaces or any sense of location within these spaces.

The second strategy is to fix a point within the design space by selecting a particular implemented system. This would imply creating a definition of the mapping from the analysis space by building a knowledge acquisition tool, which enables the original system to be refined or augmented. The next step would then be to explore how the knowledge acquisition tool may be used to build what are believed to be similar systems, i.e. in related domains. This corresponds with perturbing the domain of the mapping and using the information gained as a means of better understanding the description spaces. The difficulty with this approach is that the number of data points currently available is very small and the creation of new ones is laborious. There is also the risk that in perturbing the domain to another intuitively believed to be close, one may misjudge the metric properties of the space. This would in turn mean that one is dealing with a domain in fact very different from the original one.

The third approach is to tackle the issue of characterising the languages adequate to describe a significant number of points within these spaces. The drawback of this approach is that it is very difficult to produce such a language and once one is proposed it is difficult to evaluate its adequacy. While there are some proposals relevant to the analysis space, there is a dearth of material relevant to the design space. As a rare exception to this, Newell's classic characterisation of weak problem solving methods (Newell, 1969) can be named. More recent work on reflexive systems (e.g. Maes 1986) and object-oriented systems (e.g. Stefik & Bobrow 1986; Booch, 1986) may contribute to a better characterisation of the design space, but at present we are

far from having a coherent view.

At this point in time the second and third approaches are highly complementary, although they appear to be very different. One hope in providing a characterisation of the research issues which encapsulates both is that more researchers will be tempted to consider their results within this framework thus adding to the data available. It is a common complaint that AI research results are frequently non-comparable and thus scientifically of little value.

Looking at the state of this research area overall one main conclusion must be that more systems are required. The larger population of systems will only be seriously useful, if closer commonality of description for those systems is adopted. The fact that we can perceive a framework within which previously apparently disparate works can be related should, nevertheless, encourage us greatly. We are beginning to take the step from producing interesting or curious single results to gathering a body of scientific data on systems.

The remainder of this chapter is devoted to categorising works within the framework proposed here (above), and comparing their results to KADS approach at a more detailed level. We begin with considering what we term "single-model" systems, which we see as exemplifying the second strategy outlined above. We then turn to language based approaches in the third strategy. As we have previously indicated, we do not see the first strategy as contributing to our understanding of the issues at the level at which we are pursuing them. It will, therefore, not be considered further in here.

2.4. Single Model Systems

2.4.1. ROGET

An attempt has been made to generalise very widely from MYCIN within a system known as ROGET. This had been coloured by the perspective of the time, in the same way that EMYCIN had been thought to be a very general purpose tool. This is due to the fact that MYCIN's structure is very limiting and thus ROGET in practice must be seen as single-model based. The philosophy of the endeavour has, nevertheless, been to devise a "multi-model" based system, and the objectives of ROGET have much in common with later work in this paradigm.

ROGET has a knowledge base of "conceptual structures" of existing expert systems. A "conceptual structure" is an abstract description of the structure of an expert system. It consists of types of data, types of inferences, etc., comparable to interpretation models. For example, the conceptual structure called "recommend action to fix problem" consists of determined actions, determined causes, determined problems and evaluated evidence. Some relations are also specified, e.g. "evaluated evidence" determines "determined problems" and "recommended actions" and "determined causes". Brief descriptions and examples are available of the categories in the conceptual structure. Many categories have got subcategories (e.g. laboratory tests are a subcategory of evaluated evidence) as well.

The program helps the user to select an appropriate conceptual structure from the library of such structures, by asking questions about the current task and domain and presenting descriptions of the structures in the library. In the next next stage the structures are edited (by adding and deleting categories), and then the categories are used as frames to construct the actual knowledge base.

The second function of ROGET is to provide the user with practical advice. For example, ROGET contains heuristic rules about the feasibility of systems (taking into

account the complexity of the domain and the conceptual structure, the experience of the knowledge engineer, etc.).

The second function of ROGET is to "compile" the resulting knowledge base into an expert system, by translating the concepts and rules into a shell. ROGET currently can translate simple concepts and rules into EMYCIN.

ROGET derives its power from the fact that: an abstract description of an existing expert system (similar to the new system), can be used to establish a coherent framework for the new domain. The idea of ROGET is very similar to that of KADS. A study of the prototype version of ROGET will reveal that it is still a very limited system. It can only recognise domains that are very similar to MYCIN. For new domains, it will be difficult to see how well the conceptual structure of MYCIN applies. ROGET focuses on the structure of objects and disregards inference methods and strategies (a bias that may stem from MYCIN). The advice on knowledge engineering is very ad hoc and seems to be based on practical experience in the past with a limited number of systems.

2.4.2. MOLE (Eshelman et al., 1986)

This system is a good example of the single model strategy. As its authors express "MOLE the knowledge acquisition tool gets its power from its knowledge of the problem-solving method of MOLE the performance system ... MOLE's problem solving method is a variant of heuristic classification". Since heuristic classification is a very general method, MOLE may be expected to have wide applicability. The penalty for this generality, however, is lack of expressive capability. The authors note "MOLE's method still places strong limitations on the type of tasks for which it would be appropriate." MOLE requires exhaustively specified symptoms and hypotheses. It then requests knowledge which may explain the symptoms ("covering knowledge"),

and knowledge to enable differentiation between hypotheses ("differentiating-knowledge"). The two types of knowledge are then used to build a network of associations between symptoms and hypotheses. This view of the use of knowledge in refinement of a knowledge base appears very similar to SEEK (Politakis & Weiss, 1984; Ginsberg et al., 1985).

A weakness of the MOLE approach is that it focuses on gathering associations rather than states or objects. Thus if an association requires the definition of an intermediate possibility as a basis for discrimination, then the system has no means of detecting this or probing for the knowledge. As the authors note: "since its constructive ability is rudimentary, we have continued to present MOLE as only appropriate to tasks that are amenable to heuristic classification." We would interpret this as saying that the chain of associations between any given symptom and hypothesis is short and only loosely linked to other chains. To claim anything beyond this would imply a move from heuristic classification to fuller causal modelling.

2.4.3. STUDENT (Gale, 1986)

This system lies at the other end of the spectrum from MOLE in the sense that MOLE uses a general but weak problem solving model, while STUDENT uses a very specific model - for the domain of statistical data analysis. The author refers to the method as knowledge-based knowledge acquisition. MOLE's knowledge is general and concerned with how hypotheses and symptoms must be related, and contains heuristics such as parsimony of explanation of associations. Meanwhile STUDENT's knowledge is solely about data analysis. The critical point is that the knowledge in the knowledge acquisition system must be more general than that required in the target system, i.e. (as Gale notes) the tool must be useful to build more than one system.

Gale explicitly identifies the core of the knowledge based knowledge acquisition system as a "conceptual framework for the domain". This appears precisely equivalent to an interpretation model. However Gale provides no formalism for describing such a "conceptual framework" and it is effectively implicit in the STUDENT system. A second major issue is how the conceptual framework is derived, Gale suggests the framework to start shaping from the first instance of a system in a domain. We certainly believe that examination of implemented systems provides useful data for creating interpretation models. We do not see this as a prerequisite in the way suggested in STUDENT, the overhead and lead-time is simply too great. We would however suggest that our relative freedom is acquired as a result of having a descriptive formalism applicable to all the models. This is the consequence of the language-based research programme rather than the model-exploration one.

2.4.4. KNACK (Klinker et al., 1986)

This is another specific domain system used to construct target systems capable of evaluating the design of electro-mechanical systems using a particular reporting format. The systems created (called WRINGERs) gather information about a design, point out possible design flaws and make suggestions to correct and improve the design. The model (in our terminology) is referred to by the authors of KNACK as the problem solving methods together with the identified knowledge roles. These methods are described explicitly, albeit informally, in a way which is closely akin to a task structure referencing a set of metaclasses or knowledge sources. The following are given as "roles" (for us metaclasses): report structure, synonym, information selection, design fix; and as categories of knowledge (knowledge sources): information identification, information gathering, consistency evaluation, completeness evaluation, design evaluation and design default. It is not difficult to imagine that with a little more data one could provide a KADS four layer model for this domain.

2.4.5. Other Systems

The TKAW system (Kahn et al., 1986) is similar to MOLE; its focus is, however, somewhat narrower. This is because it handles domains concerned with the diagnosis of equipment failure, in which failures can be represented in an abstraction hierarchy. This would appear to allow support for more complex chains of causal connections than MOLE provides. TKAW thus represents a point on the spectrum somewhat towards STUDENT.

SALT (MARCUS, 1986) acquires knowledge for construction tasks which are based on planned sequences of steps which can be created on a propose and revise basis. This covers at least configuration and scheduling tasks, but with the proviso that the relationship between "components" can be described in relatively simple dependency networks. Thus the domain layer is represented primarily as associations comparable in form with those in MOLE. The knowledge at the inference level is concerned with how constraints interact and how to fix inconsistencies. The task level is 'plan creation' using the 'propose and revise' strategy.

The systems described so far cover not only the static knowledge about a domain but also the knowledge about how this can be used in problem solving. A number of systems tackle the more limited issue of developing the static knowledge but within an assumed overall model. Thus OPAL (Musen et al., 1986) supports the acquisition of domain level knowledge for ONCOCIN. BLIP (Morik, 1986) also operates at the domain level but seeks to refine and extend an initial model ("a sloppy model") using a limited amount of meta-level data. Whereas OPAL is highly specific to the ONCOCIN knowledge base, BLIP is entirely general. The generality is manifested in the user being able to start by specifying a model for the domain of interest, which BLIP can then refine. It seems more natural to treat BLIP as a single model system. BLIP with a 'sloppy' model is a single model system and it is only this combination which

constitutes a knowledge acquisition system. BLIP without a sloppy model is essentially first order predicate calculus with a small amount of very general meta-knowledge. While one might argue that predicate calculus is adequate to characterise points in the analysis space, it certainly does not provide any support to the knowledge engineer in the analysis stage. It, therefore, cannot be considered a 'modelling language' in any practical sense.

This characterisation of BLIP in terms of the framework outlined above does not seem entirely satisfactory. This may be due to the fact that logic as a language does not fit conveniently with the more specific 'language' described in the next section. It may be because learning systems (or systems incorporating automated refinement of knowledge) cannot be characterised in the framework given here. This issue requires further elaboration.

2.5. Language based Approaches

2.5.1. KRITON (Diederick, Ruhman & May, 1986)

KRITON is a knowledge acquisition system designed to support the process of protocol analysis, and the conversion of this data via an intermediate representation into a knowledge base. The objectives are very similar to KADS, although in practice more emphasis seems to have been placed on automated elicitation and less on supporting analysis. The intermediate knowledge representation level is described as having two layers: "a descriptive language for functional and physical objects, representing the generic concepts, and a propositional calculus representing the transformation path of those concepts during the human problem solving process." The first is clearly equivalent to the domain layer in KADS, but it is not clear whether the second is an amalgamation of inference and task layers or only the inference layer. The latter

possibility is implied by the description of the "propositional calculus" as using "semantic primitives to describe the basic relations of concepts detected by protocol analysis." Unfortunately the referenced paper does not contain any further specification or illustration of these "semantic primitives". The KRITON approach differs from KADS in that there is no notion of generic models and in that it is assumed that the intermediate knowledge representation can be transformed directly into the knowledge base. Thus there is no explicit acknowledgement of a design process or of a space of design descriptions.

2.5.2. Expertise Specification

The work of Johnson (Johnson & Gruber, 1986) is of great interest to us because his philosophy is strongly in tune with ours in his belief in modelling expertise using protocols and avoiding early commitment to implementation. As he says: "we believe that building a prototype system early in the knowledge acquisition process may carry with it commitment to specific model of thinking (inference process) that does not adequately represent the expertise we are trying to understand". He also provides a well specified language for modelling expertise, but (at least superficially) it appears totally different from ours. Johnson's language is single level and provides the following constructs:

bubble

context of problem solving,

arrow

directed relations between bubbles; the " "pathways" of problem solving determining a way of moving between, components of the solution",

triangle

the set of abilities deemed necessary to move one context to another,

cloud

"identifying the goals that specify the abilities needed to "travel" on that relationship or "pathway"",

box

"representing the set of possible "triggers" to activate the abilities (goals)".

This descriptive language is applied directly in analysis of protocols. A protocol is coded into episodes and then contexts are formulated. In addition to the constructs defined above a "sequencer" is defined. The sequencer defines the primary structure of the problem solving in temporal sense, i.e. how the problem solving is organised at what we would call the strategic level. The contexts and the sequencer define a temporal division of the protocol, and the episodes, which fall within each temporal interval, are then analysed to determine the relationship and its associated properties (abilities, goals and triggers).

This scheme is obviously syntactically totally different from KADS and although there is an apparent relationship between at least some of the semantic elements, it is far from obvious that one scheme could be mapped onto the other. This doubt is increased by noting that Johnson's language basically describes *states* of the problem solver and the transitions between them. There is no attempt to model the *structure* of the problem solver as in KADS. Thus Johnson's model may be a 'purer' real world model than that provided by KADS, since there is less specification of internal structure and thus less commitment to implementation. This is supported by the fact that the language is applied directly to protocols whereas in KADS the model is further removed (abstracted) from the description of specific data. It would also appear to us that it is less obvious how such a representation might be converted to an implemented

system than in the case of KADS model. This would also seem to be the implication of Johnson's statement "At this point, we claim only that our representation can serve as an *initial* specification for a computational model of expertise" [emphasis added]. However we do not wish to see these approaches as competing in the sense that they are mutually exclusive. The fact that we find another proposal so much in tune with ours while quite different in detail appears to us as a strength. The fundamental argument is in favour of a structured approach to analysis, not in the parochial support of one particular model.

2.5.3. Generic Tasks

The work of the group under Chandrasekaran at Ohio State University is focussed on the definition of "generic tasks" which are claimed to be at the "right" level of abstraction to support effective knowledge acquisition (Bylander & Chandrasekaran, 1986). Within our framework the first question is whether a "generic task" is a point in the analysis description space or in the design space. Given the more detailed characterisation of a generic task as "an elementary generic contribution of a task, representing an inference strategy about concepts" one might assume that it was in the design space. However the fact that a problem is envisaged to map directly onto a generic task (or combination thereof) could suggest the former. If we look at the rationale for introducing generic tasks the situation becomes clearer. Bylander (cf. ibid.) notes the "interaction problem", i.e. the interaction of decisions about knowledge representation and control, and suggests that generic tasks are a way round this problem by fixing primitive combinations of the two. Knowledge acquisition is then performed in the context of the generic task and indeed CSRL (Bylander & Mittal, 1986) is proposed as a language specific to the description of instances of a single generic task. Thus in our terminology a generic task is in fact a region in a design space but with the assumption that it can be fixed as the range of a region in the analysis space. Thus a generic

task in fact defines a fixed combination of points in the analysis space and in the design space - hence the ambiguity referred to earlier.

We suggest therefore that there is a fundamental difference between this strategy and the KADS approach. Rather than exploiting the interaction problem, we attempt to sidestep it by providing different representations at different stages of development process. Which approach is more fruitful only time will tell, but we do have some reservations about the pragmatics of the generic task approach. Firstly it is not obvious that generic tasks are at the right level of abstraction. It is odd, to say the least, to refer to "elementary generic combinations": if the entities are combinations then their components are more primitive (elementary). It is being suggested that other combinations of these components are logically impossible, this seems unlikely. Surely the components and the manner of their configuration requires deeper analysis. We may also ask at what level of abstraction do generic tasks lie, relative to the entities in the KADS four layer model (see chapter 4). Since it is suggested that an application system may be the instantiation of a generic task, it seems likely that a generic task is analogous to an interpretation model, or at least the inference structure. On the other hand Bylander and Mittal (cf. ibid.) suggest their notion of classification is more primitive than Clancey's heuristic classification (Clancey, 1985), which may be decomposed into three elements. This would make the generic task analogous to a single knowledge source, but it could hardly constitute the basis for a complete application system.

This leads to a second problem. It is envisaged that application systems may be made up of more than one generic task, but since generic tasks are highly discrete (given they are combinations of representations and control) it is not clear how the interaction in such composite systems would be defined. This problem is greatly exacerbated if one defines a different *language* for each generic task. This would seem precisely the

wrong level at which to define a language: one surely requires (or at least prefers) a language which is uniform across all the components of a system.

We suspect that at least some of these difficulties arise from trying to solve the knowledge analysis problem (i.e. contamination of analysis by presuppositions about representation) within the context of implementation paradigms. Our suggestion would be to back away from implementation altogether and recognise a distinction between analysis and design / implementation (as does Johnson). This creates the difficulty of mapping the analysis to design but we find that a more tractable research strategy than that adopted at Ohio state.

2.5.4. Ontological Analysis

This technique is analogous to the domain level tools described above, in that a language is provided to specify objects within a domain. The ontology of the domain is a specification of the objects in a domain, which Alexander et al. (1986) define in three parts:

static ontology

"defines primitive objects, their properties and relations,"

dynamic ontology

"defines the state space of the problem solving domain, and the actions that transform the problem from one state to another state,"

epistemic ontology

"defines the constraints and methods that control the use of knowledge applied to the static and dynamic ontologies."

These ontologies are defined in a language (SUPE-SPOONS) which is based on the

domain equations of denotational semantics and algebraic specification.

While the epistemic ontology may overlap with the inference level of the KADS model, this language seems essentially to define objects at the domain level. As the authors note:

> There is no operational component identified. The operations on a type are necessary for fully specifying a type. We have no way to define the behaviour of an object, only its structure.

Thus the descriptive capability of this language is admittedly limited. However the domain level of the KADS model has so far been very partially described and explored and it may be that the two frameworks can be combined with little difficulty. We hope to assess this further in future experimentation.

2.6. Conclusion: KADS vs the Rest

The primary purpose of this chapter has been to propose a framework for describing and discussing knowledge acquisition methods and tools and to relate specific cases to this framework. However in evaluation of our own approach we should summerise the major distinctions (and similarities) between KADS and other systems/techniques.

Firstly, as has been emphasised, KADS is a language based approach, in the sense that it provides a vocabulary and grammar, and the other constituent parts as described in chapter 1 (above). It makes some commitment to internal system behaviour, particularly at the upper layers of the four layer model. Thus it could be viewed as a pure behavioural specification. However, some compromise in this regard seems necessary in order to be able to use the model as a basis for subsequent design, i.e. the design is partially constrained at the stage of conceptualising about the real world.

The structure of the KADS model is a notable feature. While one could produce other formalisms possibly as adequate for modelling expertise, KADS provides a natural partioning between knowledge elements (see, for instance, the four layer model in chap. 4), when compared with other systems/approaches.

One component of the KADS approach which we have not found elsewhere is the provision of "interpretation models" to support the analysis of data from knowledge elicitation. The 'single model systems' each embody (more or less explicitly) a model which in fact performs this function. The attraction of the KADS approach - from the point of view of the general system developer - is that it combines the strength of a single model with the generality of the language based approach. Clearly the adequacy of utility of these general models needs testing, but we now believe that we have a sufficient number of these models (cf. Breuker et al., UvA & Davoodi et al., STC, 1987) to enable such an assessment.

3. REQUIREMENTS ANALYSIS

3.1. Introduction

The integration of KBS into real life commercial applications requires a large degree of conformity with the overall aims and constraints of the user environment. The degree with which such systems will become common place in commercial applications, as well as those of academia, is a measure of how successfully they can adapt themselves to those environments. It is therefore essential to consider and capture all of the requirements expected of the intended system in the early stages of a project lifecycle. This is the task of requirements engineering in which a requirements analysis is carried out and its outcome recorded in a number of documents. The requirement documents should ultimately filter into a number of statements describing constraints to be placed on the design of the system.

Requirements analysis provides the context (or shell) for the detailed analysis phase in which the expertise behaviour and the user requirements are mapped out prior to design. Our concern in this chapter is the overall requirements analysis, references to those parts of the analysis phase concerned with the expert behaviour are made in order to demonstrate the relation between the two activities (user and system perspectives) within the total analysis Lifecycle Model (hereafter also referred to as LCM) (fig 3-3).

3.2. Requirements Analysis

We distinguish two views of a KBS, one the functional view describing the 'internal' working of the intended automata modelling the expert or some domain knowledge

and, the other, the external constraints placed upon the working of the automata. We shall refer to these two as the *internal* and *external* views respectively. Requirements analysis is concerned with defining the external view, treating the internal view as a black box sufficiently well behaved to support the external requirements.

Consider the capturing of an expert knowledge in the automata, where the expert's function is to give advice on possible types of computer hardware configurations. As part of the external requirements the KBS simulating the expert may have to reside on a particular machine for a host of reasons such as: compatibility, security and cost. These constraints all belong in the external view, and have nothing to do with the way the problem solving behaviour of the expert will have to be modelled at the analysis stage. In devising a system to perform the hardware configuration, we shall, however, need to ensure that the machine on which the KBS is to be implemented is capable of supporting the system. There will, therefore, need to be a process of negotiation taking place between what is externally required, and what is technologically possible. We shall consider this point more explicitly in the context of the analysis global LCM.

3.3. Analysis, Global Life Cycle Model

Four global activities can be identified in the analysis phase of a KBS project (cf. Barthelemy, et.al., sec. 3, 1987). These are:

(1) *Determine Scope of Project*

the initial activity in which proposals are submitted for consideration, in which global requirements and resourcing issues are also introduced.

(2) *Requirements Analysis*

the external view is identified and documented in this activity. The constraints

introduced by the view should affect the design of automata such that it adapts closely to the user community's needs.

(3) *Knowledge Acquisition*

the expert problem solving capability, and its extension in terms of mode of interaction with users, also known as *modality* are modelled in this activity. The model documents, in turn, represent the internal view.

(4) *Feasibility Estimate*

The information gathered and modelled during the requirement and knowledge acquisition activities needs to be assessed to ensure its validity, as well as establishing that the scope of project is feasible. This process will give rise to the feasibility estimate, upon whose outcome the scope of the project may or may not have to be revised and renegotiated.

The sequencing of the four activities and possible negotiation and iterations between them give rise to the analysis global life cycle model (fig. 3-1).

It is interesting to note that the knowledge acquisition phase is normally driven by *technology push* in terms of what the latest AI and KBS technologies are capable of supporting, with the possibility of those techniques *pushing* the frontiers of the possibilities. On the other hand, the requirements analysis is driven by *application pull* ensuring that the technology available can support the external requirements. Often, the negotiation between the two views brings about a compromise which is suitably technical as well as being capable of supporting the user community.

The nature of the external and the internal views are primarily decided by the business, commercial, industrial, or academic needs of the potential users. The needs themselves are translations of the users' corporate strategy in terms of where they aim to be in the future, and how they perceive achieving it. The analyst or a team of analysts in

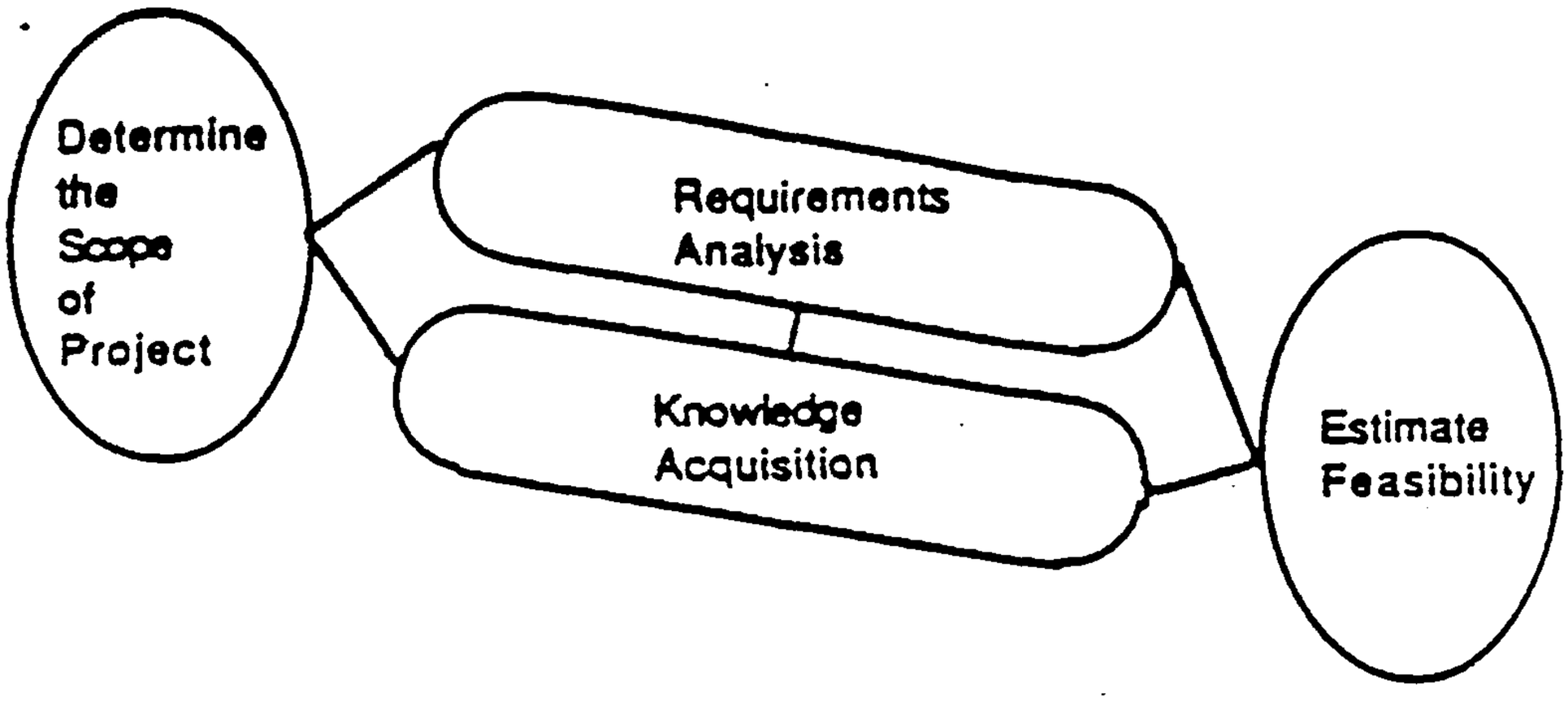

Figure 3-1: *Analysis Global LCM*

the shape of knowledge cum requirements engineers should establish in conjunction with the users and experts the *actual* process and possibilities of achieving the users' objectives. The analysis phase, clearly, is the place where the nature of this process, in terms of system requirements, is identified. The outcome of the analysis, for instance, might indicate that the internal view is a composite one calling for conventional database management system (dbms), and decision support system (dss), as well as KBS. This can be accommodated for within a similar LCM with an expanded internal view (fig. 3-2).

3.4. Analysis, Detailed Life Cycle Model

The analysis of the external view consists of three activities of:

(1) *Analyse Present Situation*

during this activity a good understanding of the user environment in which the intended system is to perform should be attained.

(2) *Analyse objectives and constraints*

the role of the perspective system is assessed in this activity, in terms of the

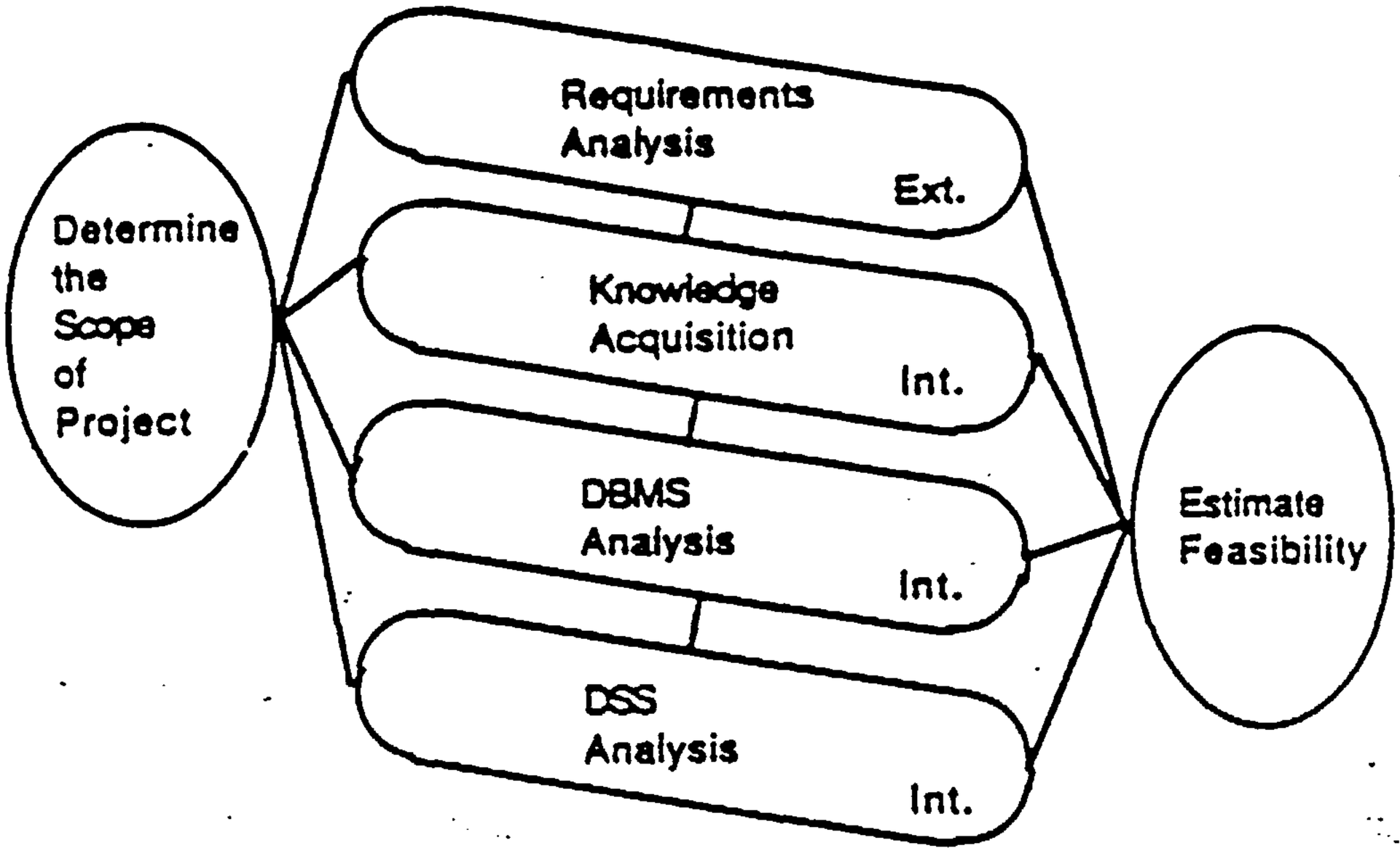

Figure 3-2: *Analysis LCM, with a compound Internal View*

objectives it sets out to satisfy and constraints placed upon it.

(3) *Determine Functional Requirements*

an overall consolidation of the external and internal views will have to be arrived at; this is achieved within this activity.

The analysis of the internal view: capturing the expert(s) behaviour, in turn, will divide into three activities of:

(1) *Analyse Static Knowledge*

the analysis of concepts and relations (see chap. 4) pertaining to the domain of expertise will take place in here.

(2) *Analyse Objectives and Constraints*

the analysis of the expert problem solving behaviour, and the intended mode of communication between the prospective system and users is the concern of this

activity.

(3) *Construct Conceptual Model*

a four layer model (see chap. 4) is constructed in here, in which expert knowledge is described in terms of elements of knowledge, relations between them, and meta-levels of manipulating that knowledge to achieve the problem solving behaviour.

Out of the eight activities we have identified, two of them are concerned with global issues of the scope and feasibility of project, and the other six capture the internal and external views. Varying emphasis is placed on these activities in different projects; on the other hand, some of them might be absent in certain applications. For instance, a project conducted in an academic setting may pay little or no attention to the external view, and be directly concerned with the problem solving technology.

In devising an analysis lifecycle model we have taken the view that all of the phases are always present by default. A less desirable solution would have been to have different LCMs devised for different projects. The latter approach will leave the LCM open to unhelpful interpretation, and make the analysis less sharable and transparent; whereas, in the former case phases not present can be ignored. The order in which the activities are traversed are shown in figure 3-3 , the order is a tentative one and can vary should it suit a particular project's needs. The LCM follows Jackson Structured Design (JSD) diagramming techniques (Sutcliffe, 1988), in which each activity results in a number of documents. The documents may be used as input to other activities as indicated by the arrows. The order of activities does not necessarily imply time, that is some of the activities might take place concurrently. Equally, an activity upstream of the LCM may be suspended, until data or knowledge needed for the resumption of it is gathered downstream of the LCM.

The names of the documents generated in each activity have been abbreviated according to the following scheme:

P[n]:

"P" stands for project document required for project management, and suffix "n" is used to distinguish between project documents introduced at different layers within the phases. We shall be using suffix "n" in the same way throughout for other documents generated in the analysis phase (see below).

R[n]:

Requirements document will contain a description of the external constraints placed on the prospective KBS (or hybrid system, such as an integration of KBS and dbms). The KBS itself is treated as a black box of which a number of functionalities are expected as part of the overall requirements. The document uses a descriptive, natural language format in outlining the requirements. The document may, therefore, suffer from all senses of vagueness and ambiguity normally associated with non-formal languages. It is possible to interleave the natural language description with formal notations such as Vienna Development Method; we, however, find external requirements so informal and diverse in nature from one organisation to another as to make it difficult not to use natural language in some form to capture it.

M[n]:

Model document will describe the requirements for the internal view, be it a KBS or an integrated system. The modelling language, as far as KBS is concerned, is a prescriptive one using specific vocabulary and modelling stages. The language will be described in detail in chapter 4. The model document should describe fully all aspects of the intended system, in terms of capturing human expertise and the mode of interaction with users.

F[n]:

Feasibility document will provide an assessment of whether the required system is feasible, as well as describing the process by which the feasibility estimation is reached. This document will be the basis for negotiating the scope of a project.

There is, additionally, 'support document' containing interview transcripts, and background information which may need to be recalled at any point during or after a project completion.

In the next section we shall provide a brief description of each activity within the analysis LCM (see fig. 3-2), together with the associated documents. We shall use the document abbreviations, already introduced, throughout our description.

3.5. Description of Activities within the Analysis LCM

3.5.1. Determine Scope of Project

This activity may take place either at the point of inception of a project, or at intermediate cycles of renegotiation as a result of the feasibility estimate (see fig. 3-2). The proposal submitted in this phase should establish the boundaries of the overall requirements, as well as identifying the man and machine resources required to carry out a project.

3.5.2. Generated Documents

3.5.2.1 P1, Background and Prerequisites

This document will act as a common source of pre-project documents, such as all of the abstracts of meetings, and documents that give rise to the project. P1 will provide

information required by P2 and part of P3 (i.e. P3.1.).

3.5.2.2 P2, Project Terms and Directive

The objectives, aims, direction, extent and limits, of the intended system as well as its connection with other possible systems should be described here.

3.5.2.3 P3

This will consist of three documents, namely:

P3.1

Project LCM: careful attention should be paid to devising an LCM for the project at hand, in order for it to correspond closely with the needs of the project. The LCM need not unnecessarily contain all of the activities depicted in figure 3-3 , nor need it follow quite the same sequence of events. The point to consider is that the general LCM should be used as a guideline, and not as a definitive. The more emphasis is placed on technology push, the more prominent will become the 'modelling' activity. Conversely, if emphasis is placed on organisational and user needs, then the external view will require a deeper analysis.

On the whole, a number of decisions taken during this activity, in order to define what documents and activities are required during the analysis phase together with reasons behind those decisions will have to be registered in document P3.1.

P3.2

Project Plans: overall project plans in terms of allocating time schedules against activities, project staff, and users should be documented here. This document will provide the blue-print for project milestones as well as ensuring that the activities of project staff and users will be suitably coupled in terms of interviews and analysis of transcripts.

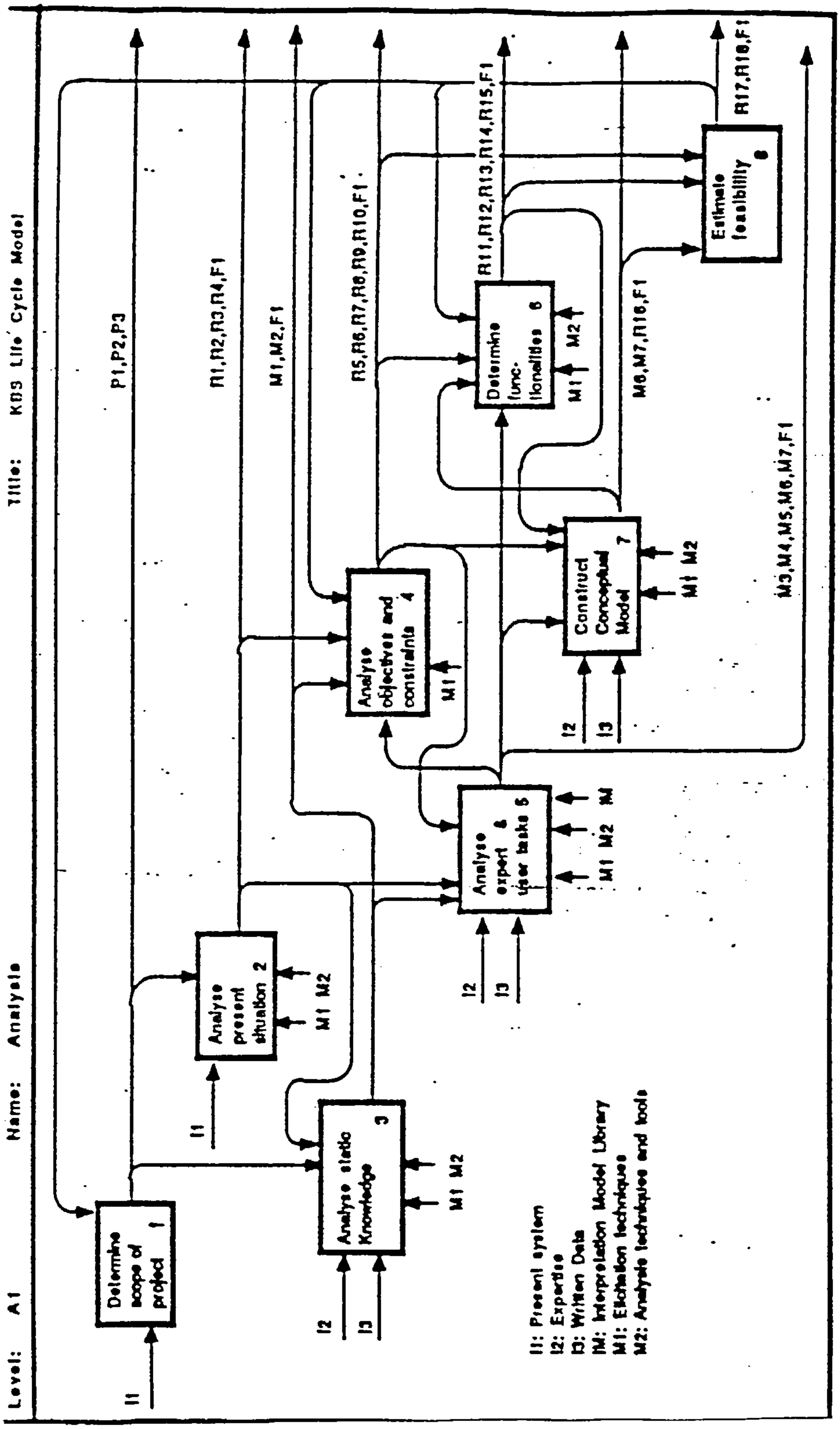

Figure 3-3: *Analysis Detailed LCM*

P3.3

Project Organisation: project staff and users together with their names, roles, and positions will be documented here. Some or all of a number of project groups such as: project steering group (PSG), reference groups, project management group, and working groups will need to be identified here with their respective responsibilities to / for other project members.

3.5.3. Analyse Present Situation

This is the first activity concerned with the external view, in which an overview of the organisation concerned will be attained. This is performed by interviewing managers and some of the potential users in order to establish the objectives and problems in achieving them. The extent to which those objectives can be addressed by the intended system is limited by the scope of the project.

3.5.3.1 Documents Generated

3.5.3.2 R1, Model of Present Situation

The structure of activities known as functional organisation, and hierarchy of people responsible for those activities known as formal organisation should be documented here. A number of diagramming and modelling techniques such as "Jackson Structured Design" (JSD, see, for instance, Jackson, 1975) and entity relationship may be used here, but their use will depend on the suitability of the technique to the domain at hand.

3.5.3.3 R2, Functioning Objectives of User Organisation

The objectives documented here will fall into two broad areas of "measurable," and "non measurable". The former will have some numerical value qualifying it such as:

"increase production by 40%". The latter, however, is in the form of a statement of desire in terms of future enhanced capability, such as "being able to respond efficiently to mangement requests"

3.5.3.4 R3, Functioning Problems of User Organisation

This document will contain all of the problems perceived from the users' point of view in achieving those objectives described in R2.

3.5.3.5 R4, Task Organisation:

This document will list all of those tasks which will need expert knowledge to solve. The document can also be used to register those tasks which will require conventional data processing skills, should we be aiming for an integrated system solution.

3.5.3.6 F1, Feasibility Estimate

This, the first feasibility document in the series of such documents will be concerned with whether:

- a system is needed, i.e. if there are *not* fulfilled objectives in R2 and problems in R3 that a system can solve,
- a KBS is needed, i.e if there are tasks in R4 requiring fifth generation technology,
- the problem domain is too large or too small,
- the resources are sufficient.

3.5.4. Analyse Static Knowledge

This is the first activity concerned with the internal view, in which all the concepts, relations and structures making up the domain of expertise (cf. chap. 4) will be gathered. In order to gather the static (domain) knowledge, interviews with the expert(s) will have to be conducted in which a number of structured elicitation techniques will be used (see Breuker, et al., 1983[a,b], Wielinga, et al., 1984; and in particular, Breuker et al., 1983[c], and Breuker et al., 1984).

3.5.4.1 Documents Generated

3.5.4.2 M1, Lexicon

This document will contain all of the terms thought to be pertinent to the domain of expertise, some of these terms may have to be refined, deleted or enhanced at a later point in the analysis.

3.5.4.3 M2, Static Structure

The domain layer (cf. chap. 4) will take shape in this document. Concepts, relations and structures will be defined against those tasks identified in R4.

3.5.4.4 F1, Feasibility Estimate

This document will, once again, put the question of whether a KBS is needed or not. It will also contain a realisability estimation, in terms of whether or not what is required can be supported with the use of KBS technology.

3.5.5. Analyse Objectives and Constraints

This activity will consider organisational objectives and constraints from a system's point of view. That is, we shall need to establish what functionality the system will have to provide for the overall organisation. In considering this, we may need to

arrive at a compromise between the views of users. We shall, also, need to identify the changes eventually brought about to the organisation as a result of the introduction of the system within it. The likely constraints imposed upon the system's development and use will also need to be identified within this activity.

3.5.5.1 Documents Generated

3.5.5.2 R5, Objectives of Prospective System

This document should register objectives that the system should satisfy with regards to a consensus of users' views.

3.5.5.3 R6, Compatibility Requirements

This document will include all compatibility issues regarding possible other systems which will need to interact with the intended system, as well as software and hardware requirements in developing the system itself.

3.5.5.4 R7, Man-Machine Interface

This document will contain issues pertaining to the level, depth, and variation of machine interfaces foreseen for different users of the system. Man-machine interface is an important feature of any sophisticated system, providing for a level of interface which should be helpful to users without being unnecessarily elaborate and time-consuming.

3.5.5.5 R8, Development and Operational Environment

The environment in which the system will be developed and ultimately made fully operational will be described in here. Different aspects of the environment are: machine, operating system, language, tools, standards, methods, and system owners

organisation.

3.5.5.6 R9, Control and Security Constraints

All issues pertaining to the levels of security and use of the system, and legal implication of the system operation will be outlined here.

3.5.5.7 R10, Organisational Model

The same model as that described in R1 will be presented in here with some differences:

- The way the organisation will be , once the system is installed.
- Different levels of interaction between the system and the user organisation.
- The overall behaviour of the user environment, part of which is the system itself. Most of the culture shocks confronted as a result of system introduction will be mapped out in here.

3.5.5.8 F1, Feasibility estimate

This document will assess the possibility of supporting the external view both from a system and social (such as legality issues) point of view. The document may also incorporate suggestions as how to modify parts of the organisation, in order to make the system, and thus the solution, more feasible.

3.5.6. Analyse Expert and User Tasks

This is the second activity concerned with the internal view. During this activity we shall devise the other three layers (cf. chap. 4) of KADS four-layer model, as well as incorporating a model of the intended system users.

3.5.6.1 M3, Interpretation Model

A suitable interpretation model (cf. chap 5) will need to be identified and documented here. These are generic models of problem solving which will be instantiated in a bespoke fashion to suit different domains of expertise.

3.5.6.2 M4, Inference Structure

A structure containing primitive domain operations; in which, input and output to and from those operations are identified and combined in an inference structure. The operations and i/o objects are known as knowledge sources, and meta-classes respectively (cf. chap. 4). The inference structure will show all possible interesting primitive problem solving paths.

3.5.6.3 M5, Task Structure

Some of the paths in the inference structure are sequenced and combined together under the control of a number of control statements in the task structure. This is a structure describing the problem solving as abstracted from an expert(s) (cf. chap. 4).

3.5.6.4 M6, Strategies

Where possible, strategies should be devised to control and dynamically generate task structures, so that the system show greater flexibility, and 'intelligence' in problem solving than can be expected from the task structure alone (cf. chap. 4).

3.5.6.5 M7, User Model

This document should provide a basis for understanding how the user will choose to interface with the system. Some of the information may be volunteered by the user herself, and some of it has to be speculated by considering her level of competence in terms of familiarity with computers in general.

3.5.6.6 F1, Feasibility Estimate

All of the issues regarding realisability of the internal view up to this point of the analysis should be addressed here. The concern should be that of recording whether information and knowledge of the right level can be gathered in order to analyse the internal view. On the other hand, it should also contain an assessment of whether the knowledge would lend itself to representation schemes currently available within the AI technology.

3.5.7. Determine Functional Requirements

This activity is concerned with a synthesised view of the external requirements, and the expert tasks. The consolidated view should provide the overall requirements architecture.

3.5.7.1 Documents Generated

3.5.7.2 R11, Functional Requirements

All of the requirements which will make up the overall system will be documented here. The KBS is the problem solving part of the overall system, the other components of it are environmental and users constraints placed upon the problem solver.

3.5.7.3 R12, System Structure

A logical decomposition of the overall system will take place in this document, showing clearly different levels of interaction between different man and machine parts of the system.

3.5.7.4 R13, Information Requirements

The actual information needed within the system, and ways in which that information will be used within the system will be documented here.

3.5.7.5 R14, Expected Future Enhancements

This will contain all of the enhancements to be made to the system which can be foreseen at the time of writing the document.

3.5.7.6 R15, Consequences

Environmental, and man-machine consequences of installing the system will be documented here. This should indicate the level of resourcing required for the system, as well as changes it will bring about to the present working conditions.

3.5.7.7 F1, Feasibility Estimate

This document will be included only if the last activity in the analysis under the same title is not performed. The contents of this document will be similar to that of the said activity (see below).

3.5.8. Construct conceptual model

A four layer model of the expert(s) problem solving behaviour known as "conceptual model" (see chap. 4) will be devised within this activity. A large part of the model has already been constructed in an earlier activity, viz. "Analyse expert and User Tasks".

3.5.8.1 Documents Generated

The two model documents generated M6 and M7 are respectively concerned with refining "strategy", and "user model" sections of an earlier activity as mentioned

above. The refinement will take place in the light of further information made available as part of the last activity. Two other documents are also introduced:

3.5.8.2 R16, Knowledge Base Requirements

This document will contain information regarding the perceived size of the knowledge base, and all of the issues possibly not captured in the four layers, such as special type of reasoning or logic (i.e., fuzzy logic) which might be required for the design of the system.

3.5.8.3 F1, Feasibility Estimate

This document will contain an update of the feasibility estimate as contained in its counterpart in the activity "Analyse Expert and User Tasks".

3.5.9. Feasibility Estimate

This document will contain the overall feasibility of the external and internal views. A great deal of this document can be gleaned from F1 series of documents already produced within the earlier activities. In short, the document will address the combined feasibility of the two views, as well as making possible recommendations for different types of prototypes in assessing various system possibilities in application environments.

3.5.9.1 Documents Generated

3.5.9.2 R17, Development Requirements

The document will detail support requirements for developing the system. These might be aspects such as: machine time, experts, test environments, conference room, and so forth.

3.5.9.3 R18, Validation Procedures

Procedures for testing and validation of the system are outlined in this document.

3.5.9.4 F1, Feasibility Estimate

This document will contain the final feasibility estimate divided into four parts:

- Summary with the project teams conclusions,
- Detailed description of the external view,
- Detailed description of the internal view,
- Background materials for references, such as: tests, prototype results and so forth.

4. Coceptual Model - the Internal View

4.1. Introduction

Experts invariably rely on their extensive knowledge of a domain in performing their tasks. There is, however, a clear indication that structures and paths they pursue in performing their tasks is only partially present within the static elements of knowledge itself. Static knowledge is a collection of facts, and experiences which once established can be used independently of any particular human agent. An expert is largely valued for his capability in dealing with complex issues independently, an ability which extends beyond a straightforward use of static knowledge. He is, also, able to perform with a large degree of flexibility, with the possibility of 'graceful degradation', in confronting unforeseen situations and atypical problems.

It is widely recognised that a large degree of expert's ability stems from the control mechanism he uses in manipulating the more static elements of his knowledge. This has led to a number of AI workers to recognise the need for a meta level description over and above that of the object level at which the domain knowledge resides. The separation between the object and control level is seen both as desirable and useful within the AI community (see for instance, Davis, 1980; Clancey, 1983, 1985), and in psychological theories on problem solving (sternberg, 1980). The division between the two layers has largely been brought about by introducing a strategy level which controls reasoning using meta knowledge or rules about the domain knowledge itself (Davis, et al., 1977 [a, b]; 1982, Clancey, 1985). In the same tradition, in logic programming the notion of meta level control is introduced by Hayes (1973), and further pursued by Gallaire and Lasserre (1982).

One of the more interesting notions is the control of domain knowledge by the use of meta knowledge which is domain specific (Bundy, et al., 1979; Bundy & Sterling, 1981). This is further elaborated by Sterling (1984) maintaining that meta level knowledge embodies a theory of a particular domain. We hold with this view and maintain that such a theory is the basis on which expert's proficient and flexible problem solving capability is founded. We shall, therefore, exploit the idea in a modelling language we shall be describing next, a language that we shall be using for capturing the internal requirements of a system.

4.2. Analysis Modelling Language

The main point of departure between knowledge based expert systems, and earlier expert systems such as Dendral (Feigenbaum et al., 1971; Buchanan and Mitchell, 1977, 1978; Buchanan and Feigenbaum, 1978) is the extensive use of domain knowledge by the former. The domain knowledge will provide useful restrictions and constraints on thinking and tackling expert problems. The reasoning knowledge will, therefore, need to reflect those constraints, and benefit from them. In recognition of this fact a multi layered framework for modelling expertise (Wielinga & Breuker, 1986) is devised in KADS, in which the domain knowledge acts as the competence model for all of the other layers. The layers will provide a gradual and explicit means of transcending from the domain knowledge to that of control. The aggregate of the layers will make up the *conceptual model* which represents the 'internal view' (see chap. 3). The conceptual model is a model of the expertise at an *epistemological* level, which contains the theory of the domain as negotiated between the knowledge engineer and the expert. The model will not contain a purist view of the world in the sense of representing the psychology of expert's behaviour. On the other hand, beyond the use of a set of KADS vocabulary and notations, there is no requirement to employ any artificial or formal languages (including computer languages) in

constructing the model. We believe that the imposition of any such languages will go against the grain of KADS analysis. The aim at this stage is to capture the expertise in its fullest possible 'glory' devoid of any undue biases introduced by various implementation languages, which can distort an epistemological representation of the domain of expertise.

Four layers are identified which between them contain the object and control level knowledge. They are domain, inference, task, and flexible strategy layers which will be described next.

4.2.1. Domain Layer

This layer will contain the static knowledge pertaining to a domain of expertise. There are two primary sources of knowledge; one, documents and texts relating to the domain, and the other, transcripts (protocols) of interviews with expert(s). Knowledge in its raw form will need to be processed, and only those part of it thought to be in some way connected with the problem solving should be registered in the domain layer.

The basic element for representing domain knowledge is *concept* which has a conceptual and representational realisation. At the conceptual level, a concept is the real world interpretation of a set of percepts (cf. Sowa, 1984) associated with an object or an abstract entity. In this sense a concept can stand for virtually any individually identifiable entity, be it a physical object, or a concept as identified by the philosopher as an uninstantiated (viz. generic) predicate (for a detailed discussion of this sense of "concept", see Frege, 53, 60, and 72). We hold that expertise domains introduce sufficient real world constraints to save us from having to enter into any major philosophical or psychological debates over some of the more purist (abstract or metaphysical) views on the term "concept".

At the representational level, the structure of a concept comprises one or more attributes, each containing: a value restriction, a value, and a relation with possible other concepts. The value restriction will indicate the range over which a concept is valid, for instance the concept "positive number" has the value restriction of ">0". The value slot, should it contain anything, will cause the instantiation of the concept with a certain value. On the whole, we maintain a position similar to Brachman and Schmolze (1985) over concepts, except that we use rather different structural notations to them.

We find it useful to divide concepts into three types of 'concepts,' 'relations,' and 'structures', a distinction which will prove useful in developing a KBS. A relation is a simple concept whose role is to connect other concepts. Chair as a concept, for instance, 'consists-of' a back, legs, and a seat. The 'consist-of' concept in this example stands to describe the relation between the components of the concept 'chair' in some domain of interest. Two types of relations, in turn, can be identified: 'internal,' and 'external'. An internal relation describes the relationship amongst the components of a concept, whereas external relations are used to describe intra concepts relations. A structure is a composite cluster made up of other concepts combined together using external relations. A structure can, for instance, be the model of some process or a 'complex' readily observable in the domain. The most frequently occurring type of knowledge elements are relations and concepts.

The overall structure of the domain layer should reflect the necessary (essential) relations observed in the static knowledge. This structure stems from the way 'things' are in the real world, as well as how the knowledge engineer might interpret them for his purposes. The structure will ultimately provide the *axiomatic* organisation upon which the domain layer is founded. For instance, in NEOMYCIN (Clancey, 1985) two structuring principles are merged as part of the overall domain layer structure. The first of these is that each 'cause' in the aetiological hierarchies is related to others on the

strength of a theory of how such causes might be connected. It would have been equally possible to organise the hierarchies such that they would be dependent on the anatomical relationships between causes. That is, to cluster together those causes which are normally associated with a particular part of the human anatomy. The other structuring principle is one of relating causes with findings for diseases across hierarchies containing causes and findings. This principle stems from the way a general practitioner diagnoses a disease (cause) by basing it on the observed symptoms (findings) of his patient.

4.2.2. Inference Layer

The domain knowledge contains all of the facts which will need to be explored in arriving at a solution for an expert problem. This layer should contain the format of problems and solutions and all of the intermediate stages. The complexity of the control knowledge is not in constructing these stages, rather to map them onto paths ending in appropriate solutions. It is, therefore, useful to introduce an intermediate layer in which the domain facts and possible ways of traversing them in order to get from one to the other ones are made explicit. The outcome of such a layer can then be used as an 'A-Z' for the control knowledge (or the reasoning process).

The inference layer does exactly that in containing classification of domain concepts which could be conceived of as the collection of all of the stages, some of which a problem solving path may have to visit. Concepts are grouped together in different classes, according to the role they play in the reasoning process. For instance, in NEOMYCIN (cf. ibid) the data regarding patient symptoms are classed as 'findings'. Each of these classes is referred to as a *metaclass*, since it is used to describe the role of domain concepts in reasoning. A concept may be a member of more than one metaclass; for instance, fever as a concept could be either a symptom or a cause,

depending upon the course of a medical diagnosis.

The domain layer also contains a number of paths between concepts in terms of the *relations* between them. The paths are realised within the inference layer in the shape of *knowledge sources* (hereafter, also referred to as ks). Knowledge sources establish the link between metaclasses, and in that sense they are operations on the metaclasses. If we consider a metaclass as some problem solving state, ks's can be seen as providing the primitive state transformations by resulting in new metaclasses (states). We can, therefore, consider a ks as a primitive operation which uses the domain relations in order to establish the possible reasoning paths between metaclasses. The reasoning paths are primitive ones in the sense that they are inherent to the static knowledge and do not contain any control elements.

The composite of all metaclasses and ks's are brought together in a network called the *inference structure.* The structure uses a number of arrows to show the possible paths between metaclasses and ks's, the direction of the arrows is not an essential one and may be reversed, where feasible, during the reasoning process. The structure is a flat one containing no temporal dimension, any sequencing of the way metaclasses are traversed is introduced at the reasoning level. We identify ks's pictorially with ovals, and metaclasses with boxes respectively qualified with their names.

It is possible to construct a typology of knowledge sources, since we can observe that there are, in essence, a limited number of primitive domain operations across different fields of expertise. Other operations tend to be more high level and ultimately constructed out of the more primitive ones. On the other hand, metaclasses by nature tend to be more free formated, and less amenable to categorisation. This is due to the fact that problem states can be quite varied across domains, although there are some prime candidates which tend to occur time and again. We shall provide a typology of knowledge sources in the next section, before proceeding to describe the reasoning

process.

4.2.2.1 A Typology of Knowledge Sources

Knowledge sources are characterised in terms of the effect they have on their input metaclass(es), resulting in output metaclass(es). We can identify four types of generic operations which can take place on domain concepts within metaclasses (cf. Breuker et al. UvA, Davoodi et al. STC, 1987), these are:

(1) *Change Concepts*

(2) *Generate new Concepts*

(3) *Compare Concepts*

(4) *Manipulate Structures*

1) This operation is employed in order to manipulate the value of an attribute of a concept, two ks's are identified as performing this operation.

assign-value

[concept with attribute --> concept with attribute with value]: The ks will assign a value to the attribute of a concept, overwritting any previous value.

compute

[structure --> concept (in structure) gets value assigned to attribute]: The ks will evaluate the value of a concept in a structure, the structure itself is used to guide the computation of the value of the concept within it.

2) Generating new concepts is performed using the following ks's.

instantiate

[concept --> instantiated concept], [structure --> instantiated structure]: The knowledge source will create an instance of a generic concept or structure.

classify

[instance --> concept]: This is the inverse of instantiate ks; the ks will involve matching attributes of an instance in order to determine if it can be grouped under a certain concept.

generalise

[set of instances --> concept]: The ks will examine the attributes of a set of concepts in order to establish whether they can be classed under an already existing concept, otherwise it will create a general concept for them. The ks is similar to 'classify' in the case of an existing concept, in the other case where a new concept has to be generated the ks is also known as 'induction' (cf, for instance, Charniak & McDermott, 1985).

abstract

[concept --> concept]: The ks deletes a number of attributes from the input concept resulting in the desired output concept.

specify

[concept --> concept]: This is the inverse of 'abstract', in the sense that the output concept will have more attributes than the input one. The more abstract a concept, the higher it is likely to be in a hierarchy of concepts. On the other hand, the lower down the hierarchy a concept is, the more detailed and specific it is likely to be.

3) The comparison between concepts will take place using two ks's 'compare' and 'match', in either case resulting in an output concept representing the difference between the input concepts.

compare

[value of X , value of Y --> concept with difference-value]: The attribute values of two concepts are compared, resulting in a concept with an attribute value which is the difference of the respective input attribute values.

match

[structure of X , structure of Y --> difference-structure]: This is similar to 'compare', except that it applies to structures rather than simple concepts, the knowledge source is therefore more complex in nature than its counterpart 'compare'.

4) The manipulation of structures is an operation in which the input structure is transformed into some other structure. The knowledge sources performing this type of operation are:

assemble

[set of instances (components) --> part-of structure]: The ks pastes together a number of instances into a structure according to some skeleton structure. The instances will become the components (part-of) of the generated structure..

sort

[set/series of instances --> series of instances]: The 'sort' ks will arrange its input instances into a series of instances according to some predefined sequencing 'principle'. In this sense it is a special case of 'assemble', in which the skeleton structure describes a sequential arrangement.

decompose

[part-of structure --> set of instances]: This is the inverse of 'assemble' in which a structure is decomposed into its constituent components according to some skeleton structure.

transform

[structure1 --> structure2]: The knowledge source will transform the input structure into that of the output structure. Two types of 'transform' operation can be identified. The first of these produces an output structure which contains all of the input components but in a different order; the sort operation can be seen as a special case of this. The other type ends up with an output structure in which further details are added to, or abstracted from the components of the input structure. This is the more interesting and complex of the two types, which is also referred to as 'parsing'.

parse

(see transform).

The typology of knowledge sources as presented in here may not exhaust all possible primitive operations. We, however, think that the set is a very extensive one, with applications over a wide range of domains. In some cases, ks's are identified which do not correspond with any given here, in which case there are three possibilities to consider:

(1) Redundancy: A different name might have been used for an operation which is already contained in the set.

(2) Non-primitiveness: The operation is not a primitive one, i.e. one that cannot readily be observed in the domain. It, therefore, will not be justified to call it a

ks.

(3) New operation: In cases where neither of (1) or (2) are valid, then we have come across a new ks. The new ks should be added to the set of existing knowledge source types, sitting in the appropriate part of the typology.

The typical relations to be found in a domain, and those on which ks's are founded are:

is-a

Instance-of (is-a) is a relationship existing between concepts in a hierarchy in which concepts in the hierarchy are instances of their parent nodes. The relation is also known as 'refinement', since concepts lower down the hierarchy refine the more abstract concepts above them, by adding further detail.

consist-of

This relationship exists between concepts in a hierarchy in which nodes higher in the hierarchy *consist of* those below them. This is also known as 'subsume' relationship, since every concept in the hierarchy subsumes those under it. The inverse of this relationship is known as 'part-of', since every concept in the hierarchy can be viewed as being 'part-of' those above it.

caused-by

This is a relationship existing between concepts caused by each other. For instance, a 'hot tin roof' is *caused by* 'mid afternoon summer sun'. This type of relationship usually implies time, since for something to be the case, something else must have happened *prior* to it, to have caused it.

empirical

This is a relationship between concepts based on some statistical or certainty factors. For instance if observed symptoms are 'headache,' patient suffering from meningitis.

The relations identified here are some of the more usual ones, others may be identified (such as quantitative), and named accordingly as they are discovered in a domain. In short, what connects two or more concepts is a relation. If it is to be used at some point for traversing the concepts then it must be named and defined, else it may be left implicit within the concepts and structures.

4.2.3. Task Layer

In 'think-aloud' protocols in which the expert utters his thoughts whilst going through a problem solving scenario, the session can be recorded and used to abstract from it the solution path. The path and elements involved in it can then be discussed and agreed with the expert, before they are turned into task structures. The more general the problem solving task, the wider will be the application of the resultant structure. Think aloud protocols are an important way of devising task structures, and certainly the most reliable when a competent domain expert is available.

A task structure usually contains three types of statement:

(1) *goal statement*

this will describe an operation in order to satisfy a certain goal within the overall solution tree. Each goal statement will make a contribution to achieving the overall solution, and can thus be regarded as a subgoal. A goal statement will contain one or more ks's, where every ks will have some domain concept parameters as part of metaclass(es) input to it.

(2) *control statement*

this statement will control explicitly the order and frequency of application of goal statements. The order in which goal statements appear in a task structure implies an inherent sequential ordering control. Control statements such as 'if..then' can disturb the sequential ordering, if and when this is required.

(3) *modality statement*

the mode of external interaction and communication of the intended system is contained within modality statements. The statements will describe the way in which data and knowledge exchange should take place between the system and the outside world (e.g. users, data-bases).

Task structures are a powerful means of representing fixed strategies for problem solving, they constitute the main sequencing and manipulation of domain knowledge via the inference structure. A task layer can contain several task structures for different problems, or a general one which can be modified to suit different situations. A task structure, however, will not contain the flexibility that experts employ in dealing with different, sometime unforeseen, situations. In the next layer we propose a framework for representing a flexible problem solving strategy, providing the top level control mechanism.

4.2.4. Flexible Strategy Layer

One can argue that in providing a sufficient set of varying task structures, we are able to tackle a large enough number of expert problems in a flexible fashion by switching between different structures. This is a reasonable view, except that its achievement is quite complex and currently the most seminal part of the analysis stage.

In order to shift attention from one task structure to another we need:

- a) A scheduling, or overall supervisory control mechanism which can allocate task structures to different problems.

- b) A monitoring component which both understands when a new problem is encountered, and one which can also oversee the success or failure of a task structure in action.

- c) An execution component which will cause the execution of one task structure until control is passed to another one.

- d) a, b, and c suggest a planning loop in which the execution of task structures is monitored, with the possibility of invoking new structures or aborting some of the old ones. The loop terminates once a solution is found, meanwhile the system's behaviour would be more 'non-deterministic', and more flexible and problem oriented than that suggested by a fixed strategy.

The flexible strategy layer will allow for such a planning-action-loop (above), with the additional possibility of generating some of the task structures dynamically. The latter will save excessive effort at the task layer, by limiting that layer to defining only those structures thought to be generally applicable. Of course, strategy paradigms like the one we are suggesting in this layer, have applications in all fields of AI work. It is equally recognised that the achievement of such general planning systems is a difficult one. It is no wonder that most of KADS KBS works contain themselves to the first three layers, providing only a descriptive support for the flexible strategy.

We can cater for a more modest flexible strategy description (difficult to call it 'structure') by limiting its functionality to scheduling control amongst different task structures. The schedular can identify the appropriate task structure (procedure) by examining control rules and deciding what behaviour needs to be invoked next. The meta strategy in NEOMYCIN (Clancey, 1985) consisting of a number of meta-rules which

decide what procedure to invoke next is a good and classic example of this. We shall next examine the application of the four layer model in the context of NEOMYCIN. The example should help to understand the ideas, notations, and the analysis modelling language described above.

4.2.5. A Post-hoc analysis of NEOMYCIN

In this example we shall provide a very brief account of a four layer model for NEOMYCIN, the system will be described in some detail in a later section (see chap.8).

4.2.5.1 NEOMYCIN - a brief description

NEOMYCIN is a medical diagnostic computer system concerned with diagnosing meningitis, and related diseases. Diagnosis is based on observed symptoms of the patient. The architecture of the system is a general one, applicable to most diagnostic problems.

4.2.5.2 Domain Knowledge

This will contain the concepts, relations, and structures pertaining to NEOMYCIN's domain of medical facts (Davoodi, 1987[c]; also in Davoodi et al, 1987). The concepts are hard (lab) and/or soft (circumstantial) data, as well as a dictionary of possible diseases (causes). The main structures in the domain are the hierarchies of diseases (aetiological taxonomy), hierarchy of symptoms (data on patients), and networks connecting symptoms and causes across hierarchies.

The major domain relations are subsumption (consist-of), refinement (is-a), and causal. These relations connect concepts within and across symptoms and causes hierarchies.

4.2.5.3 Inference Layer

The inference structure (fig. 4-1) contains three knowledge sources abstract/transform,

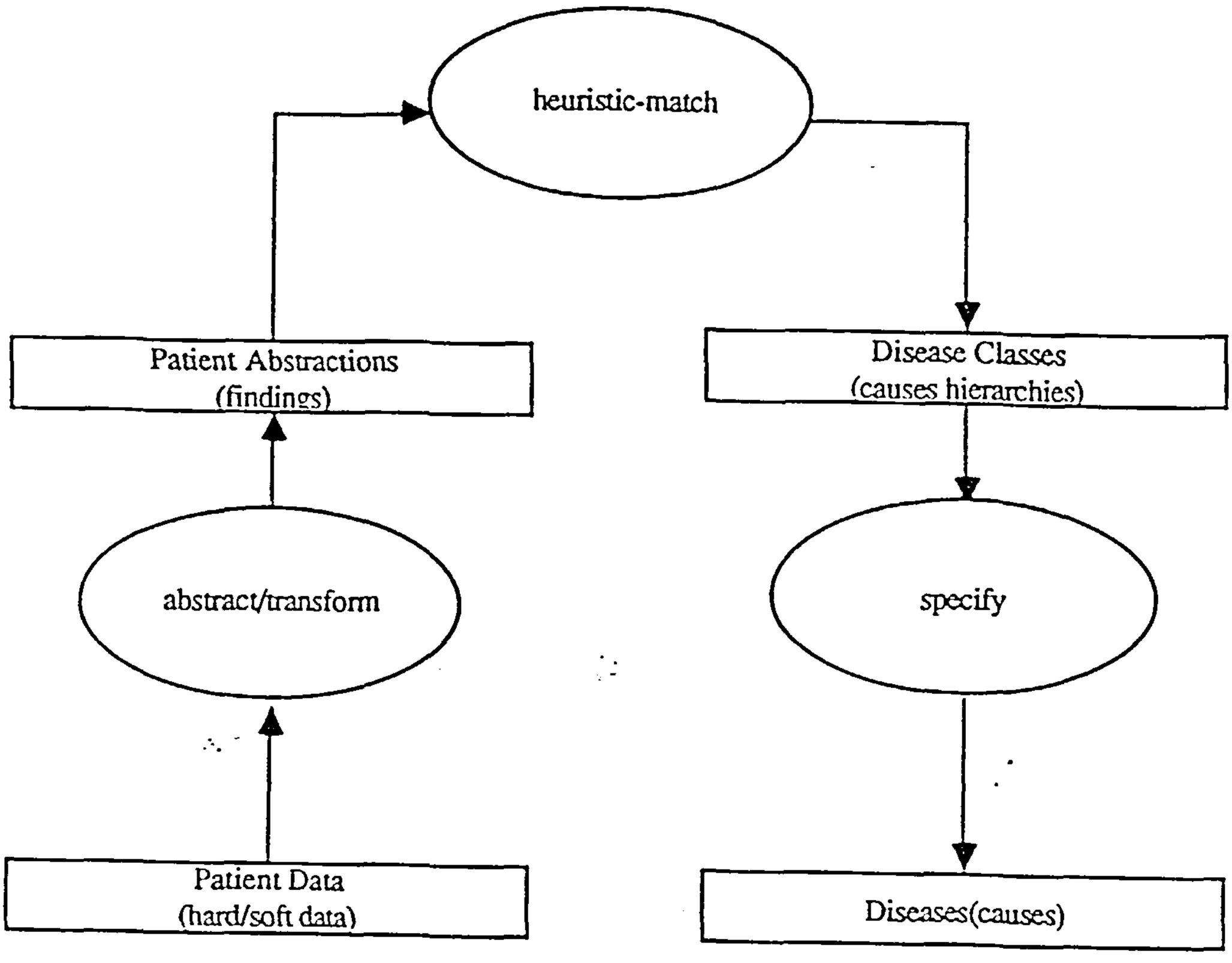

Figure 4-1: *NEOMYCIN's Inference Structure*

heuristic-match, and specify. 'Abstract/transform' will use domain relations to derive findings (symptoms) from the given patient data. The patient data will consist of laboratory test data, as well as circumstantial evidence observed by the GP. The ks is the combination of two knowledge sources 'abstract' and 'transform'; 'abstract' will ensure that those parts of patient data relevant to symptoms will be used. 'Transform' will then take over, and will ensure that the resulting concepts will have structures similar to those within the respective symptoms within the domain. It is not unusual to see these two knowledge sources often appearing together in different domains,

since their operations are complementary to each other.

'Heuristic-match' is a variation of knowledge source 'match' in which "certainty factors" are used. The "certainty factors" themselves are part of the domain concepts. This ks will correspond patient findings (symptoms) to possible diseases using probability measures (certainty factors). The knowledge source employs, in main, the causal relations between findings and causes in order to bridge across from one hierarchy to another in order to provide the initial diagnosis. The initial diagnosis is further focused by the ks 'specify', thus enabling the system to identify the exact cause of the problem. This ks uses, in main, the 'is-a' relationship between causes in order to climb down the hierarchy of causes to arrive at a more detailed and precise causes of the problem. In our description of the ks's, we have also provided an account of the nature of metaclasses used as input and output to/from ks's.

4.2.5.4 Task Structure

We present in here a rather general and simple task structure, which is almost self explanatory (4-2). The structure does not contain any control statement,

```
diagnose(disease)

        abstract / transform(data)
                obtain(data)

        match(findings, diseases)

        specify(disease)
```

Figure 4-2: *NEOMYCIN's General Task Structure*

the control, therefore, is implicit in the sequencing of the statements within the structure. The indentation used is indicative of the order of carrying out the operations. That is, the satisfaction of a goal statement will require, a priori, the execution of all of the statements below it which are indented further to the right of that statement. The only modality statement employed here is 'obtain (data)'.

4.2.5.5 Flexible Strategy

This is contained within a set of meta rules, in NEOMYCIN meta strategy layer, which order and invoke different procedures (subtasks) depending on the course of diagnosis.

4.2.6. Summary, and Discussion

The four layers, as part of the KADS conceptual model, will provide the means to capture the expert static and reasoning knowledge at an epistemological level. The "four layers" will make it possible to capture a model of 'expertise' in its full glory, thus ensuring that the ultimate system is a close variation of the expert in terms of the extent of the problem solving capability. The model should, also, make it possible to grasp a clear understanding of the relationship between the problem solving (internal view), and the external view components. The nature and extent to which the two views are related will be reflected in the design of the system at the later stage of design phase (see chap.7).

The conceptual model itself can be used for a model driven knowledge elicitation, a most useful concept in the often difficult process of knowledge acquisition. The model can often, also, help the expert in discovering gaps in his own knowledge, and will therefore encourage him to extend or amend his knowledge. In this sense, a conceptual model may be revised a number of times before the knowledge engineering team,

and the expert(s) are happy with it. Such a close collaboration, and involvement is most useful when it comes to designing the system, at which point the expert will share a lot of insight about the stage of development.

We have found that describing the expertise domain in four consecutive layers (see fig. 4-3) will allow us to gradually and

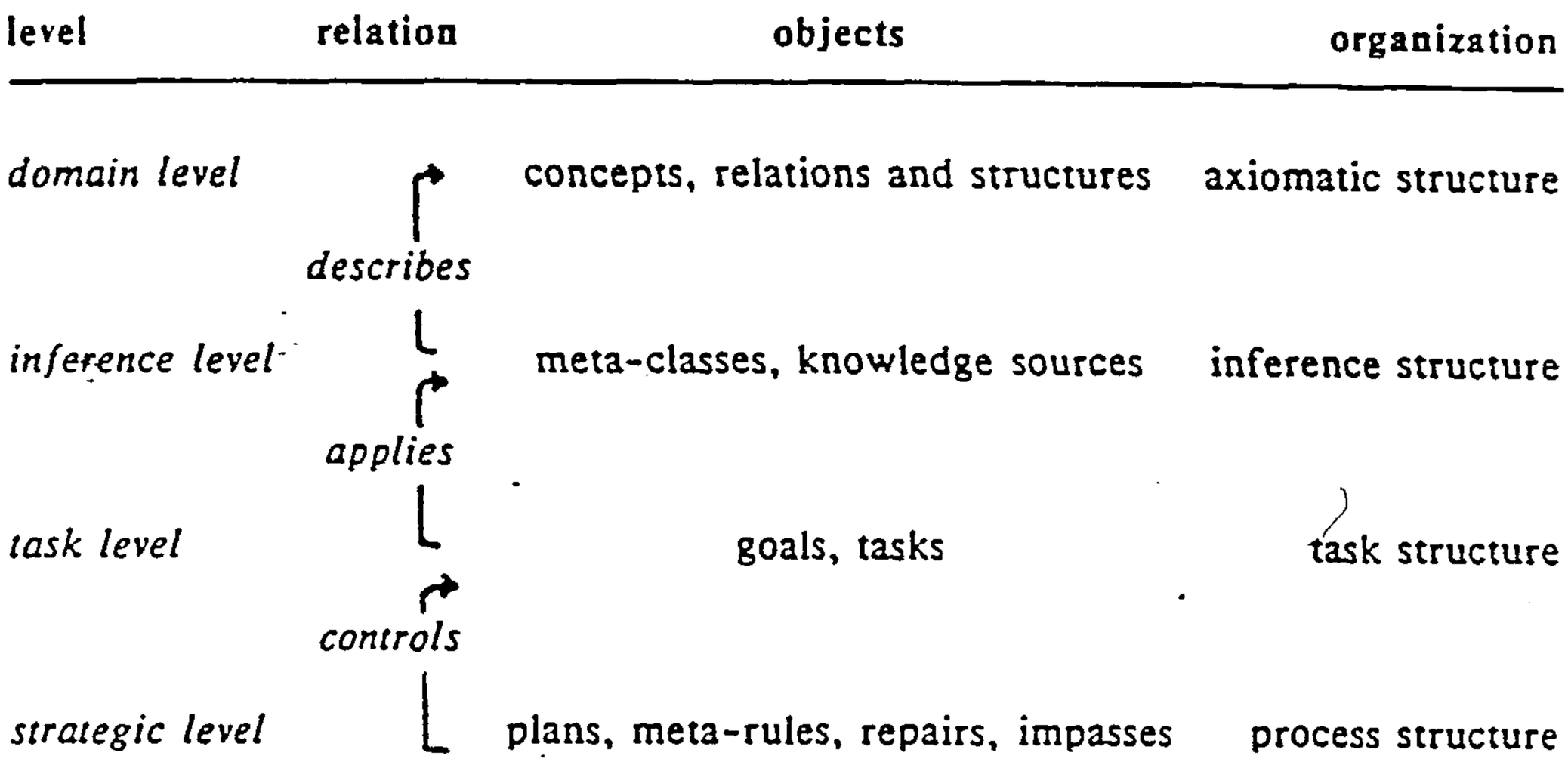

Figure 4-3: *A Schematic Representation of the KADS four layer model*

explicitly transcend from static to control knowledge. The domain layer will contain all of the knowledge elements already observable independently of the expert (this is what we mean by static knowledge). The other layers will capture the dynamics of problem solving within the domain using inference paths (in the inference layer), control statements (in the task layer), and at times using flexible strategies (in strategy layer). The concern of these three layers is, therefore, to capture the reasoning and control knowledge. The domain layer will act as a competence model for all of the

other three layers throughout this process. It would have been possible to reduce the number of layers, but we believe that would have made the process less discernible to analytical scrutiny, and the nature of various relationships between the reasoning and the underlying facts would have been made less clear.

In the next chapter we shall consider *interpretation models* which are based directly on widely applicable conceptual models. Interpretation models will provide a powerful modelling tool both for knowledge acquisition, and for developing knowledge based systems.

5. Interpretation Models

5.1. Introduction

A number of researchers over years have registered their concern over a lack of proper understanding of the nature of the knowledge and data to be used in a KBS implementation. Incremental prototyping has emerged as a way of interpreting and acquiring further knowledge (cf. Hayes-Roth et al., 1983), thus making it possible to negotiate and reflect the required understanding through progressive levels of experimental development. Incremental prototyping, however, is an insufficient medium for providing a progressive understanding of domain expertise.

The process lacks expressiveness in being constrained by a machine formalism in the shape of some programming language. The language constrains what can be read into, and read off from the prototypes, thus making it impracticable either to capture or to negotiate the expertise through those prototypes. The process of modelling in KADS, on the other hand, will not use any machine language in achieving a model of expertise. The only constraints are that certain diagramming techniques, vocabulary, and ordering of different layers of knowledge will need to be observed.

We are aware of the argument that certain type of tacit knowledge cannot be captured within any model of the kind we are describing. We are also aware that some of the proponents of 'incremental prototyping' would argue that some tacit (or soft) and non-tacit knowledge can only be captured by the use of such prototypes. Beyond the following observations, we shall not discuss any further aspects of the apparent dichotomy between the two schools of 'structured methodology', and 'incremental prototyping':

- If incremental prototyping of certain type of tacit knowledge is to converge on a working expert system, then each prototype has been based on some mental model of that knowledge on the part of the developer. Furthermore, the experience of the developer has enabled him to choose the right programming tool for developing the prototypes. We propose that such mental models should be explicitly described using KADS modelling language.

- If some elements of knowledge can be prototyped, and successfully converged onto a working system with little effort, then it would be useful to examine whether those elements are species of conventional systems assembled as 'heuristic' expert systems. This type of development is quite common place in industry and it has created a technology hype, which is mostly attributed to commercially available expert system shells. There are many examples of 'spread sheet' applications dealing with co-relating a number of matrices, which appear under the banner of expert systems, simply because they are developed using expert system shells. Examples of this type can be seen in applications dealing with expert systems for licensing high technology equipments to certain parts of the globe, or many 'help desk' expert advisors. In all such cases it would be possible to translate the production rules used within these systems into initial entries (variables) of spread-sheets, thus deciding the allowable permutations by examining the various spread-sheet matrix entries (co-relations, or premises).

As we have mentioned earlier (see chap. 4), we shall need to describe the domain expertise at the level of theory of knowledge (epistemological level) as seen from the common point of view of the expert(s) and the knowledge engineer(s). This description should provide for what Newells (1980) calls knowledge level, or the "missing level" of Brachman's (1979) analysis of semantics network. The description closely

represents the elements and structures of a domain, and ways of exploiting them in solving problems pertaining to the domain. Our representation of expertise domain is neither a purist one, nor one which is unduly biased by any implementation vehicle. It is not purist in the sense that it does not employ detailed level psychological models of the expert behaviour, rather an interpretation of it as agreed between the knowledge engineer and the expert. The intention is not to use KADS to represent the psychology of experts' problem solving behaviour. The intention is to capture those elements of experts' knowledge, which are seen as being essential to the *actual* process of problem solving.

Conceptual models (cf. chap. 4) should be seen as the major touchstone for devising generic problem solving models which capture the epistemology of a number of interesting domains. We refer to these generic models as *interpretation models* (we also refer to this as IM), since they can be used both in constructing a model of an expert domain, and in *interpreting* data within that domain using the model itself. An IM, similarly to a conceptual model, is a powerful tool for facilitating knowledge acquisition, which is often thought to be a bottleneck in developing a KBS. It provides a model driven framework for conducting interviews with experts, and analysing data thus generated.

Interpretation models are usually devised for problem solving behaviours which can be shared amongst a number of useful expert domains. The main difference between an IM and a conceptual model is the range of applicability, and the level of specificity. A conceptual model is very specific toward a particular domain, and any wider application is accidental as opposed to intentional. An IM, on the other hand, is only specific toward a domain in the sense that it can be used as a template for devising a conceptual model for that domain. The range of application of an IM is the set of all domains which can use that IM as a template, in the way described above, for part or

all of the problem solving behaviour pertaining to those domains.

In this chapter we shall describe the process of constructing interpretation models, and the major role they play within the KADS methodology. We shall also describe different types of these models and the space of expertise domains to which they may apply. In the end we shall examine an IM in a hypothetical domain, in order to demonstrate the power of IMs. In the next chapter we shall provide a real life example of an interpretation model for devising a conceptual model for a financial domain. The financial domain in question currently benefits from the use of a knowledge based decision support system which is founded on the said conceptual model (cf. Davoodi 1987[a]).

5.2. How to Construct an Interpretation Model

An IM is an abstract problem solving model based on a conceptual model for a general domain (or class of domains) of expertise. It is important that the domain considered should contain a problem solving behaviour widely observed in other domains. Otherwise the conceptual model based on it would be of limited application, and thus not a good base for an interpretation model.

The process of abstraction is one of deleting from the conceptual model all that is particular to a specific domain, thus ensuring that the resulting IM can be used as a template for modelling domains of that nature. The parallel to this can be observed in EMYCIN (Empty MYCIN), or HERACLES abstracted from NEOMYCIN. These generics models are abstracted from computer systems, whereas IMs are abstracted from conceptual models devised using the descriptive power of KADS methodology. In practice the 'domain layer' is deleted from the conceptual model, and all of the references made to that layer are modified such that they reflect a sense of genericness. That is, the names of metaclasses are suitably replaced with ones which can be applied

across domains, thus affecting some of the details in the inference and task structures. It may also be necessary to replace the titles for some of the goal statements within the task structure with some general ones. If the nature of communication between the system and the outside world is also domain dependent, then suitable *modality statements* should replace the more specific ones in the task structure.

An example of the process of arriving form a conceptual model to an IM can be seen in the IM for hardware configuration (cf. Davoodi, 1986[a]) in which a generic modelling template is based on a conceptual model for designing computer hardware configurations. The IM thus generated can be used for all types of hardware design configurations in which the design components and ways of assembling them into modules are already described. It may, for instance, be possible to use the IM for describing the behaviour of R1 (also known as XCON, McDermott, 1980), and systems similar to it, if only in parts.

Finally, sensible names should be chosen for IMs which indicate the class of problems for which they are intended. The name of an IM should help the potential user of a library of such models to concentrate on those IMs which seem to be of relevance. We have taken the view that both the type of IMs and the space of expert problems to which they apply should yield to some form of classification. This categorisation should, in turn, help us in devising a library of interpretation models aimed at supporting various classes of expert problems.

5.3. Types of Interpretation Models

An IM is likely to be of one of the two types of 'generic,' or 'real life' template. We shall firstly have to dispose with an apparent paradox in our typology of IMs. An interpretation model, by definition, is a general problem solving model (template) for a class of domains of expertise. Both 'real life,' and 'generic' IMs are general models

in this sense. The major distinction between the two types, however, is the level at which their use is perceived. An IM of the 'generic' type will always need to be *part of* a larger template before it can be used for real life applications. That is, the type of task for which it is intended cannot be seen in real life as one which is an end to itself. Rather it is a task which is aimed at some general (thus generic) problem solving behaviour whose presence is needed as part of a combination of tasks. 'Assessment' is an example of a generic task, which should appear in all domains in which the solution to a problem will depend on assessment of some prerequisite, say, parameters.

Real life IMs, on the other hand, are aimed at supporting classes of expert tasks which can be identified in the real world as tasks whose achievement is both the means and the end to a problem. Real life models usually contain one or more generic models within them (see 5-1), though there are real life models whose

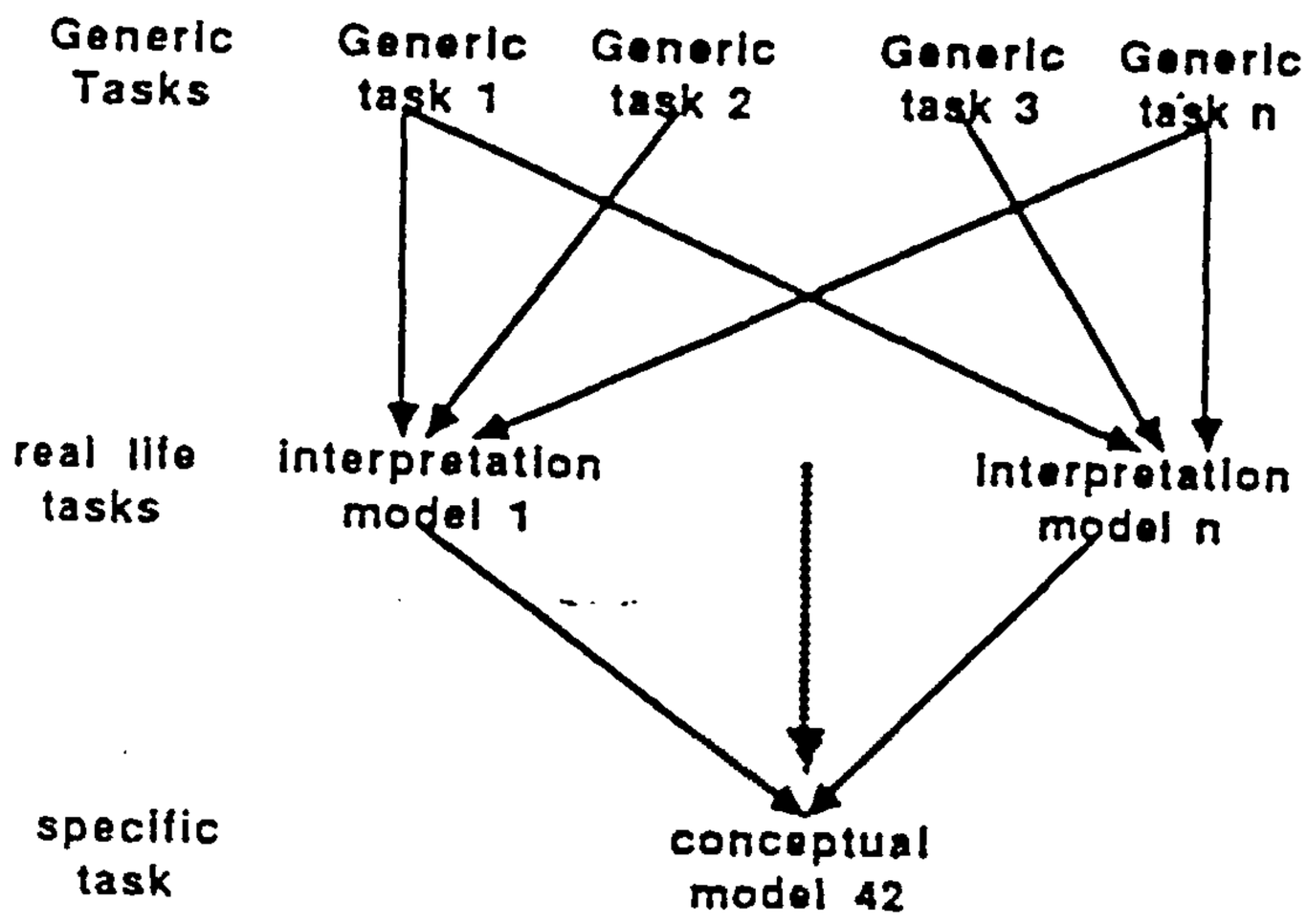

Figure 5-1: *Construction of a Conceptual Model*

composition is entirely independent of any 'generic' type. Most of the models contained in the library of interpretation models (see below) consist of the 'generic' type.

Although, as time goes by, more and more real life models will be identified in using the KADS methodology at a wider front of KBS applications.

5.4. A Classification of Generic Tasks

The type and number of IMs we shall identify will, naturally, depend on the space of problem solving domains, and a classification of them. The broadest view of a task is one in which, we either seek to construct or 'invent' some solution given a pattern as a problem, or that given the problem we shall try to 'somehow' find a path to a solution which is already in existence within a domain of expertise. Tasks will, therefore, divide between those of 'analysis' and 'synthesis' depending upon whether the solution already exists within the domain of expertise being considered, or that one has to be devised.

We, also, identify an in-between case in which part of the solution may have to be constructed, or reshaped, whereas the rest of it will contain elements already within the domain. This third type we refer to as 'modification tasks,' which could tend toward either end of the 'analysis' to 'synthesis' spectrum, depending upon how extensive a modification the solution to a problem will have to undergo. The classification of tasks we have identified here will allow us to construct an IM associated with each generic task thus identified (cf. Breuker et al., UvA, Davoodi et al., STC, 1987).

5.4.1. Analysis Tasks

Two classes of such tasks can be identified (fig. 5-2 cf. Breuker et al., UvA, Davoodi et al., STC, 1987) depending upon whether the solution is an attribute of the task domain, or whether it is a state which will change over time. In the first case the task of solving the problem is one of *identifying* (see fig. 5-2) the solution amongst the many possible domain attributes. In the second case, in order to isolate the solution

```
system_analysis
|   identify
|   |   classify
|   |   |   simple_classify
|   |   |   diagnosis
|   |   |   |   single_fault_diagnosis
|   |   |   |   |   heuristic_classification
|   |   |   |   |   systematic_diagnosis
|   |   |   |   |   |   causal_tracing
|   |   |   |   |   |   localisation
|   |   |   |   multiple_fault_diagnosis
|   |   |   assessment
|   |   monitor
|   predict
|   |   prediction_of_behaviour
|   |   prediction_of_values
system_modification
|   repair
|   remedy
|   control
|   |   maintain
system_synthesis
|   transformation
|   design
|   |   transformational_design
|   |   refinement_design
|   |   |   single_stream_refinement_design
|   |   |   multiple_stream_refinement_design
|   |   configuration
|   planning
|   modelling
```

Figure 5-2: *Taxonomy of Problem Types*

we shall need to predict some future state within the task domain. For instance, in order to predict what state a chemical substance may be found in an experimental domain, we shall need to take into consideration the relevant attributes of the substance together with external constraints imposed on it such as temperature, and atmospheric pressure. Toward the end of this chapter, we shall use 'classify' as a member of 'identify' group of tasks in order to show the IM associated with that task, as an example of how such models can be used in real life problem solving.

5.4.2. Modification Tasks

This type of tasks are closely enmeshed with analytic tasks, in which the 'cause' of a problem is isolated, and then made subject to some *modification* in order for it to be put right. The contingent relationship between analytic and modification tasks can be summarised as thus (cf. ibid., p. 49):

> heuristic classification -> remedy, repair
> causal tracing -> remedy, repair
> localisation -> repair
> monitor -> control, maintain

The difference between "remedy", and "repair" is in the nature of the modification, this is discussed towards the end of this section. The modification tasks in real life will appear either as the concluding part of analytic tasks, or that they are quite complex in behaviour. In case of the former their inclusion is, almost, implied by the analytic tasks consuming them. For instance, medical diagnosis is usually seen as a 'classification' problem in which any proposed remedy for an ailment is a secondary feature of the problem solving behaviour. That is, once the cause for a disease is isolated the major part of the solution is delivered, and it will only remain to associate that cause with some sort of prescription. Beyond a separate classification that can proposed for such tasks, it would be quite difficult to devise IMs relating to them. Since, any such attempt may fail in one of the following ways:

- a possible IM may contain too large a section of an analytical IM for it to be identified separately as a template for a class of modification tasks.

- because of the overtly complex nature of the modification task, any IM attempting to capture it, is likely to appear as a series of guidelines as how to proceed rather than as the kind of template that was intended originally.

We shall, however, provide a description of the types of association one can seek

between the analytical and modification tasks listed above.

In 'repair' a defective component is replaced (or extensively modified) after it has been identified using one or a combination of the, three classes of, analytical tasks identified previously. The component(s) to be repaired are isolated by disassembling a composite structure or system. This may involve planning of ways in which the device is continually decomposed into its constituent parts, and each part is then examined. The examination should finally arrive at one or more defective components in need of repair.

In 'remedy' a process or malfunction is counteracted by initiating another process. A typical example of this will be in process control, in which a corrective measure may be needed for avoiding process features which somehow disturb a system's norm of operation. Control is a task in which parameters of a system which deviate from some expected or desired state (or value) are controlled by having their values changed or refused categorically. The control mechanism itself is triggered by some discrepancy observed in the system, indicating that some system components (parameters) are outside a predefined acceptable range.

5.4.3. Synthesis Tasks

The type of synthesis task will depend on the nature of input and output to / from the task (or process). Input to 'design' tasks comprise such elements as functional specifications, and external requirements resulting in an output consisting of a detailed architecture annotated with the description for the actual requirements. 'Planning' is similar to 'design,' except that it takes as input 'activities,' and partial priority orders resulting in a dynamic schedule for assigning activities to processes containing the all important temporal axis.

Modelling tasks are, also, similar to design, except that the input consists of more than just the requirements and the constraints, but also of data. The output is an abstraction of data in a framework which brings out the tacit interactions between data, and the roles of data within the given input domain. The nature of the framework, itself, will depend on the type of the modelling language utilised.

5.5. Interpretation Model - the Use

In this section we shall examine the use of an interpretation model, namely the 'systematic diagnosis' IM, in order to illustrate how such models can be applied to their intended domains.

5.5.1. A Template for Systematic Diagnosis

Description: systematic diagnosis is a branch of problem solving concerned with identifying defective device components using structures depicting the explicit relationship amongst the device components. If the structure used is one in which a 'part-of' model of the device is used, then the diagnosis is said to take place by *localisation*. That is, in using the part-of structure model of the device, one is able to isolate and thus identify the defective component by homing in on its *location* within the structure. The other type of systematic diagnosis is diagnosis by causal tracing, in which the device structure is described in terms of a causal network, in which different components are connected using causal paths and are thus traceable.

The conditions which should exist for a domain to be amenable to either of the two diagnoses are as follows:

Conditions for the application of diagnosis by localisation:

- The presence of or possibility of developing a 'part-of' model for the device to be examined. The model can be in the form of some configuration design, or a drawing, or any other specification which yields to decomposition of its constituent parts in an explicit fashion.

- The possibility to test system components, and the availability of system output behaviour in terms of the output of individual components within it.

What has been said thus far makes 'diagnosis by localisation' IM a prime candidate for trouble shooting in electrical and electronic devices. The circuit layout or design of circuitry of a device such as a computer can be used as the required part-of model. The model can then be decomposed into its constituent modules and submodules, such as half-adders and transistors, in order to test the suspect components individually.

Conditions for the application of diagnosis by localisation:

- The ability to show clearly how the device (or system) functions in terms of a set of causal relations (paths) between its constituent parts, as well as between possible states the device might go through during diagnosis.

- In order to invoke causal tracing in an effective, and thus useful, manner, there needs to be some data available on a significant part of the components of the system causal model.

A typical domain for diagnosis by causal tracing is trouble shooting in mechanical systems, such as car engines. Steels and Velde (1985) provide an example of a causal network for a car engine, in which malfunction within the engine is causally traced by testing various observable states within the device.

In trouble shooting of devices causal tracing and localisation often alternate; a good example of this can be observed in SOPHIE III (Brown et al., 1982) in which both

views are incorporated. Diagnosis by heuristic classification is another major form of problem solving behaviour, in which short cuts are taken in arriving at possible problem areas, by the use of certainty factors. It is possible to see the application of diagnosis by heuristic classification in cooperation with one or both of the other types of diagnosis in a domain. The main role of heuristics in such applications will be to concentrate on the problem area by the use of 'educated guesses' (heuristics). It will then be possible for the other two types of diagnosis to take over and *systematically* go through the components or states of the problem area until the defective component(s) is identified. In the following sections we shall concentrate on the IM for diagnosis by localisation.

5.5.1.1 Inference Structure

The inference structure for systematic diagnosis is depicted in figure 5-3 (cf. Breuker et al., UvA, Davoodi et al., STC, 1987). A description of metaclasses and their respective domain concepts can be found in figure 5-4 (cf. Breuker et al., UvA, Davoodi et al., STC, 1987).

5.5.1.1. 1 A description of KS's

In this section we shall provide a brief description of knowledge sources, their input and output metaclasses and the kind of domain knowledge required by them as seen within the inference structure (fig. 5-3).

Select a system model

select: Diagnosis is initiated with the selection of a 'part-of' (or 'consist-of') representation of the system in which one or more components are suspected to be faulty.

Input: *complaint* can be about a system with at least one faulty component.

Output: *system model* is the actual 'consist-of' (or 'part-of') model of the system

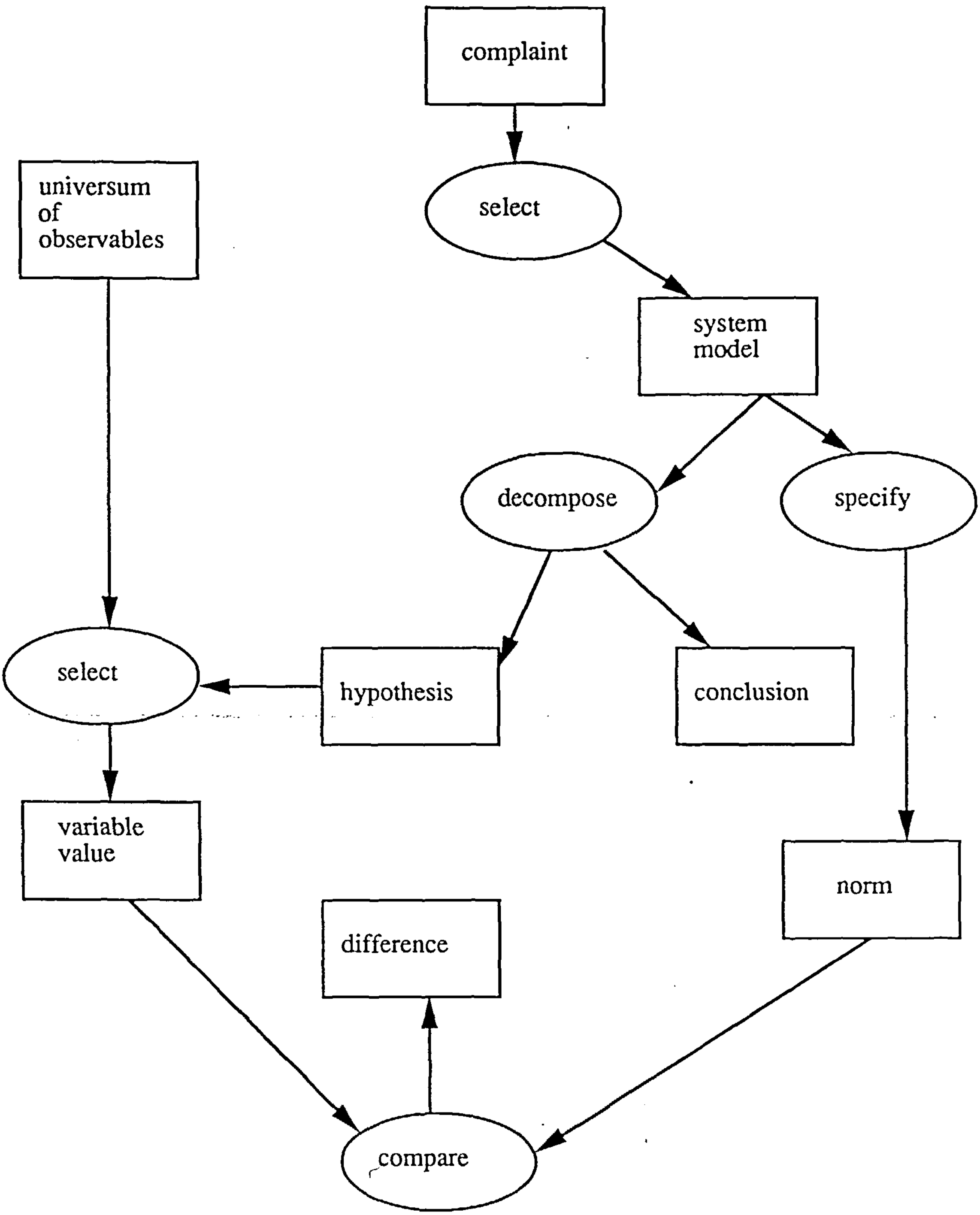

Figure 5-3: *Inference Structure for Systematic Diagnosis*

meta class	*localisation*	*causal tracing*
system model	part-of model	causal model
complaint	faulty system	faulty state
universum of observables	observable output variables	observable states
hypothesis	(sub)system containing faulty component	sub-network
variable value	observed output value	observed state
norm	output specification	top state causal subnetwork
difference	faulty component	faulty state
conclusion	faulty component	cause of complaint

Figure 5-4: *Metaclasses in Systematic Diagnosis and their member concepts*

under diagnosis.

Domain knowledge: knowledge about the system behaviour and structure will be required by this knowledge source.

decompose the system model

decompose: the system is decomposed into a set of components which sit in the part-of hierarchy. Every new decomposition will result in arriving at the next lower level of the part-of (or decomposition) hierarchy.

Input: *system model* which is the part-of model as described before.

Output: *hypothesis, conclusion*; a hypothesis is made in terms of 'suspect' faulty component, in our jargon the component is the hypothesis itself. Having decomposed the suspect component into sub-components to a depth at which no further

decomposition is possible, the resulting sub-components form the 'conclusion'.

Domain knowledge: this is the system's 'consist-of' or 'part-of' model.

Select a variable value

select: given the 'universum of observables' (see below), variable values (see below) are selected.

Input: *hypothesis*, and *universum of observables*; the latter is the collection of observations on the system components behaviour (or output). The 'universum of observable' is used to associate with a hypothesis a certain value known as 'variable value'.

Output: *variable value*

Domain knowledge: knowledge about various methods of testing different system components is required.

Specify a norm

specify: this knowledge source specifies the *expected* output value (or behaviour) of the system or components within it. This output is, naturally, associated with a fully working (thus expected) version of the system under diagnosis.

Input: *system model*

Output: *norm*, which is the already mentioned 'expected system behaviour'.

Domain knowledge: knowledge about system behaviour

Compare the variable value with the norm

compare: this ks will make it possible to compare the expected system behaviour against its *actual* output.

Input: *variable value* and *norm*.

Output: *difference* will make it possible to examine if the actual output of the component is in variance with its expected output.

Domain knowledge: the judgemental information which will make it possible to assess the significance of the 'difference' between the expected and the observed. In the sense that, whether a given difference is cause for further investigation on a component, or that it can be ignored on the basis of usual wear and tear.

5.5.1.2 Task Structure

The diagnosis starts with the selection of a system model, for which an iterative diagnosis is carried out. The diagnosis is concluded once the last of components within the 'conclusion' metaclass is dealt with. At this point all components diagnosed to be faulty have been clearly marked and can be recommended for further action of the 'repair' type. The task structure is depicted in figure 5-5

```
Task
    Diagnose(fault)
        select(system model)
            while (no conclusion)
                decompose(system model)
                    while (no. of hyps in diff. > 1)
                    select(variable value)
                    specify(norm)
                    compare(var. val, norm)
```

Figure 5-5: *Task Structure for Systematic Diagnosis*

; this structure can on occasion be supplemented with one for heuristic diagnosis. The

enhanced structure will make it possible to deal with complex and detailed system structures.

In the next chapter we shall illustrate the use of an IM for the financial domain of 'Commercial Loan Assessment' (also referred to as CLA) in the domain of 'Underwriting'. The example will describe the criteria we use in applying the IM for CLA to a similar domain. It will also illustrate the changes we have to make to the CLA template before it is suitable for our new application.

6. Case Study 1 - Analysis of an Underwriting Domain

6.1. Introduction

The work described in this chapter provides an example of use for an interpretation model in a real life application. The model in question is that based on the 'Commercial Loan Assessment' (hereafter referred to as CLA) paradigm, as presented in 'Models of Expertise' paper (Wielinga & Breuker, 1986). The reason for choosing the IM is the essential similarities exhibited by the application domain of "underwriting" with that of CLA. The changes to the IM reflect closely the requirements of the underwriting domain for which it has been adopted.

The chapter appears in three sections of results (the current section), history, and conclusion. We find the distinction useful, in the sense that it enables the reader to have both a formal and an informal understanding of the use of the methodology, and its impact on the use of interpretation models in general.

The IM has been used over a period of six months to provide consultancy in identifying the detailed functional specification of a proposed KBS called 'Automated Decision Support Aid' (ADSA) in the domain of *export credit guarantee underwriting*. The specification supports the problem solving in this domain, which we shall refer to as 'underwriting domain' hereafter. The case study is based on a real life example in providing consultancy to 'Export Credit Guarantee Department' (hereafter referred to as ECGD) of the Welsh Office, the largest UK organisation engaged in underwriting UK exporters. We have used KADS to analyse this domain, and as a result a 'conceptual model' has been devised for the domain. The resulting KBS is currently in real

life application in various branches of ECGD. Due to some of the commercial implications of the consultancy, the nature of some of the data in this chapter has been modified to ensure confidentiality. The modification will not, however, affect the nature and essence of the discussion put forward in the chapter.

6.2. A brief description of the Underwriting domain

The buyer division at ECGD provides an underwriting service for insuring the business of potential client exporters. In this sense when a business is underwritten, the policy holder (referred to as p/h throughout) obtains a cover against possible risks and anomalies arising from the buyer's business conduct. The underwriter, therefore, in essence is a 'buyer underwriter'.

To obtain a cover the exporter (viz. potential p/h) submits an application to ECGD, in which a description of the type of policy he requires together with some detail about the potential buyer(s), and his respective market condition is included. The underwriter is then able to consult his existing files to obtain more information on all aspects of a case. The major task of the underwriter is, however, the gathering of information on the buyer and his market conditions. The typical data obtained on the buyer concerns his business characteristics in the form of his payment record, trading history, and such like. On the other hand, independently of the buyer, the underwriter might need to assess the market characteristics in which the buyer operates.

There are several sources from which the underwriter can glean information on the buyer and the p/h. These can be agencies, embassy reports, other policy holders trading with or knowing the buyer, and so forth. The underwriter decides to guarantee a case only when the risks involved are outweighed by the merits of a case based on just the type of information discussed so far.

6.3. The Role of ADSA

The role of ADSA is to process each application and make recommendations to the underwriter (expert) whether to accept/reject an application. In highly marginal cases if ADSA cannot arrive at a specific recommendation, it will refer the case to the expert altogether. The overall aim of the buyer division is to improve radically the speed with which applications made to them are processed. This is in order to attract new business and win back business lost through unacceptable processing time currently being experienced by some policy holders.

ADSA is being applied in a number of phases, at the final stage of which it should enable the local branches of ECGD to provide a decision on a very high percentage of applications within a 24 hour period. The ultimate version of the intended KBS should also provide advice of the kind involving detailed, and informed judgements on various factors contributing to making the final decision. In very marginal cases where the system is unable to offer a decisive advice the expert will take over, with the added benefit that all of the data gathered by then, will have been processed by the system. The expert can then concentrate just on those parts which can sway his decision one way or the other, by calling on his life long expertise and feel for the market.

6.4. Principal similarities between the 'Underwriting Domain' and CLA

In deciding whether to underwrite an application or not the underwriter is confronted with issues not too dissimilar to that of CLA. The mechanism for gathering data, and the nature of data differ to varying degrees between the two domains. but the essential problem solving behaviours are interestingly similar.

On the one hand, solutions are identified for different applications in the form of a credit guarantee of a certain limit over a specified period of time. The application, itself, contains a statement of the type of 'solution' which is required by the p/h. The need on the part of the p/h to reduce business risk is identified as the 'problem' for which a 'solution' is specified. Whilst, on the other hand, a number of parameters regarding the p/h and his buyer(s) will have to be considered, to establish whether or not the underwriting 'norm' (i.e. maximum acceptable financial risk) will be observed in insuring a p/h. In cases where a decision is marginal, or some of the information in the client's file has the potential for being dubious, there is a need for considering financial evidence which could adjust the possibly inflated parameters. This problem solving behaviour has all of the essential ingredients shared by CLA and supported by the IM based on CLA. We have thus far endeavoured to show informally the procedure for choosing an interpretation model amongst the set of many, we shall demonstrate next the use of the IM in the context of our consultancy work.

6.5. Different Phases of the Consultancy

6.5.1. Domain Layer

The first thing to construct is the axiomatic structure containing concepts, relations and structures within the underwriting domain. This takes the form of producing a domain lexicon, and an 'is-a concept hierarchy' (see fig. 6-1). The lexicon and the concept hierarchy are based on transcripts from tape-recordings of the knowledge elicitation sessions with the expert. The concept hierarchy is later modified as a result of introducing metaclasses (see below).

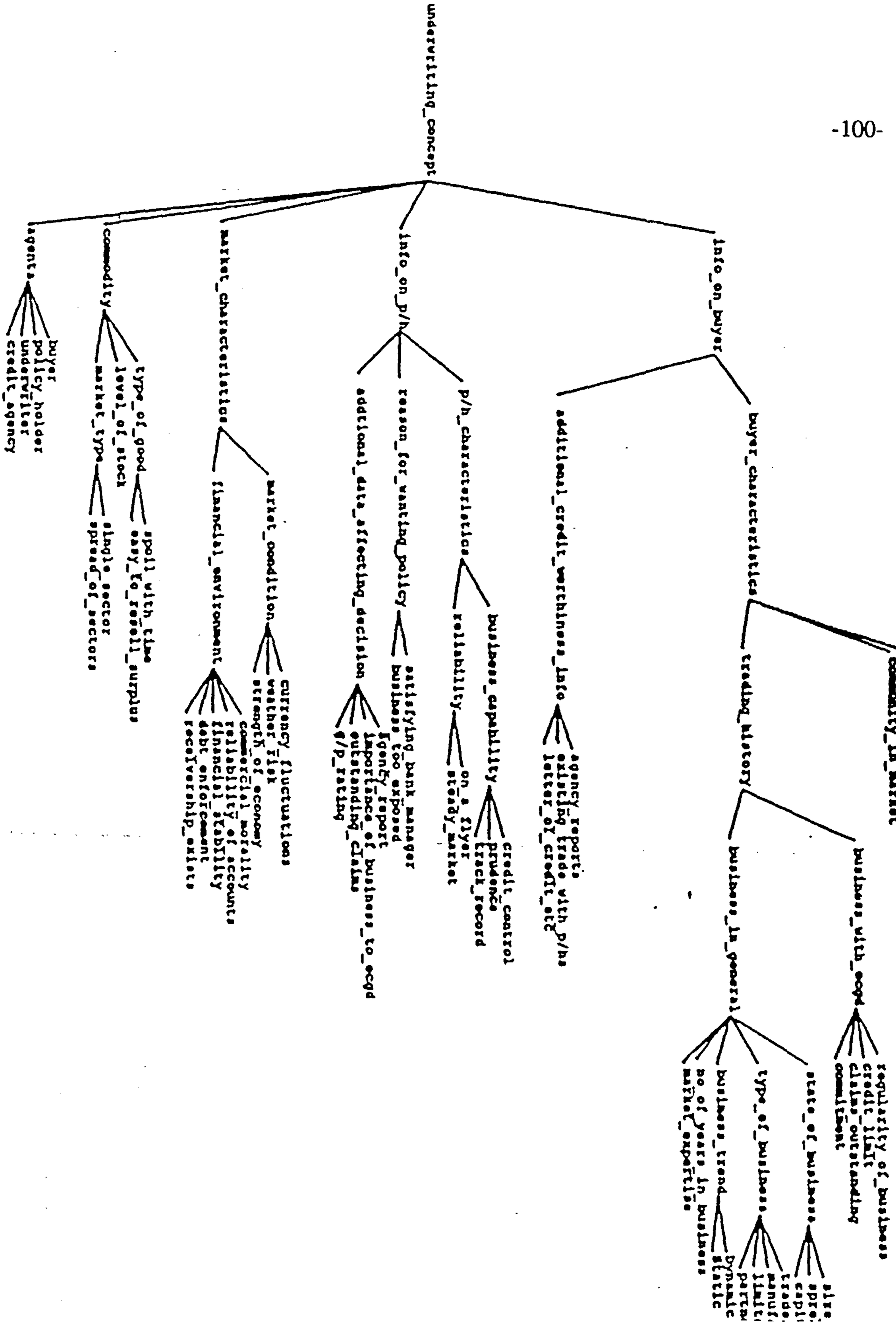

Figure 6-1: *Part of an is-a concept hierarchy describing ECGD Domain Layer*

6.5.1.1 Domain Lexicon

This contains concepts which are attributes of those parts of the system which the expert needs to consider in arriving at a decision.

6.5.1.2 Concept Hierarchy

Domain concepts are divided into five groups of: information-on-buyer, information-on-ph, market-characteristics, commodity, and agents. This division is warranted by the way the expert's problem solving behaviour comes across in terms of identifying potential sources for gathering information which should enable him to arrive at a decision (see fig. 6-1).

6.5.2. Use of IM in Analysis

The analysis should spell out the elements of the expert's (underwriter) decision making process when he considers a potential client's case. Having devised the axiomatic structure, we need to identify sources for, and branches of inference making, where these have the potential for being combined to form task structures which support the expert's problem solving behaviour. We shall, therefore, use the IM to:

- firstly, identify potential elements used in making inferences by considering the inference structure (fig. 6-2), which is the first part of the IM,
- secondly, as the next step, we shall proceed to devise a task structure which is influenced by its counterpart in the IM.

6.5.3. Problem Analysis at the Inference Level

At this level domain concepts can be mapped onto metaclasses, this will require an analysis of the roles which these concepts play in the reasoning process. In the first

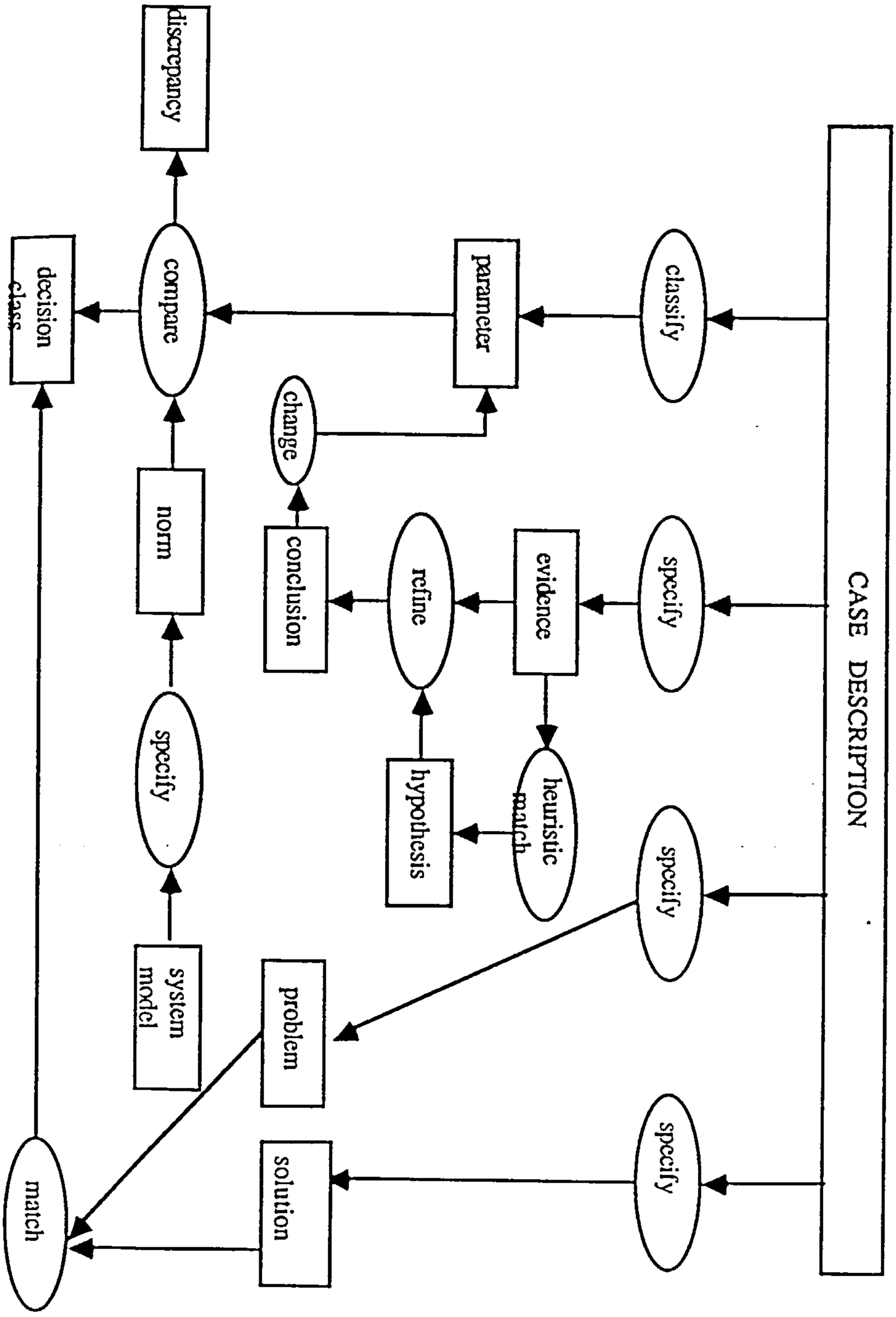

Figure 6-2: *Inference Structure for Assessment of p/h application*

instance, the client submits an application in which he expresses the need for a guarantee; the need to reduce risk in export business, implied by the application, is expressed in the form of the metaclass 'problem'. The expert is expected to provide a 'solution'

to this in the form of the type of policy to be awarded. The type of a policy may be determined in terms of credit limit, duration, and terms of payment.

Prior to applying the solution, the potential risks have to be assessed in order to ensure that the underwriting 'norm' is not violated. The norm in this case consists of a set of conditions, some of which will have to be present as part of the metaclass 'parameter' before a positive answer can be given. The norm in our case may consist of one or a combination of the following clauses:

- buyer honesty is high & policy holder competence is also high.
- buyer-exposure & buyer-market-condition are not below certain threshold & p/h competence is above average.
- if buyer-exposure & buyer-market-condition are both marginal, then buyer-competence is above average & p/h competence is high.

The metaclass 'parameter' requiring similar attributes as that of 'norm' may consist of:

- buyer-honesty
- buyer-competence
- buyer-exposure
- p/h-competence
- buyer-market-condition

The member concepts of the metaclass 'parameter' will have to be combined using and/or connectives to enable comparison against their counterpart clauses in the 'norm'. The satisfaction of one or more of the 'norm' clauses will result in a positive decision to a potential p/h application.

As with CLA, we expect to find a 'diagnostic' branch, in order to adjust those components of the parameter which might have been inflated in either direction. The sources acting on behalf of the underwriter can obtain additional information providing the required further 'evidence' in its pure (raw) form. The 'evidence' can then be 'refined' to form the metaclass 'conclusion' which is a measure of error (or exaggeration) on concepts within the metaclass 'parameter'. This can then be used to adjust the inflated values to enable a more factual assessment of an application.

The ks 'refine', strictly speaking, is not a primitive operation. It should be expanded into knowledge sources 'abstract' and 'transform'; we have however found the extended version rather too detailed for our purposes here. Note that operations taking place on metaclasses, in fact, apply to concepts under those metaclasses. Similar to CLA three distinct branches of inference making can be identified within the inference structure:

(1) *Problem and Solution Identification*

(2) *Risk Assessment*

(3) *Diagnosis*

The major metaclasses and their member domain concepts are listed below (fig. 6-3). The major domain relations are: 'quantitative' (eg. in formulas to calculate parameters), 'consist-of' between p/h's / buyer's property and market sectors and financial attributes. The knowledge sources employ these relations in order to perform different operations which yield desired metaclasses that can be used in making a decision. The required knowledge sources can be read from the inference structure (fig. 6-2).

meta class	domain concepts
case description	buyer attributes, p/h attributes, market attribute, commodity, agency reports
system model	underwriting system
problem	risk reduction
solution	type of policy
parameter	buyer_honesty, buyer_competence,.....
norm	underwriting terms (see page 4)
decision class	yes/no/refer (ADSA refers highly marginals cases to the expert)
evidence	additional info on buyer/market condition, etc. (gathered thro' agent)
hypothesis	unreliability of data
conclusion	correction value
discrepancy	difference between 'parameter', and 'norm'

Figure 6-3: *Metaclasses for decision making in Underwriting Domain*

6.5.4. Problem Analysis at the Task Level

The expert, in our case, turns out to follow an overall problem solving strategy in which there is room for maneuver in individual cases. The strategy deduced is the result of thinking aloud protocol (see chap. 3), some of the detail of which is alluded to in an earlier section.

When an application is submitted, the underwriter has to make a judgement on whether or not to insure the p/h, who as an exporter sells to buyers from overseas. It appears that the underwriter cannot exercise much flexibility in providing a guarantee (solution). This is due to the nature of the type of insurance involved; namely, the p/h needs to have a large enough credit guarantee over a fixed period of time to cover his business. The underwriter is, therefore, required to assess the financial merits of a case very carefully, since he cannot modify the terms of the application to any great extent. Certain data on p/h and the buyer, in particular, can have near to conclusive

effect on a decision being made. On the buyer's side, documents such as 'letter of credit', or characteristics such as 'good business morality' are the kind of data which can greatly influence the underwriter in giving a positive answer. In short, if the underwriter can make a clear yes/no decision based on the initial data supporting a case, then he is likely to do so without considering other potential 'evidence' likely to have a superficial effect.

In marginal cases, on the other hand, the underwriter will need to rely on his other sources (agencies, embassy reports, etc.) to gather enough evidence (or show lack of it) to correct errors of judgement which might have inflated the 'parameter' in either direction. The diagnosis part of the inference structure is then used extensively to generate the measure of error ('conclusion') needed to adjust the 'parameter'. On the whole, we find the problem solving behaviour in our case greatly resembling that in CLA.

The task structure (fig. 6-4.) combines the inferences shown in figure 6-2 in

```
assess(case, decision_class)
        obtain(case_description)
        match(solution, problem)
                specify(problem)
                specify(solution)
        assess(risk)
compare(parameter, norm)
        specify(norm)
        classify(case description)
while discrepancy unaccepatble
        confirm_or_deny(hypothesis)
                specify(evidence(discrepancy))
                        obtain(new case data)
                heuristic_match(evidence)
                refine(hypothesis, evidence)
                change(parameter)
```

Figure 6-4: *General Task Structure for ADSA*

order to achieve the objectives described in the strategy. If the outcome of issues having a bearing on a decision remain very marginal after all of the branches of the task structure have been considered, then ADSA makes no recommendation and refers the processed case to the expert (a low percentage of cases). The more information is required for the assessment of risks, the deeper the branches are evaluated.

We have not made any provisions for a flexible strategy layer for this domain, since the task structure , though lacking total flexibility, is of general enough nature for our purposes in the consultancy work.

6.6. History of Development

6.6.1. Introduction

This section is intended to provide an informal account of the consultancy process. This should help to provide the users of the interpretation library (see chap. 5) with an insight into using an IM, and the possible problems they may encounter in doing so. None the less, we do not claim that in providing the history of development in our case, all possible 'how's' and 'what's' of using interpretation models will be answered. Nor can any other singular use of such models provide those answers. On the other hand, we detect that there possibly are enough interesting (or essential) similarities between the use of the IM in our case with the use of other interpretation models to provide a handle for the use of such models in general. Some of the problems that we shall discuss are peculiar to the credit assessment in the domain of underwriting, whilst some of the others are of a general enough nature to be used as, say, "problems to watch out for in using an IM".

In describing the history of the work, we ascend from an understanding of the domain

and tasks involved to describing a process structure which could support those tasks; the route is via an analysis of the tasks. We, therefore, find it useful to maintain an ordering for our description which reflects the sequence in which the consultancy work is conducted.

6.7. Task Identification

Prior to receiving and studying the transcripts from the knowledge elicitation sessions, we are introduced to the notion of 'underwriting' in the context of ECGD. Our understanding of the issues involved in underwriting grows, but not always in a top down manner. The reading of the first batch of transcripts starts with a partial, and to some extent misconstrued, understanding of the domain. A better model of the underwriting domain has begun to emerge after reading part one of the transcripts. A lot of credit for this must go to the expert who is both eloquent, and disciplined in the way he answers questions and volunteers information which to varying extent is of relevance. This is despite the fact that the expert, in trying to be helpful, sometimes goes into too much detail. The depth of detail changes from being a disadvantage to something of an advantage on the second reading of the transcript. This is due to the fact that by now we have a much better appreciation of the domain and are able to use detailed explanation to focus on the relevant points in identifying the tasks involved.

The major misconception we have had after reading the first batch of the transcripts has been one of confusing a lot of data on the buyer for that on the p/h, and to a much lesser extent the other way round. The major reasons behind this are twofold. Firstly, both the exporter (p/h), and the buyer have a lot of financial attributes which are of similar nature, for the obvious reason that they are both traders. Secondly, our partial and somewhat misconstrued view of the domain could not have helped. In a sense, the latter is greatly aggravated by the former. A lot of problems of this nature can

also be attributed to the fact that, as consultants, we are not present in the knowledge elicitation sessions; and, therefore, we are unable to benefit from the 'psychology' of the process. To bring this point home we shall describe a possible scenario.

Imagine a speaker who is comparing two issues (points of view, etc.), he starts using his right and left hands in order to refer to those issues by allowing them separate spatial existence. This is often used to focus the attention of the audience by the use of 'body language'. The audience might be clear about the speaker's way of distinguishing between the two issues in this way; whereas someone reading the transcripts of the scenario might be reduced to making conjectures to arrive at a potentially incorrect distinction. This is particularly true if the two issues are intrinsically similar. Being an audience, in this sense, brings with it just the kind of 'psychological' advantage we alluded to earlier.

We have been able to adjust our view of the data on the buyer and the p/h to reflect reality, after having a further consulting session with the actual interviewers responsible for eliciting expert's knowledge in the first place. Having completed the reading of the rest of the transcripts, we are able to tune our view of the domain, and, also, enrich the lexicon, which we have had partially developed by then.

We have chosen the domain concepts against two major criteria. The first of these are the tasks we have had identified by then as part of the problem solving behaviour in the domain; these have provided us with constraints (or framework) for identifying the relevant concepts. The second criterion, on the other hand, is more intuitive, in the sense that there are a number of concepts which have either the potential for being utilised as part of the problem solving process, or that they seem too financially oriented to be ignored. We have been working on the assumption that it is better to have redundant concepts to start with, which can be eliminated later, than to overlook data by not including a number of seemingly unimportant concepts. We have been

able to revise our lexicon after reading the total transcript a second time, at which point we have also been able to devise a rough version of an 'is-a concept hierarchy'. The hierarchy (see fig. 6-1) provides us with a classified view of the domain enabling us to talk about different aspects of it without having to enter into an unnecessary depth of detail. We have also had a view (based on previous experience; cf. Davoodi, 87[a]) that we could use the hierarchy as a way of clustering together those concepts within the domain which can be regarded as potential units for holding information on the domain. The hierarchy was rather rudimentary at this stage; it was, subsequently, revised as the result of selecting a number of metaclasses. This is a process to which we shall allude in a later section.

6.8. Why use the IM

We had started the consultancy with a feeling that the IM supporting CLA is likely to be of use in the case of ADSA. The fact that the IM was based on a financial application, coupled with the simple reality that it was the only one around of its kind, was reason enough to put the IM to test. We find that our intuition about the IM's potential use has been a reasonable one, as we have identified (sub) tasks in the 'underwriting domain' which are similar to those in CLA. As the nature of the underwriter's decision making has become more and more clear to us, we have come to realise that there are also essential similarities between high level tasks involved in the two domains. The major difference between the two domains is the mechanics of gathering data, this is the auxiliary part of the process, thus having no impact on the way decisions are made.

We can distinguish two important tasks that the underwriter will need to carry out before making a decision. Firstly, he has to assess the risks involved to make sure that:

- he does not take too much risk in underwriting a case, and
- he does not reject a case because he is not prepared to take *some* risk.

In short, he has to get the balance right to earn ECGD profit without overtly exposing it. Secondly, the underwriter is able to call on his potential sources (ie, agencies, embassies, and so forth) to gather information, when he needs to assess the risks more thoroughly, especially in processing marginal cases. The same information may, also, be used in order to provide further data on a client's case which seems (to the expert) as not having been sufficiently presented. These tasks show essential similarities to those in CLA, it is now time to consider how we can use the IM to analyse the tasks we have already identified. This can be done by considering the inference structure (fig. 6-2), and seeing how its 'elements' apply to our domain.

6.9. Task Analysis

The tasks identified thus far are matched against the principal branches of the inference structure within the IM. The first of these is that of assessing risks involved in underwriting an application submitted by the p/h. The first thing to consider, now, is the forming of metaclasses comparable to 'parameter' and 'norm'. The concepts making up different parameters are classed together, so that they could be referred to collectively. The classification is based on the type of role the concepts play in the reasoning process, namely, providing the kind of information needed for the assessment of risks. On the other, the same process can be pursued in order to define the metaclass 'norm' in terms of its member domain concepts.

It could be argued that the process of defining 'norm' should precede that of 'parameter', since the former will establish what we should look for in the latter. In reality, one finds that defining parameters first comes more naturally, since one can speculate

about the sort of classification involved in order to identify concepts making up the extension (set of all members) of 'parameter'. The identification of 'parameter' should also provide us with a handle for deciding the 'norm'. The 'norm' will have to be checked and negotiated with the underwriter through the knowledge engineers, in order to ensure that it contains the correct attributes. We can also conceive of domains in which norms are defined very clearly; and, thus, the 'parameter' can be established in an, almost, top down fashion, using the 'norm' as the model.

In negotiating the 'norm' with the underwriter, we find that despite the 'norm' containing the right ingredients, it is important that it is pitched at the same level of abstraction as that which the underwriter considers. That is, on the one hand, the expert does not seem to appeal to what seems to be relatively low level concepts when he makes a decision. Whilst, on the other hand, he seems to be using a combination of concepts as clauses which on their own can individually indicate the amount of risk acceptable in different situations. In order to ensure compatibility with the underwriter's view of the 'world', we shall have to make certain that concepts participating in the 'parameter' and clauses of the 'norm' are of the same level of abstraction as those which the underwriter uses. The concept hierarchy depicted in figure 6-1 has been used constantly in order to adjust and refine our perception of the extension (membership) of 'parameter' and 'norm' against that of the expert's.

Meanwhile, we are also able to enrich the concept hierarchy by including new high level concepts near the root of the hierarchy, which we had not thought of when we were devising the domain layer. The process of refining and enhancing 'concept hierarchies' is a typical one which we expect to take place in almost all applications of KADS using the hierarchies.

It is interesting to note that an 'is-a hierarchy', like the one we have constructed, may have a number of possibilities for representing "which is" an instance of "what". At

the inference layer, one is able to constrain the possibilities by introducing metaclasses near the root of the hierarchy, thus grouping together concepts in a highly organised fashion. Most of the metaclasses happen near the root of the hierarchy providing a *clustered* view of domain concepts, which will depend upon the role they will play in reasoning.

We have, meanwhile, identified a new metaclass named 'discrepancy' which would hold the difference between the constituent concepts of norm clauses with their counterpart concepts within 'parameter'. This metaclass is not present in the IM and has had to be added by us, we can only assume that prior to its inclusion it was being used ,within the IM, implicitly. 'Discrepancy' will act as a triggering factor in reasoning, which might invoke gathering of further evidence through the diagnosis branch of the inference structure.

The knowledge sources *classify*, and *compare* introduce a useful constraint on thinking about the format of the metaclasses 'norm' and 'parameter'. In other words, 'parameter' is the outcome of *classifying* concepts in 'case description'; the format of the classification process is further constrained by the fact that 'parameter' should be comparable with 'norm'. The philosophy of using ks's as constraints to reduce possibilities for metaclasses has assisted us significantly throughout the analysis process. In this sense, we find the new version of the inference structure (fig. 6-2) much more to the point, by representing an *appropriate* and more explicit view of the kind of primitive subtasks involved in underwriting. The changes made to the inference structure within the IM have been as a result of further advances in devising ks's within KADS, as well as constructing a bespoke structure for the underwriting domain.

In the marginal cases when the underwriter is unable to make a clear yes or no decision, he will have to consider a second level of assessment. There are two distinct occasions in which this can happen. The underwriter may decide, on the one hand,

that the data supporting 'parameter' is insufficient, in which case he has to consult his sources for more data of the relevant kind. Whilst, on the other hand, he may have some cause for doubting the reliability of the support data, in which case he is, again, prompted to look for 'evidence' which may confirm or reject his original doubts. In both these cases, the 'evidence' is used ultimately to possibly 'change' the parameter. This type of behaviour, we find, concurs with the 'diagnostic' branch of the inference structure. As before, ks's are used as constraints for defining the format of the remaining metaclasses. Finally, the metaclass 'hypothesis' is used as a measure of 'heuristic match' (educated speculation) in 'refining' the 'evidence'.

One of the interesting aspects of using the inference structure is the way in which it has helped us in organising our thoughts by keeping ideas in our minds separate, and identifiable from one another. The prime example of this is the use of the 'problem/solution identification' branch which once used in a task structure, it establishes a logical order of priority. That is, one should assess risks after clearly identifying the 'solution' to a proposed 'problem'. Otherwise, one is likely to end up with a confused view of the world in trying to establish a 'solution', whilst also assessing the client's financial 'parameters'. Despite the apparent obviousness of the argument, it is not clear that one could reason so parsimoniously and clearly, when one is confronted with a new domain in which problems of getting to grips with the expertise may be multifarious.

6.10. Process Structure

After the task analysis, the next natural step would be to devise the task structure and possibly a flexible strategy structure. The former is shown in figure 6-4, in which sub-tasks of problem / solution identification, assessment of parameters, and diagnosis are sequenced respectively. The diagnosis branch of the task structure may be over-

looked in cases, where the assessment of 'parameter' against 'norm' strongly suggest a "yes" or "no" decision.

We have not devised a 'flexible strategy' layer, since the task structure seems general and flexible enough for our purposes.

6.11. In Conclusion

Prior to any attempt to use an interpretation model, the user of the library of interpretation models (see chap.5) should be able to answer two questions,

- how to choose a model,
- and how to apply it to the domain at hand.

Given a number of interpretation models, the user should decide whether any of the models provides for a problem solving behaviour of the kind in which he is interested. Since models are named sensibly, it would be the natural thing to see whether the title is one which makes sense for an application under consideration. This activity could be augmented with the reading of the description of the general task which the model supports. There may be cases in which a given model satisfies the application only in parts. The user should use that model as a partial solution, and look for other models to complement it. If he is unable to find any other models, then he has no choice but to construct his own inference / task structures for the remaining parts. He should then combine the outcome of this with the model he has already chosen, to arrive at a coherent view of the problem solving for his domain. In fact, in our case the assessment part of the IM can be viewed as a model in itself.

When choosing an IM, the user should not be disappointed with the possible fact that even the best fit model shows dissimilarities with his own application. So far as there

are essential similarities between his domain and the model, it is, then, up to the user to modify different parts of the model to fit his application domain. Having decided that a model is possibly a suitable one, the user should, then, examine the task structure and the supporting flexible strategy to find out if the model can support his application throughout. The more IMs are used, the more experience can be recorded on how to choose and apply one.

In answering the question 'how to use an IM', we can do no better than to list a number of important points from the previous sections.

- Identify the main tasks (and sub-tasks) involved in the application area, and, then, decide whether the candidate model can support them. This guideline can, also, go some way toward answering the other question of 'how to choose a model'.

- At the inference layer, identify the main inference making branches and traverse them in an order which seems natural to your thought processes.

- Establish the metaclasses by using constraints such as the name of objects (i.e., 'norm' has a particular implication in a given domain), coupled with the type of knowledge sources for which the metaclasses are input or output. It is important that the right level of abstraction is chosen for the metaclasses, otherwise they will be either too detailed and tedious, or too abstract and incomprehensible.

- Use the inference structure to organise your ideas about the problem solving behaviour, without allowing it to impose on you any unrealistic (or superficial) inferences.

- Make changes to the task structure if need be, or only use some part of it.

- Use the task structure as a guideline for the particular structure needed to support the problem solving process in the domain.

- In the unlikely event in which the task structure is totally irrelevant, it can, at least, be used as an example of how to construct a task structure by combining 'elements' from the inference structure.

The methodology allows for operating at both levels of *data model driven*, and *process model driven*. That is, we are able to switch from a view of the 'world' which is not concerned with the functionality of data, to one which *is*, whenever this is of help. Such a flexibility is not associated with conventional software development, and we regard as being a major contribution of the methodology to the field of KBS.

Nothing in the end can substitute for experience; our work here should be seen as an informal education for users of the library, and not as a 'manual' on using interpretation models. We encourage the user to record his experience in using interpretation models, and abstract from it what can be of benefit to the community of KADS users.

7. A Framework for Design

7.1. Introduction

In describing the design of a system we need to identify the 'processes' and 'components' circumscribing this creative activity. We shall refer to these as 'design processes' and 'design vocabulary', respectively. The distinction is a useful one, since processes are vehicles for transcending from design components in one layer to those in the next layer. Figure 7-1 identifies components and processes with boxes and ovals, respectively. The vocabulary consist in a set of terms whose meaning and implications should be made clear at the outset.

Design in the context of KADS uses the results of 'analysis' as the initial input (see figure 7-1). The final output is a 'structure' directly supporting the artifact. The structure should leave the developer with minor implementation decisions, decisions concerned primarily with 'housekeeping' (e.g. initialisation) and system 'tidy-up' (e.g. garbage collection).

In the ensuing sections, we shall propose a framework for KBS design, which is based on a design philosophy inspired by development in KADS to date, namely the results of 'analysis'.

7.2. A Philosophy for Design

We view design as part of the methodological spectrum, a continuum along which the output of 'analysis' should somehow transform into a workable and elegant architecture (cf. Davoodi, 1987[b]).

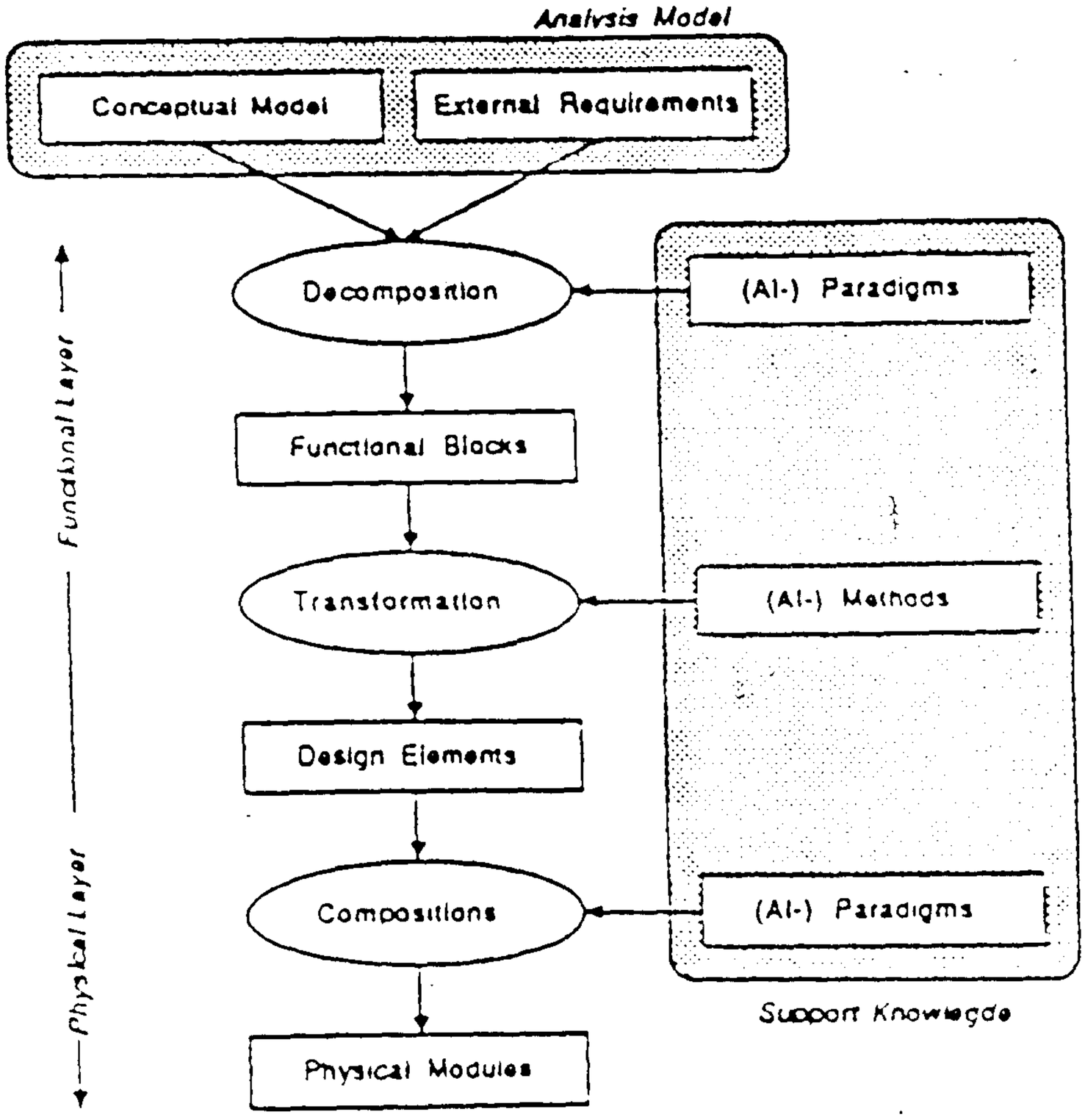

Figure 7-1: *KADS Design Process - a Bird's Eye view*

We find it useful to identify two global layers over which design will take place. The first of these captures the behaviour of the system with respect to analysis, this layer sits in the continuum immediately after analysis. It uses the results of analysis as input in order to capture 'WHAT' is required of the system. The next layer decides 'HOW' to achieve that requirement, the latter is what we call the *physical layer* of design.

At the physical layer we shall consider how to support the artifact such that the desired behaviour of the system is captured within the machine. Design, by implication, does not include the artifact (code) itself. The term 'physical', therefore, refers to immedi-

ate support for the code, and not the inclusion of it. For instance, a physical module called 'schedular' may be designated as containing a number of procedures for monitoring and allocating some system processes to different devices within a manufacturing machinery. The 'schedular' as a module will be realised within that part of code which represents the procedures the module is meant to contain.

The physical layer should also address issues to do with optimum and/or transparent support for the artifact. The degree to which these two issues are compromised will naturally depend on the constraints placed upon the designer, as well as his level of competence.

7.3. The Design Layers

We shall describe next the two layers over which design will take place.

7.3.1. Functional Layer

The functional layer is concerned with the *functional description* of the system (Davoodi et al., 1987). The functional description should encompass both the internal and external views. The two views are incorporated in a number of *functional blocks* which are the building blocks of the functional description layer.

In forming the functional description, the conceptual model is used as the starting point for the formulation of the functionality of a system. One might argue that the conceptual model should be treated as the complete functional description of the artifact, and thus no further transformation will be required. We maintain, however, that the distinction between the conceptual model and the structure consisting of the functional blocks is an important one and should, therefore, be sustained. There are a number of reasons for this distinction, the prime ones amongst which are:

- The conceptual model is concerned almost entirely with the problem solving part of a system. In particular, I/O and data storage functions are usually implied in the conceptual model, but their existence is not fully described. In the functional description these functions are explicitly identified and clearly described by decomposing them into a number of sub-functions.

- External requirements may require additional functions within the artifact to support them. An example of this might be the requirement to use an external database, which will call for an interface function allowing for communication between the intended system and the database.

- The conceptual model has a purely epistemological bias, and as such it may not be the most suitable, clear, or economical abstraction of the artifact require-ments.

7.3.1.1 Components of the Functional Layer

The basic element of the functional layer is the so called *functional block*, which represents a distinct functional unit of the artifact. Every functional block may have three types of relation with the other functional blocks, these are:

(1) *Consist-of*

Every functional block can be decomposed into a number of (including none) other blocks at a lower level of functional decomposition; this decomposition will form a consist-of hierarchy.

(2) *Input/Output*

A functional block may use output from and provide input data to one or more other functional blocks.

(3) *Control*

A functional block may control (or be controlled by) another block.

Figure 7-2 shows the 'consist-of' hierarchy of the functional blocks for the MYCIN system (Shortliffe, 1976; Shortliffe

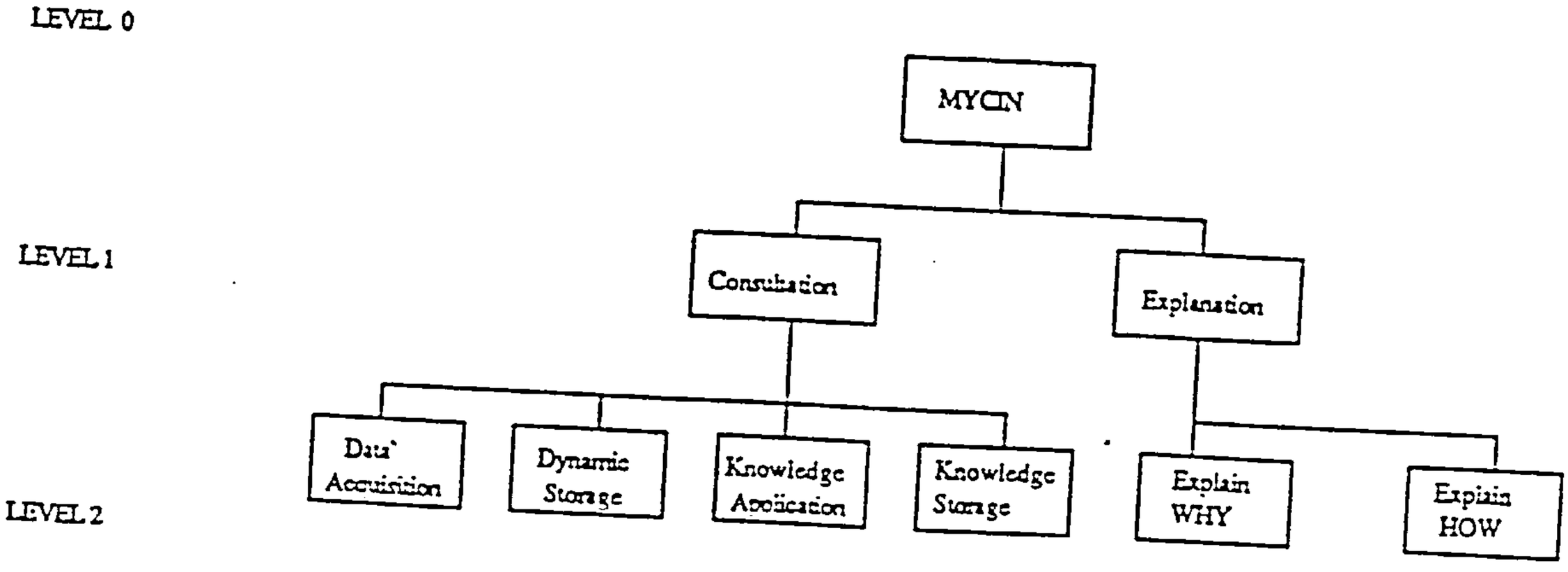

Figure 7-2: *Consist-of hierarchy for functional blocks in MYCIN*

et. al, 1979, Buchanan & Shortliffe, 1984). Figure 7-3 gives an overview of the input / output relations between the functional blocks within the same system.

The functional layer consists of the description of its individual component functional blocks, a description which will contain the following slots:

(1) *Sub function of*

The ancestor of each block in a consist-of hierarchy (if any) should be

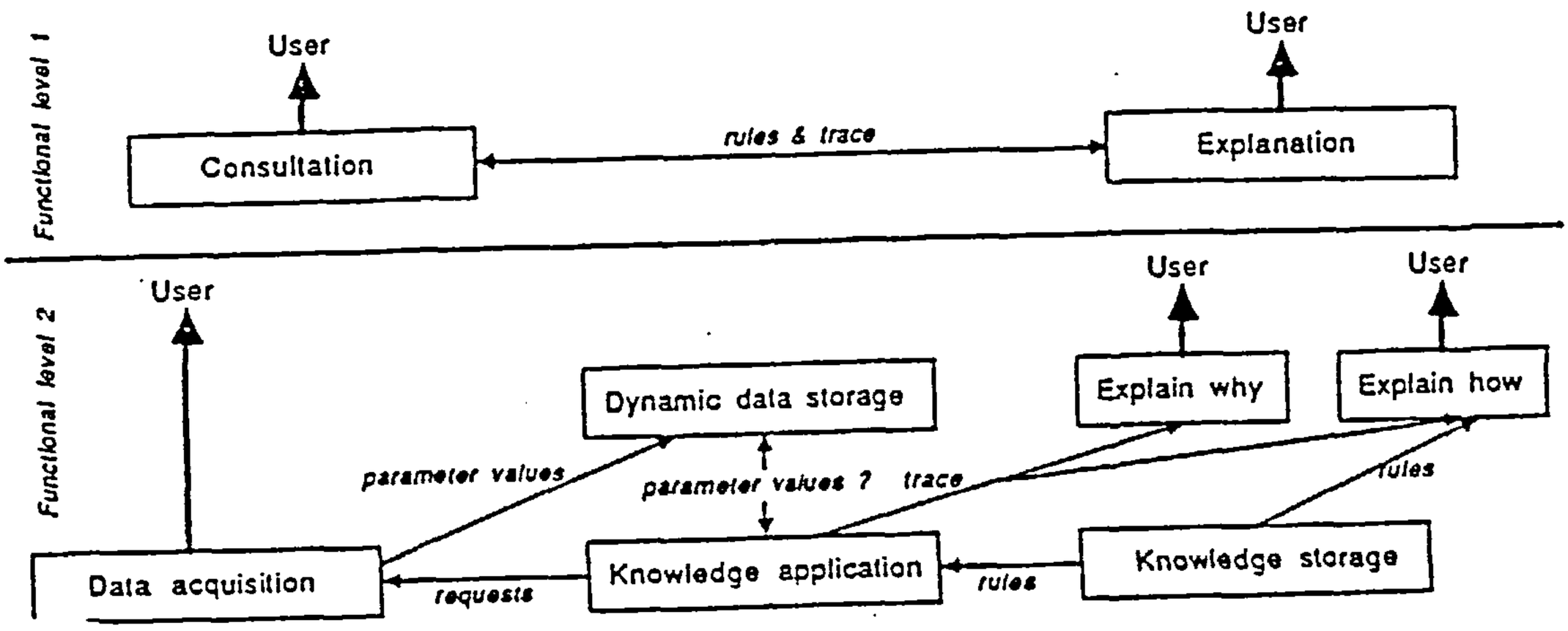

Figure 7-3: *I/O relations between functional blocks in MYCIN*

identified.

(2) *Function type*

A characterisation of the role that the function plays in the intended system should be described; for instance whether it is an I/O, or an 'explanation' function.

(3) *Input data*

A list (possibly empty) of all output data from the other blocks used as input,

(4) *Output data*

And, similarly, a list (possibly empty) of all output data used as input to other functional blocks.

(5) *External interface behaviour*

Description of the I/O of the functional block at the level of user interface and/or any other external interface.

(6) *Controlled by*

A description of the control logic associated with the activation of the functional block ('activation condition' or 'pre-condition').

(7) *Controls*

A description of the control logic associated with the termination of the functional block ('termination condition' or 'post condition'). The description so far should provide a full account of a functional block in relation to the other functional blocks. In addition, the designer should also indicate the relation between a functional block and the analysis output.

(8) *Relation to Analysis*

The part of the analysis that gives rise to the introduction of a *block*, this can be a knowledge source, a metaclass, a goal statement in the task structure (eg. 'obtain data', etc), or an external requirement.

With regards to the *function type* of a functional block, we have thus far identified the following set:

- *Problem Solving*

  Typical problem solving blocks are functions like *classify*, *match*, and *compute*. These functional blocks typically correspond to a knowledge source or a sequence of knowledge sources in the inference structure of the analysis model.

- *Explanation*

  The word "explanation" is used here in a very broad sense. The explanation-

type functional blocks may represent functions that generate a trace of the reasoning process, or explain the possibility and the limitations of the system itself.

- *Data I/O*

  These functions will allow the system to transfer data at the external interfaces of a KBS (such as: user interface, external database interface, etc.). They may be of a simple nature (i.e. menus) or be more complicated (such as 'natural language' type dialogues).

- *Data Storage*

  Data storage functions are often neglected in KBS system design; by default, the built-in database facilities of the implementation language are used. A *truth maintenance* system is an example of a more elaborate data storage function.

- *Control*

  Control functions are often introduced in order to put in place an *explicit* control unit in a system.

7.3.2. Diagramming

Diagramming techniques are used to represent the relationship between the functional blocks within the functional layer explicitly. Most of these techniques show only some, but not all, of the relationships between the blocks. It is not our intention to either recommend one technique over another or attempt at creating new ones. We shall, however, discuss the characteristics of some of these techniques, with the aim of helping the designer in applying them in various stages and types of KBS design.

- *Consist-of Tree*

  A hierarchical tree of functional blocks, thus describing the consist-of relations.

- *Data Flow Diagrams*

  Data Flow Diagrams (DFDs) show the input-output relations between functional blocks. The MYCIN example (fig. 7-3) above is in fact shown in two (non-standard) DFDs at consecutive hierarchical levels. A DFD, however, is incapable of representing the control mechanism explicitly; in such cases control is implied in the flow of data. A DFD may, none the less, be enhanced with a diagram(s) depicting the control mechanism.

- *JSD*

  The Jackson Structured Design (JSD, Jackson, 1975) technique is useful for illustrating the control relations amongst the functional blocks. The control as defined by the use of 'control statements' within the task structure of the conceptual model is rather naive. If the designer is, however, satisfied with the level of representation of control as depicted in the task structure, JSD is, then, a good candidate for translating such statements into their counterparts in the design. A major disadvantage of JSD is its lack of ability in representing parallelism and delegation of control to local processes.

- *Petri Nets and SADT Diagrams*

  Both SADT diagrams and Petri Nets can depict input-output and control relations. SADT diagrams can also show explicitly the consist-of relations amongst blocks. Petri Nets can represent the consist-of relations amongst blocks in a similar fashion to DFDs. Petri Nets are, also, particularly powerful in representing control structures using paralielism amongst the processes they control.

7.3.3. Selection of Methods

The selection of methods constitutes the second part of the functional design layer, methods specify the way in which functions are realised within the artifact. Our use of the term *method* usually applies to artificial intelligence (or heuristic programming) processing methods such as natural language interpretation, search algorithms, classification using *subsumption* and/or *refinement* hierarchies, truth maintenance and so forth. This, of course, is not to the exclusion of conventional data processing (dp) methods, such as data retrieval and depositing, or relational database management methods.

The selection of methods is one of the most crucial design decisions, and a major point of departure between the conventional and KBS (or AI) design. In the first generation knowledge based systems the selection of methods was implied in the deployment of a particular shell (e.g. EMYCIN). In the second generation KBS development, the way should be open to use a combination of various methods in a KBS in a much less constrained fashion.

The major distinction between the first and second generation KBS is in representing knowledge itself. The main concern of the first generation of KBS has merely been one of problem solving. The static and dynamic / reasoning knowledge (see earlier chapters, eg. chap. 4) are tightly coupled in such systems, a classic example of this is MYCIN. In the second generation KBS, there is a conscious separation between the domain facts (static knowledge) and the reasoning / procedural knowledge. The reasoning knowledge may at times be refined even further by having, say, 'procedural', and 'strategy' (i.e. meta-reasoning) knowledge represented separately. A classic example of this is NEOMYCIN, in which reasoning knowledge is globally shared amongst different diagnostic procedures, and applied collectively to a module called 'domain knowledge' which contains aetiological taxonomies (see chap. 8). The second generation KBS are particularly powerful means of transferring expertise in teaching

sessions. They are much easier to maintain than the first generation, since different components of the system can be modified and enhanced in a semi-independent fashion. It will also take less code to write such systems, since the reasoning procedure is global, and local replications are eliminated.

A method should be seen as an abstract entity, whose realisation in the artifact will require a number of components we have come to call *design elements*. For instance, a search method such as the A* algorithm (cf. Winston, 1984) requires a procedure to implement the algorithm, state operators, and heuristics estimates. We maintain that the distinction between the abstract method, on the one hand, and the design elements used in implementing it, on the other hand, is an important one. In the A* example it would be a mistake to consider the procedure and the search method as one and the same entity, since the method will require more than just the procedure. In Figure 7-4 a list of a number of methods and their corresponding design elements is provided.

Table 3-1: Methods and their Design Elements	
Method	*Design Element*
Hierarchical Classification	Classifier Subsumption relations Class definitions Attribute value pairs
A* algorithm	Search procedure State operators Heuristic estimates State description Start state Goal state
ATN parsing	Parser ATN grammar Lexicon Text string Parse tree
Production system	Rule Interpreter Production Rules

Figure 7-4: *Methods and their Design Elements*

A method is described using the following slots.

(1) *Description*

A textual description of the method,

(2) *Reference(s)*

References to the (AI) literature in which the method has been portrayed or used prominently,

(3) *Realises*

The functional block(s) which are realised using the method,

(4) *Design Elements*

And, lastly, a list of design elements which are required in realising the method in the artifact should be provided. For every design element, the relationship between the design element and the analysis output should also be provided (see below).

7.3.3.1 Design Elements

Design elements point to elements within the conceptual model or components of those parts of the system satisfying the external view. For example, the state operators used within the A* algorithm find their counterparts in the domain layer of the conceptual model for this search algorithm. In the travelling salesman problem (cf. Winston, 1984) state operators point to the 'road (City1, City2)' relations within the domain layer of a perceived four layer model for that problem. A design element is, therefore, described in terms of:

- *Name*

  The name of the design element refers to the role played by that element in satisfying the corresponding method, this name is domain independent. For

instance, the use of a procedure in implementing the A* method transcends its use in the 'travelling salesman,' or any one particular domain.

- *Relation to the Conceptual Model*

  A description of the relation between the design element and its counterpart within the analysis output, such as a domain relation, a knowledge source, or a metaclass. For instance, a design element may be a procedure for classifying a group of input variables to a number of output parameters. It should be possible to associate the procedure to a knowledge source called 'classify' in the 'inference structure' within the analysis four layer model (see chap. 4).

It is difficult to provide a classification of design elements; we can, however, provide a number of classificatory distinctions between them:

(1) *Active vs. Passive*

Active elements use or consume passive elements, e.g. a *classifier* (active) uses a set of *subsumption relations.*

(2) *Knowledge vs. Data*

The validity of *knowledge elements* are situation independent, whereas the same is not true for *data elements.* For instance, the ATN parsing method requires two knowledge elements of 'ATN grammar,' and 'lexicon,' as well as the data elements 'text string,' and 'parse tree'.

(3) *Dynamic vs. Static*

Dynamic elements will undergo changes throughout the life of a system, whereas static elements remain unchanged. The distinction between dynamic and static elements may seem similar to that between data and knowledge elements. This, however, is not true; the difference between data and knowledge elements is 'real world' oriented; whereas, the distinction between dynamic and

static design elements is system oriented. In a 'learning' system, for instance, the changing knowledge base is characterised by its counterpart design components which are, in turn, dynamic.

Methods typically require a procedure, representing the associated algorithm, and a set of dynamic and static elements.

7.4. Physical Layer

7.4.1. Architecture

The physical layer takes the functional description and the selected methods with corresponding design elements as input; the generated output is a set of *physical modules*. The design elements are aggregated in the physical modules, where every module may be decomposed into a number of sub-modules. We shall call the structure (or hierarchy) in which physical modules ultimately appear, the *architecture* of the artifact. The purpose of this architecture is to provide the immediate design support for the detailed implementation, but it will stop short of including the code itself.

The difference between the functional description and the architecture lies in the organisational principles underlying these two descriptions of the artifact. The functional description is primarily concerned with capturing the system requirements as prescribed by the output of analysis. On the other hand, the *architecture* consisting of physical modules provides a refined and optimised representation of the same requirements amenable to immediate implementation. In the functional description the functional blocks and their associated methods are identified, and no effort is made in optimising the functional block hierarchy configuration. The functional description, in this sense, provides the logical system design.

The architecture is a representation of the system whose relation with the analysis output is not immediately obvious, since its concern is an elegant, optimal and working implementation. This is not to say that the architecture is not driven by the analysis output; rather that the missing link between the architecture and the analysis output is the functional description. Or to put it in a different way, the logic of design can be observed in the functional layer, and its working in the architecture. Some of the organising principles underlying the architecture are:

- *Avoid Redundancy*

  If certain behaviour, data, or knowledge is shared across a number of physical modules, then capture it within one physical module to which all other modules will have access.

- *Minimise Coupling and Maximise Cohesion*

  Processes contained within or across physical modules should be loosely coupled providing for a truly modular design which yields to independent testing of its components (cf. Yourdon, 1978). On the other hand, elements within a module as far as possible should be of the same type, thus maximising cohesion within the modules (cf. ibid.). The latter should make the system much more transparent in terms of the behaviour of its individual modules and submodules. In short minimising coupling and maximising cohesion should, on the whole, provide for a good system design across domains.

- *Incorporating Trade offs*

  In cases where constraints are placed on the designer in terms of implementation vehicles or hardware to be used, the architecture should adopt a configuration suited to those constraints. Of course, in such cases it is possible to have two versions of the architecture; one free of the constraints, and the other incorporating those constraints. The reason for this is *reusability* of the

design, in case the system were to be designed constraint-free, or under a new set of constraints.

- *System Management*

  One or more modules may be dedicated to managing the invocation, system tidy up (such as garbage collection), or housekeeping (such as system initialisation), and placed under the control of one (or more) system manager modules.

7.4.1.1 Components of a Physical Module

A physical module is defined in terms of:

(1) *Description*

A textual description of the module outlining its functionality,

(2) *Sub-module of*

Its 'father' module in the modules' consist-of hierarchy,

(3) *Design Elements*

A list of design elements aggregated and used within the module,

(4) *Composition Principles*

A description of the overall mode of aggregation of the design elements within the module, such as a network of hierarchies, or a simple is-a (refinement hierarchy),

(5) *Access Ports*

A description of access ports which are handles for bringing modules into life from other parts of the system, these should not be mistaken with detailed interfaces which might take place amongst modules. As a possible example, we may imagine a control module which decides to activate a task module by

interpreting a piece of information (code) within that module. The piece of code is the access port of the module which might outline the conditions under which invocation of the task module is appropriate,

(6) *External Interface Behaviour*

And, finally, a description of the interaction of the module at the level of user interface or any other external interface.

In the next chapter we shall examine NEOMYCIN (Clancey 1985) with a view to carrying out a post-hoc study of it concentrating on the design of the system. In the same chapter the relationship between the analysis output and design will be clearly illustrated.

7.5. Discussion

In different places in the AI literature (Bobrow, 1984; Newell & Freedman, 1971; Sembugamoortly & Chandrasekaran, 1984; Chandrasekaran, 1987) a distinction is made between (1) the *physical structure* of a system, (2) the *behaviour* of a system, and (3) the *function* of a system. They refer respectively to:

(1) the actual physical objects that the system consist of,

(2) the way these objects behave,

(3) and the purpose of the produced behaviour.

The distinction is used to describe different features of a particular system in the real world, and thereby support reasoning about the system (e.g. doing diagnosis, design, etc.).

Our design framework shows resemblance with ideas portrayed by these workers. For

instance our physical and functional description will find their counterparts in these works under the same names. As for the behavioural description mentioned above, it will refer to our selection of methods and introduction of design elements which is the second part of the functional description. The major difference in favour of our design framework (or methodology) is its sense of completeness, as it is complemented by the KADS analysis phase which precedes and feeds into it. This should turn the methodology into a powerful tool both at the level of epistemological, and system modelling.

Our ideas in developing the framework has been very much in the spirit of the belief that the development of complex systems within AI will require a common framework for guiding closely the selection and utilisation of methods, techniques, languages, and paradigms available to the designer. The framework should help the designer in a number of ways, namely:

- *Rational design description*

  Design, in essence, is a creative activity in which the designer cannot be expected to pursue a predetermined route. Our design framework will, however, provide a rational way of *describing* a design, even if the designer has not quite followed that route. A further discussion of the usefulness and underlying reasons for a rational design description can be found in Parnas & Clements (1986).

- *Paradigms*

  AI paradigms can be of immense help in designing knowledge based systems, allowing the designer to choose a particular approach to design at a high level. The choice of a particular paradigm will make it possible to decide the methods, languages, and techniques pertinent to that paradigm, a mechanism which reduces the selection process amongst the many possible to those exactly relevant. This will, of course, require that the links between paradigms and

their respective methods, techniques and languages are explicitly established. The design process can also be simplified to a large extent if shells are developed that support a particular paradigm by providing the relevant methods and languages.

- *Library of methods*

  The task of selecting AI methods could be a difficult one, particularly in the face of a constantly increasing collection of such methods. Efforts have been made in bringing some structure to this growing collection, the notable example of which can be found in Bundy (1985) putting forward a catalogue of AI methods / techniques. The catalogue can be usefully enhanced by relating methods with particular function types to which they might apply.

- *The role of environments*

  The number of environments (such as, embedded systems containing AI languages, editors, run-time behaviour; shells; and toolkits) for developing AI systems is also rapidly growing. Chandrasekaran (1987) observes that a major problem of existing AI systems is that they are turned to the possibilities of the environment in which they are developed. Domain knowledge is forced into the formalisms provided by the environment, as well as the likeliness that the designer will have to use the methods available within the environment despite his better judgement. For instance, if the environment provides a single world database, a truth maintenance method is not likely to be part of a system developed within that environment. We believe that the design framework presented in this chapter will allow the designer to choose an appropriate environment for the domain at hand, rather than having to start from some given environment.

- *Explanation*

  First generation expert systems are only capable of providing explanation by paraphrasing their code (e.g. MYCIN). It is now commonly accepted that this, in itself, is not sufficient for an understanding of the reasoning process of a given KBS (see Clancey 1983; Neches at. al. 1985). We address this issue, within our design framework, by downloading the design model within the artifact such that the explicit links between system components can be (almost) readily read from the code. This should make it possible to explain the reasoning behind the system behaviour in a more explicit and transparent way by referring back to the design model. It could also provide the ultimate link between the artifact and the analysis model.

- *Maintenance, Refinement, and Debugging*

  The framework is also useful when it comes to enhancing, changing and/or refining an existing KBS; this being particularly the case if the artifact has the design model explicitly represented within it. The reasons behind this are similar to those in the previous item in the list, this position will, however, require further investigation.

8. NEOMYCIN

8.1. Background

NEOMYCIN is a medical consultation system evolved over years, its domain of expertise is "neurological signs" or "headache and fever" (Clancey and Letsinger, 1981). NEOMYCIN's knowledge base is developed by extending and reorganising MYCIN's (Shortliffe, 1976) knowledge base. A change which is reflected in the underlying "EMYCIN system" (van Melle, 1979). This is in order to make possible the diagnosis of a much larger set of causes of disorders; more importantly, this makes the knowledge base suitable for use by GUIDON, a teaching program.

The overall diagnostic strategy of NEOMYCIN is one of posing tasks which will have some structuring effect on the working memory. Requests for findings are generally intended to redirect (bring in new focus), or to confirm (give confidence to) what the system is considering. The working memory (differential) is a convenient and necessary medium for updating the hypotheses generated by the system. A mechanism which should, in the end, isolate the root cause(s) of a complaint (or system fault).

Initial information supplied to the system brings it to an intermediate hypothesis in the taxonomic hierarchy. Working from the middle, the system must first look upwards in the hierarchy to focus the possibilities. It will then refine downwards by considering more specific causes. This depicts the physician's behaviour in forming a set of possibilities which will include the cause. He will then narrow down the space of possibilities to a small treatable number.

8.2. Analysis

8.2.1. The Four Layer Model

We should be able to abstract a KADS four-layer model from the description of the system (see, for instance, Clancey, 1984-5). We are assisted in this by the clear distinction made in the system between domain and control knowledge (see below).

8.2.1.1 Domain layer

This will contain the concepts, relations, and structures pertaining to NEOMYCIN's domain of medical facts. It should be pointed out that the axiomatic structure of this layer will depend on the point of view taken when it comes to examining the structure between concepts. For instance, disorders are organised in 'refinement' hierarchies without any reference to their anatomical dependencies (ie, independently of body, or device). Such a view may obviously be replaced or enhanced by structures which relate disorders explicitly with (make it local with respect to) different parts of a device (body).

At the highest level, the concepts consist of *symptoms* and *causes*. The symptoms are abstracted from hard (lab) and/or soft (circumstantial) data, symptoms and causes are referred to as *findings* and *hypotheses* respectively. Findings and hypotheses are organised into subsumption and refinement hierarchies, respectively. A distinction is made between hypotheses representing processes describing disorders, and those which refer to substances in the body (see below). There may be process features associated with disorders which would enable the diagnostician to discriminate between certain hypotheses. The features are concerned with issues like, locality or chronicity of disorder.

Apart from structural relations of 'refinement' between hypotheses and 'subsumption' between findings, other domain relations are causal links concerned with associating hypotheses with hypotheses and findings across hierarchies. A complete list of these will appear in the domain knowledge module (below). Certainty factors are used in the normal way (eg. as in EMYCIN) in order to hide causal detail. An expanded list of domain 'elements' will appear in a later section (cf "Domain Knowledge" module).

8.2.1.2 Inference Layer

Clancey (Clancey, 1985[b]) provides a general inference diagram (fig 8-1) which supports heuristic classification in general. We tend to concur with him on the use of this inference structure. We, however, should want to adopt a modified version of the inference layer, such that it reflects the KADS approach in the use of knowledge sources (cf. Breuker, et. al., UvA, Davoodi, at. al., 1987, pp. 35-40, and p. 58).

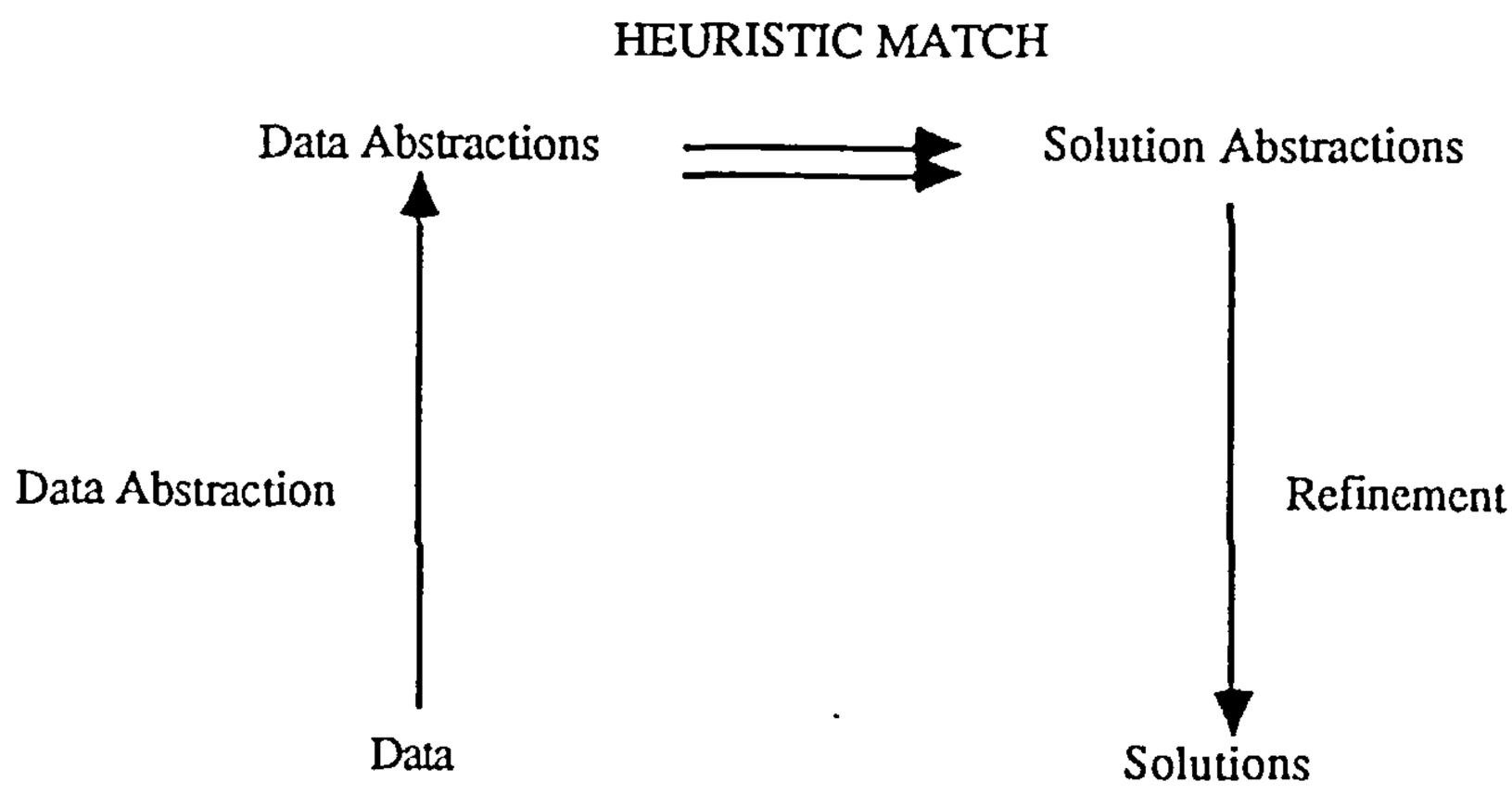

Figure 8-1: *Inference structure of heuristic classification (Clancey, 1985[b], p.296)*

Diagnosis in the broadest sense is a problem of indexation from the purported symptoms into pre-enumerated solution (cause) hierarchies, a process widely known as

classification. NEOMYCIN uses classification in conjunction with heuristic association in order to establish the causes underlying the observed symptoms in the body. The associations provide intra-hierarchical causal links amongst hypotheses as well as between hypotheses and findings.

Classification by heuristic association is implied by the use of the knowledge source 'match' (fig 8-2). The knowledge source ultimately indexes 'Patient Abstractions' into 'Disease Classes', these two being the corresponding meta-class objects. A more detailed

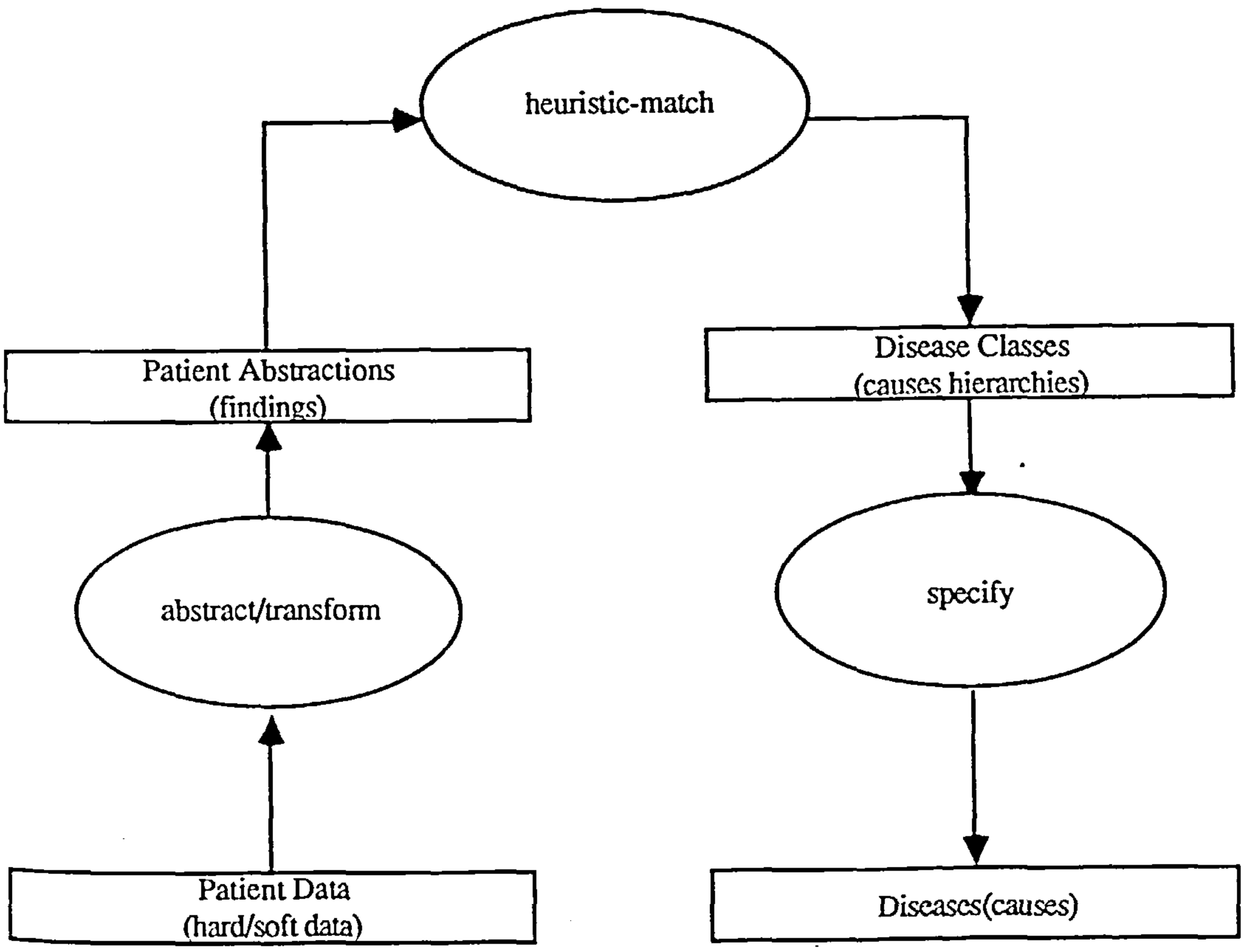

Figure 8-2: *KADS' Inference Structure of heuristic classification*

behaviour of 'match' can be seen in fig. 8-3 (Clancey, 1985[b], p.329). In turn, we need to abstract from raw data (hard/soft) on patient, that part of it which will contain the useful information. The abstracted data will then have to be transformed into a

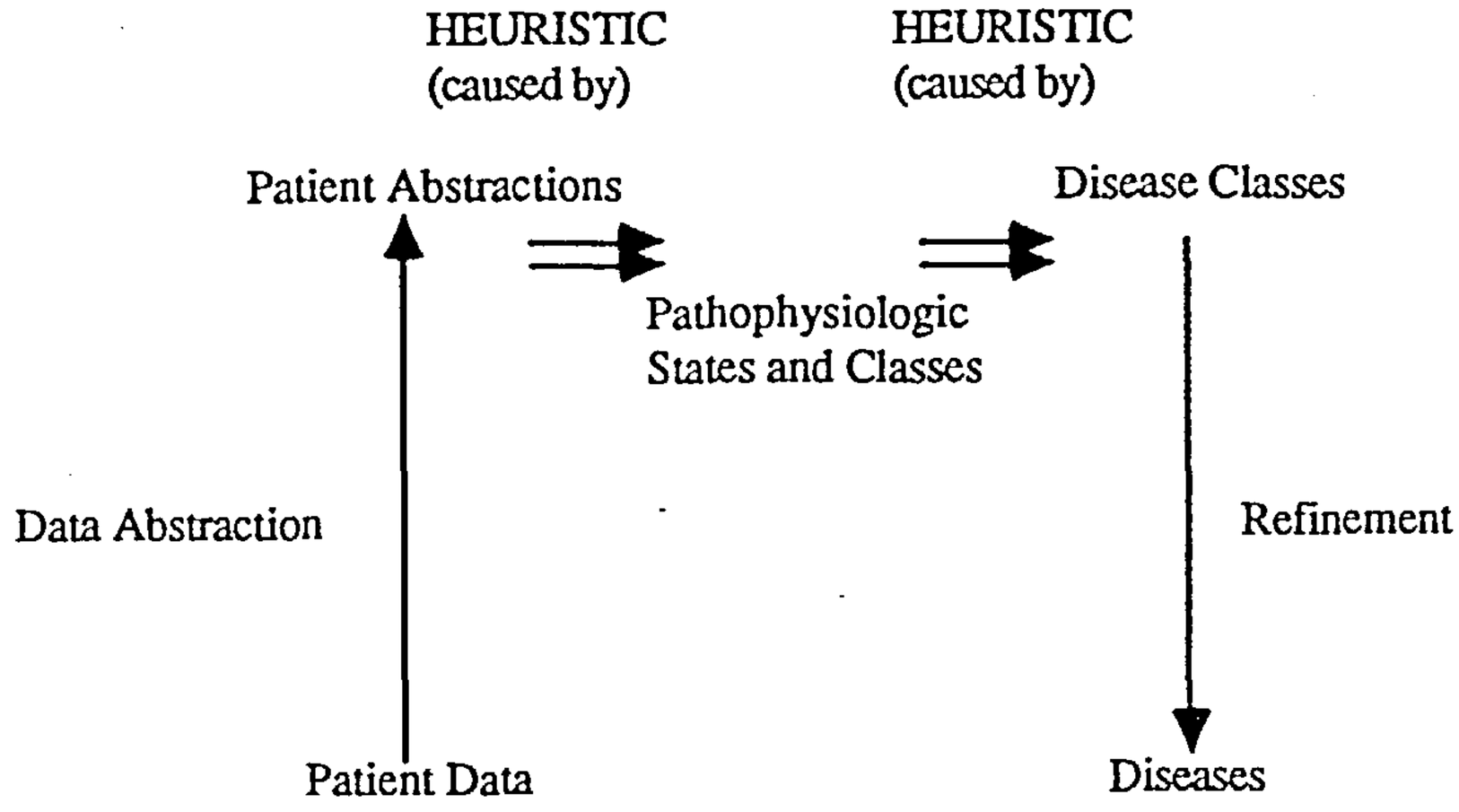

Figure 8-3: *Inference structure of causal-process classification*

structure compatible with that of 'findings' (patient abstractions). We shall, therefore, need to use the knowledge source 'abstract/transform' to get the raw (patient) data into the required format to be used by 'match'.

Lastly, having indexed into the disease classes, we shall need to *specify* the exact cause(s) underlying the symptoms. We should now be able to use the inference structure as a competence model for NEOMYCIN's diagnostic reasoning.

8.2.1.3 Task/Strategy Layers

The diagnostic procedure (Clancey, 1984-5) in NEOMYCIN is an embodiment of the task and flexible strategy structures as we know them in KADS. The underlying behaviour is one of collecting findings (forward reasoning) until some hypothesis (es) is generated. The reasoning then switches to hypotheses-driven mode, during which further findings will be requested of the user (patient). The requests for findings will either give confidence to the line of reasoning being followed (hypotheses being considered), or they may bring in a new reasoning focus.

The premise of (meta)rules in the diagnostic procedure can examine both the working memory, and the task history in order to decide what action to perform next. The (meta)rules, therefore, have got the possibility of a flexible-strategy written into them. In the sense that they can monitor the current situation, and generate new plans (eg., invoke new tasks), and have them executed by the task interpreter until a solution(s) is found. This, by the way, is what Clancey calls "meta-strategy".

Figure 8-4 represents a high level task structure for NEOMYCIN, a structure whose detailed behaviour is under the control of the flexible-strategy we have just described.

diagnose(disease)

abstract / transform(data)

obtain(data)

match(findings, diseases)

specify(disease)

Figure 8-4: *Task Structure for NEOMYCIN*

8.2.2. External Requirements

The system should ideally perform at three settings of (Clancey, 1984):

(1) *learning (knowledge acquisition),*

(2) *teaching (student modelling),*

(3) *and, problem solving (user modelling).*

The settings are, indeed, to be expected from any 'true' knowledge based expert system. For our present purposes the first two settings can be regarded as the external

requirements satisfying NEOMYCIN's external view. On the other hand, "user modelling" could in part be regarded as an aspect of functional requirements. The question of the exact position of "user modelling" with respect to internal and external views is currently a contentious one. It will have to wait until the issue of "modality" has been decided and put in its right perspective within KADS.

We shall describe briefly the extent to which NEOMYCIN attempts to satisfy the three settings, with a hint of the implications this will have for design. Where we talk about problem solving, all but its "user modelling" aspect belong in the internal view (cf. "the four layer Model"). We would, however, find it useful not to take out "problem solving" from this section, as this may create unnecessary gaps in our understanding of the settings which enjoy a close system support. The fact that Clancey finds these settings to be the essential ingredients of any 'true' knowledge based expert system can have a profound effect on where we may, in the future, draw the distinguishing line between the internal and external views. That is, if the settings are indeed the essential ones for a 'true' KBS, then they should all belong in the internal view. In this sense, among others, NEOMYCIN is inspiring and thought provoking when it comes to considering aspects of the *second generation* knowledge based systems.

Admittedly, most knowledge acquisition for NEOMYCIN takes place between people, with the knowledge base supporting the other two settings to a large extent. In teaching, the knowledge base is used by GUIDON2, a set of teaching programs. The teaching components themselves reside in GUIDON2 set of programs, exploiting NEOMYCIN as a suitable knowledge base for that purpose. Against this background the architecture of the system should satisfy three behavioural criteria of (Clancey, 1985):

(1) *problem solving,*

(2) *explaining own behaviour,*

(3) *modelling student behaviour*

The three behavioural criteria are indeed what the design of the system will attempt to satisfy, as we shall observe in the ensuing sections. The criteria required can be seen as a summary of the overall internal and external view requirements.

8.3. Design

8.3.1. Overview

The system uses the combined advantages of "rule based" and "frame based" approaches to achieve the required criteria (above). The former is used because of its simple syntax which the program can interpret for multiple purposes. Frame-based approach, on the other hand, enables the system to state the domain knowledge separately from the control knowledge, a most crucial notion providing for:

(1) non-redundant design - able to deal with large problem space,

(2) abstract reasoning - ability to explain own behaviour in terms of underlying reasoning structures divorced from 'data',

(3) multiple/clearly understood solutions - thus making the knowledge base suitable for teaching.

We shall describe next the two layers over which design will take place, given the output of analysis.

8.3.2. Functional Layer

The functional block diagram in figure 8-5 captures *what* is required of the system at

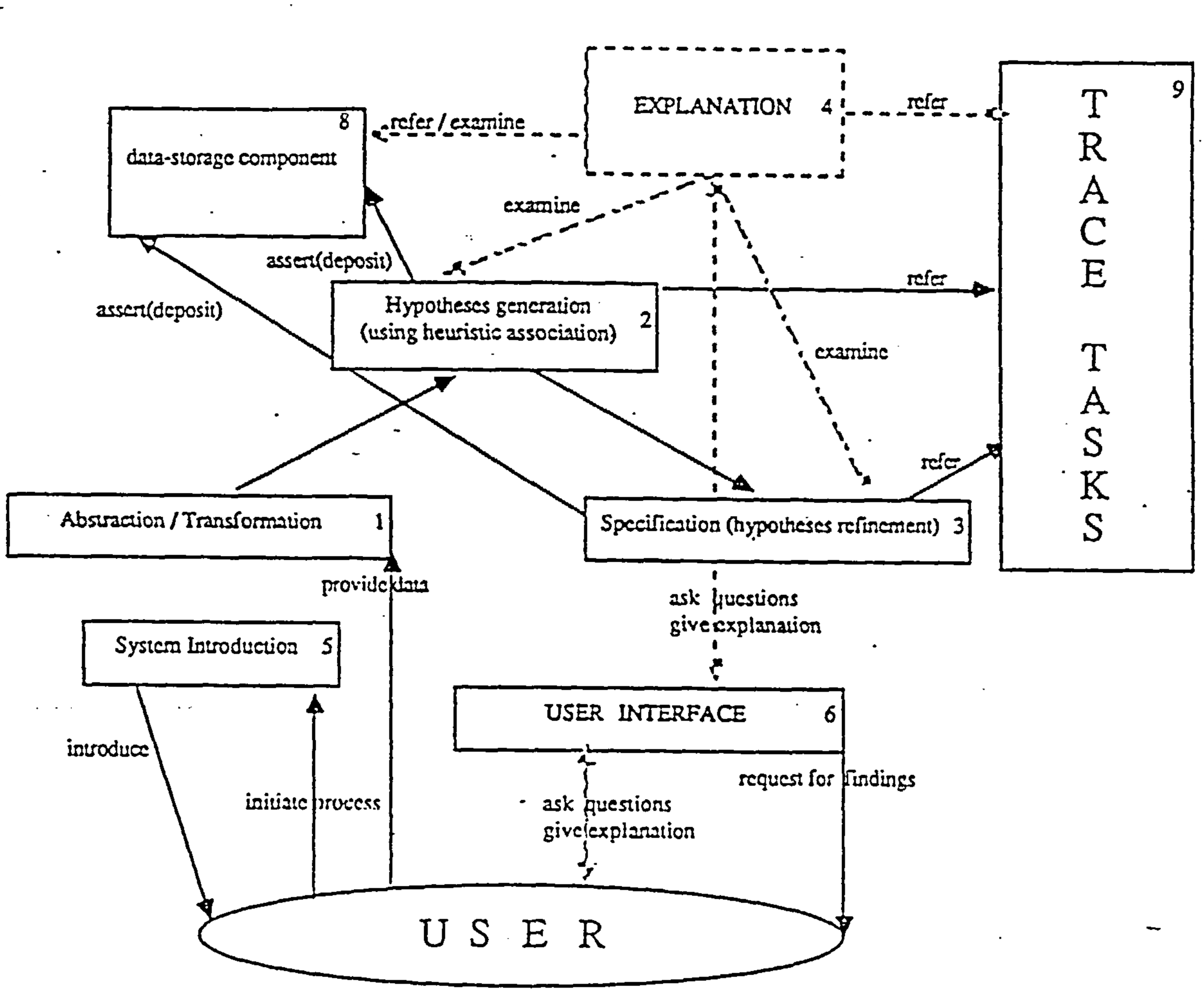

Figure 8-5: *NEOMYCIN's Functional Block Diagram*

an operational level. Blocks 1, 2, and 3 are concerned with the problem solving behaviour of the system. The blocks correspond closely with their counterparts in the inference/task structure (above).

Block 4 is an essential element of NEOMYCIN as far as the use of it by GUIDON2 for teaching, and general explanation is concerned. Note that it would be possible to

collapse block 4 onto 2 and 3. We have, however, decided to represent the explanation component explicitly, in order to emphasise its importance to the overall system.

Block 5 (introduction) is purely administrative, yet its realisation is required somewhere in almost all systems. This unit should provide the user with enough information to use (consult) the system.

Block 6 acts as a bi-directional filter to provide communication between users and the system. 'User I/F' will put out requests (for findings), or give explanation when required to do so. It will also take questions from the user and put them to the system. In both these cases 'User I/F' should ensure that messages or requests are clearly understood by the user or the system. This unit could ultimately engage in some form of 'natural language' parsing process; one day this would be an essential component of most knowledge based expert systems. The user interface activity is performed by the two tasks 'generate questions,' and 'ask-general-questions' as part of the diagnostic strategy (fig. 8-6).

Box 7 provides a means of keeping track of what tasks have been performed as part of the overall diagnostic strategy (see below). This functional block will be 'referred' to in order to explain reasoning, as well as supporting the decision to invoke the appropriate next task(s). In general, this is the 'task memory' registering all activities performed over time, as well as the order in which they take place.

Box 8 provides the means for depositing domain conclusions, providing an up to date picture of what has been established. This functional block is mainly responsible for maintaining the 'differential' (see below).

All arrows are qualified with their specific functionality, so as to make the relationship between all functional blocks clear. The input and output to each block are clearly shown in terms of incoming and outgoing functional blocks. We have specifically

avoided marking data (facts) as I/O to the blocks, as these become apparent at the physical layer.

The overall behaviour of the system is captured by the calling structure of the tasks in the diagnostic procedure (fig. 8-6). A structure in which the problem solver poses tasks for himself in order to have some structuring effect on the working memory. The procedure is intended to contain reasoning processes and structures circumscribing diagnostic behaviour in abstract. A requirement which will have a major influence on the detailed design of the system.

The tasks are called in a depth-first order, provided they are always relevant. The system starts with the top node task 'consult'. 'Consult' unconditionally invokes 'make-diagnosis' and then prints out the results of the consultation. 'Make diagnosis' invokes the two main branches of 'identify problem' and 'collect information'. The former deals with gathering information in order to establish the initial hypotheses - viz., mainly doing forward reasoning. The hypothesis driven reasoning is performed by 'collect information'.

The method used by the procedure is *heuristic classification* . It does diagnosis by selecting a *system identification* ('hypothesis' in case of NEOMYCIN) from a taxonomy pre-enumerated in the knowledge base. 'Thus, the program's architecture embodies a *general problem solving* method for constructing or interpreting systems by selecting from pre-enumerated solutions' (Clancey, 1985[a], p.4).

8.3.3. Physical Layer

8.3.3.1 Modular Structure

We identify three modules as part of the overall design architecture:

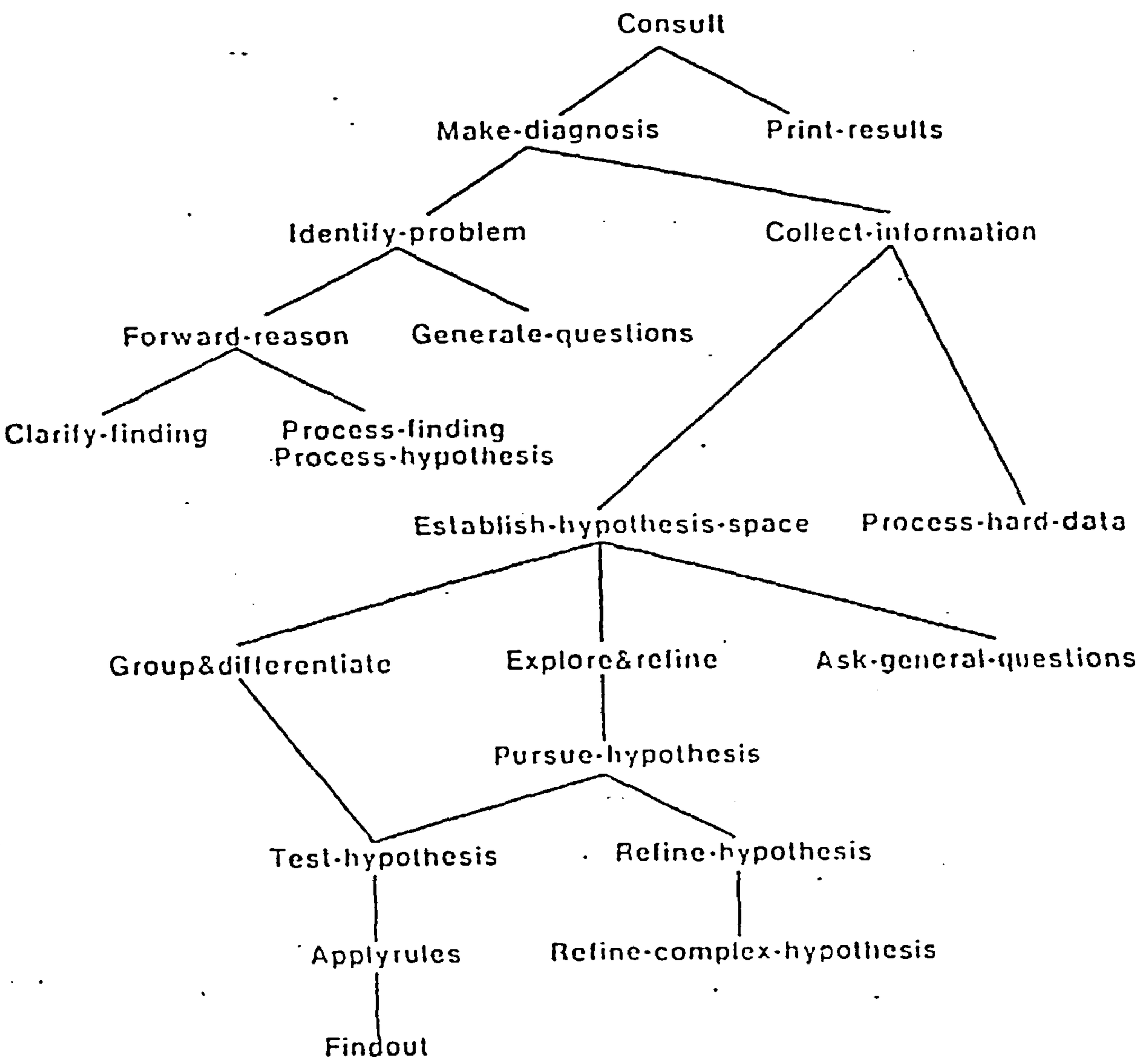

Figure 8-6: *NEOMYCIN's Diagnostic Strategy*

(1) Control knowledge

(2) Domain knowledge

(3) Working memory

'Control knowledge' separation from 'domain knowledge' should satisfy the require-

ment of representing diagnostic strategy in abstraction from 'data' on which it operates. As part of the 'control knowledge' module, a component is used to interpret the rule sets employed in tasks into actions prescribed by them (fig. 8-7).

8.3.3.2 Modules

CONTROL KNOWLEDGE

Purpose: To specify when and how the program should carry out operations such as:

- pursuing goals and focusing on hypotheses.

- acquiring findings and making inferences.

Description: This is the strategy component of the system represented as subprocedures we call tasks. Fig 8-6 depicts the internal structure of the module; all terminal tasks except 'review differential' ('print-results') invoke 'findout' directly or through 'applyrules'. Each task is a controlled sequence of conditional actions. Each conditional action reasons about domain rules (relations), thus called a *metarule* . A task has associated with it a description of how its metarules are to be applied. A task may also have an *end condition* , describing the condition under which it may be aborted by the task interpreter.

A task generally operates on a part of the working memory (hypothesis, finding, domain rule) called the task focus. A task may not have more than one focus, this could be a list which is the entire working memory. The tasks exploit the domain relations in order to provide explanation. This process takes the form of broadening the differential, contrasting hypotheses, focusing on a hypothesis, confirming a hypothesis, or determining whether a finding is present.

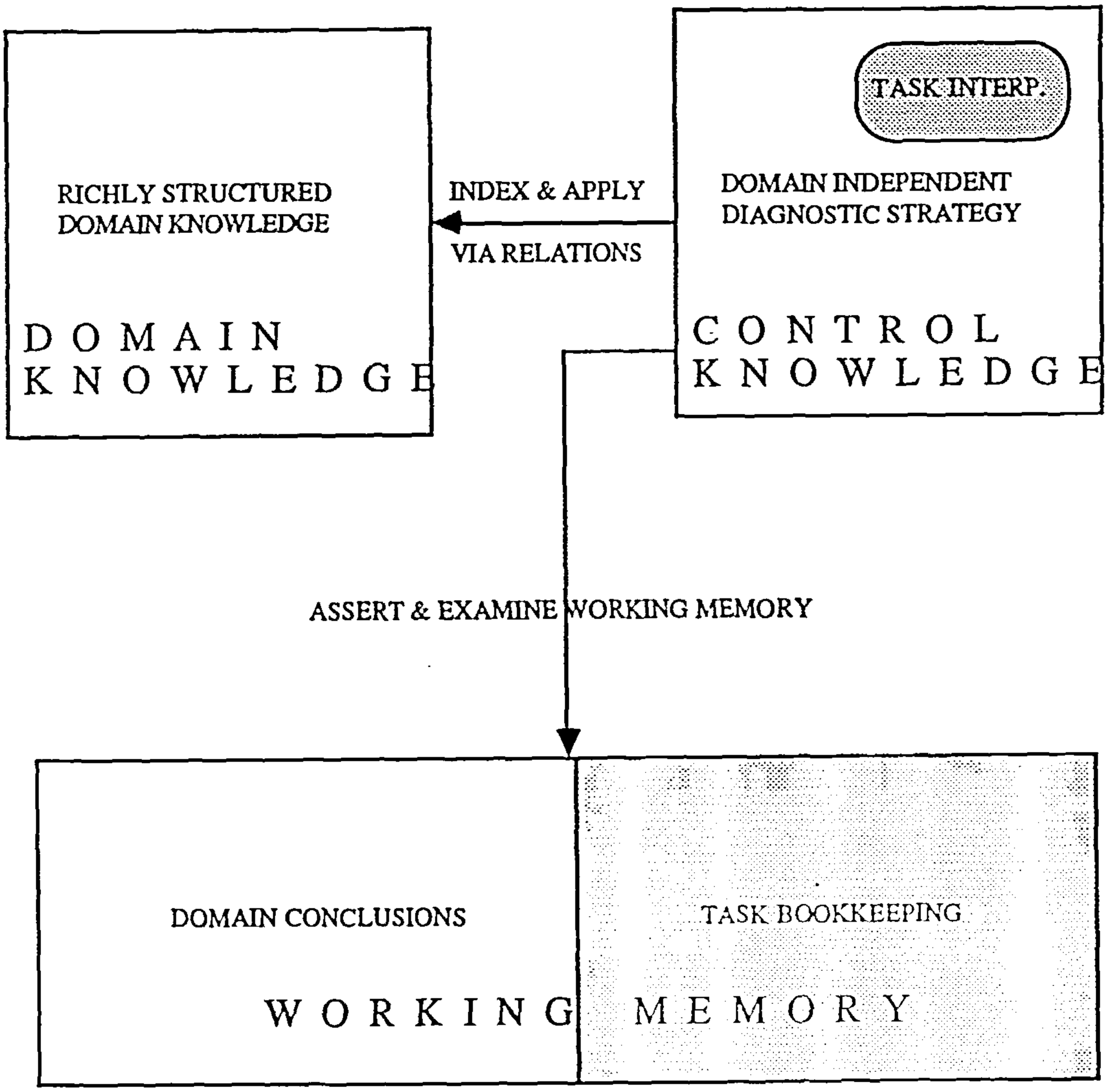

Figure 8-7: *NEOMYCIN's Modular Architecture Support*

Method

Heuristic classification : The diagnostic procedure ultimately maps onto a hierarchy of pre-enumerated solutions (disease processes), and refinement within this hierarchy; a process commonly known as classification. The classification has the added feature of relating nodes in different hierarchies by causal relations. These relations (domain rules) employ *certainty factors* describing the strength of relations between nodes

across hierarchies. A classification of this type is better known as *heuristic classification* (Clancey, 1985[b]). In applying the method the 'control knowledge' makes use of forward/hypothesis-driven reasoning. A strategy supported by *forward/backward* chaining mechanism inherent in the use of rules. The reasoning proceeds by prompting the user (patient) with general questions or requests for findings, as and when they are required.

Design elements

- *Task (subprocedure)*

  The overall element containing all other elements in the 'control knowledge'.

  A task, in turn, is translated into action by a function called the (task) interpreter.

- *Rules*

  Two general categories can be identified.

(1) *Metarule*

The rules making up the rule set associated with each task. The "if part" of the (meta)rule generally examines the working memory and domain knowledge. The "then part" invokes another task, applies a domain rule, or requests a finding of the informant.

(2) *Bookkeeping rule*

Performing operations such as resetting registers that characterise the state of the differential. These rules are placed in the task before and after the metarules.

- *Task type*

  A description of how the task should be applied. There are four possibilities:

(1) Simple, try-all ... Program type

(2) Simple, not try-all ... Conditional type

(3) Iterative, try-all ... For loop type

(4) Iterative, not try-all ... Pure production system type

- *Task focus*

  The part of the working memory on which a task is operating. The possibilities are: finding, hypothesis or domain rule. This element is part of the working memory, its inclusion here is to help us understand the make-up of tasks.

- *End condition*

  Evaluated every time a metarule succeeds. This is a mechanism for backing out of a procedure, when it becomes inappropriate or its goal is not of highest priority. 'DONOTABORT' is a special end condition, indicating that the associated task should be carried out to completion.

- *Primitive action*

  Actions issued by the "then part" of (meta)rules. The actions themselves are are carried out by the task interpreter (below). The meta-rules contain function-calls to them. The possible actions are:

(1) *Ask*

for finding (problem data).

(2) *Assert*

a domain fact.

(3) *Apply*

a domain (heuristic) rule.

Invoke a subprocedure (task).

- *Task interpreter*

This function uses a simple *deliberation-action* loop control method. In interpreting each task, the detailed application of the interpreter function is dictated by the task type. It is, therefore, justified to assume the four possible types described by task type as the detailed application *method* . The task interpreter can be seen as a sub-module of the control module. The prominent design elements of this sub-module are the primitive actions we have just described (fig. 8-8).

Composition principles *Tree* : This is the overall structure, on the nodes of which sit different tasks (fig. 8-6).

Access port *'Consult'* : The diagnostic procedure is always accessed (starts) through the top-node task 'consult'.

DOMAIN KNOWLEDGE

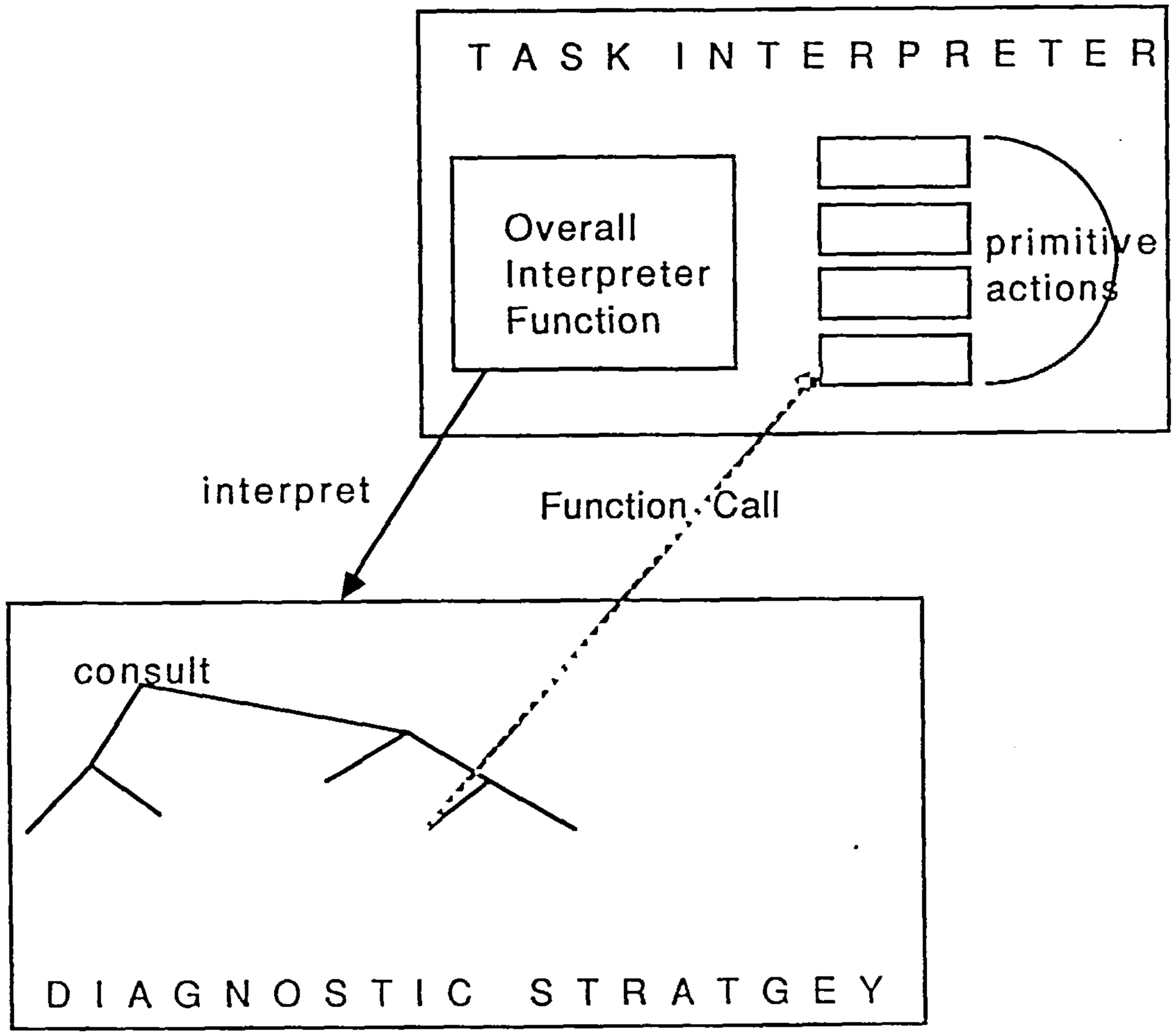

Figure 8-8: *NEOMYCIN's Control Module*

Purpose: To provide 'data' required by the diagnostic procedure in order to make inferences and give explanation. The data consist in medical concepts, structures and relations between concepts.

Description: The domain knowledge describes a medical vocabulary based on the physician's experience. The vocabulary consists in two broad categories of *hypotheses* and *findings* , and the relations between these. Findings are observations describing

the problem. In our case they consist in soft (circumstantial) and hard (lab) data on patients' symptoms. Hypotheses are partial descriptions of the findings. In this sense hypotheses explain the findings and constitute the problem-solver's diagnosis.

Hypotheses, in turn, divide into two groups of (ultimate)'causes', and 'general states'. The former relate to specific processes, whereas the latter are characterisation of abnormal conditions in the device (body).

Method

Heuristic classification : The method is applied to the domain knowledge by the diagnostic procedure (above). The classifier uses hypotheses refinement hierarchies, and findings subsumption hierarchies in conjunction with *heuristic* association. The associations are described in terms of *certainty factors* which describe in numerical terms the strength of causal relations between hypotheses and findings. The classifier makes use of forward/backward reasoning prompted by antecedent and trigger rules respectively.

Design Elements

States, unary and binary relations defined on states and other relations, and the strength of relations in terms of 'certainty factors' collectively exhaust the design elements.

- *States*

  These are similar to EMYCIN parameters. The distinction made between states (see below) are expressed in metarules. For instance, (SOFT-FINDING $STATE), (HYPOTHESIS $STATE) or (SOFT-FINDING), and (HYPOTHESIS) for short. Note that states are relations themselves, but we express them as atomic propositions (above) for convenience.

- *Hypotheses*

(1) *(A)etiological... processes*

(2) *State/category... substances*

- *Findings*

(1) *Soft... circumstantial or historical*

(2) *... laboratory or direct measurement*

- *Hierarchies*

These structures are defined in terms of relations amongst states.

(1) *Etiological taxonomy*

A refinement (subtype) hierarchy amongst etiological hypotheses (fig. 8-9). There are several of these present in the domain knowledge.

(2) *State/category subtype hierarchy*

A refinement hierarchy amongst 'general states'.

(3) *Subsumption hierarchy*

Findings are organised in such hierarchies. For instance, HEADACHE subsumes HEADACHE-severity and HEADACHE-duration.

- *Causal network*

A net of causal relations amongst state/categories.

- *(Other) relations*

It must be pointed out that all the elements we have described thus far are made up of relations. other domain relations are:

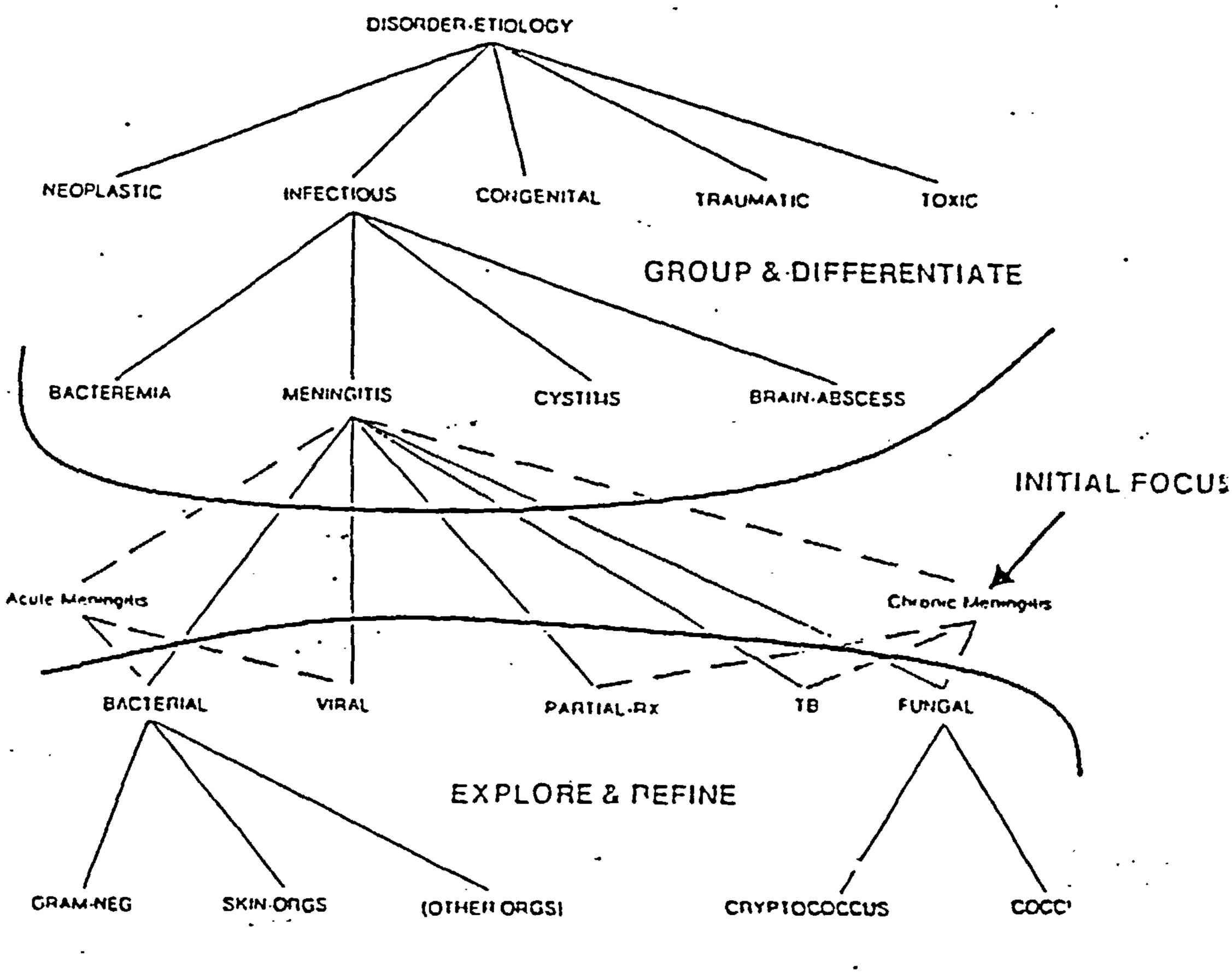

Figure 8-9: *An etiological hierarchy: containing causes and process features*

(1) *Source*

A finding can be the *source* of other findings, eg. (SOURCE-OF BLOOD-ANALYSIS WHITE-CELL-COUNT).

(2) *World-fact*

Findings can be related by *world-facts*, eg. males do not become pregnant. These relations are currently proceduralised in NEOMYCIN in the form of *'don't ask'* rules.

(3) *Definitional*

A finding can be defined in terms of other findings. Eg. "a neonate is a person

under five months of age."

- *Process feature*

  A findings/hypothesis can characterise in more detail the process described by another finding or hypothesis. A pain can be characterised by location and change in severity over time. Every hypothesis in the etiological taxonomy can be characterised by a set of similar process features.

- *Certainty factor*

  Associated with each causal relation is a *certainty factor* (CF), as used in MYCIN. CF is used for *heuristic association* between hypotheses and findings, thus omitting causal details.

- *Rules*

(1) *Antecedent*

(cf fig. 8-10) A causal relation which is definitive, i.e. CF = 1.0. This rule is used for forward-reasoning when the premise of the rule is known to be true.

(2) *Trigger*

An antecedent rule which is also labelled as *trigger rule*. In receiving a trigger rule the program will try to satisfy the premise of the rule. The trigger rule is, therefore, used for reasoning by backward chaining. **Composition Principles**

Network of hierarchies : A network at the nodes of which sit hypotheses refinement, and findings subsumption hierarchies. The network, itself, is described in terms of causal relations between hypotheses, and between hypotheses and findings. That is hypotheses causing other hypotheses, or being caused by them, and hypotheses causing findings. The network is, therefore, a causal network.

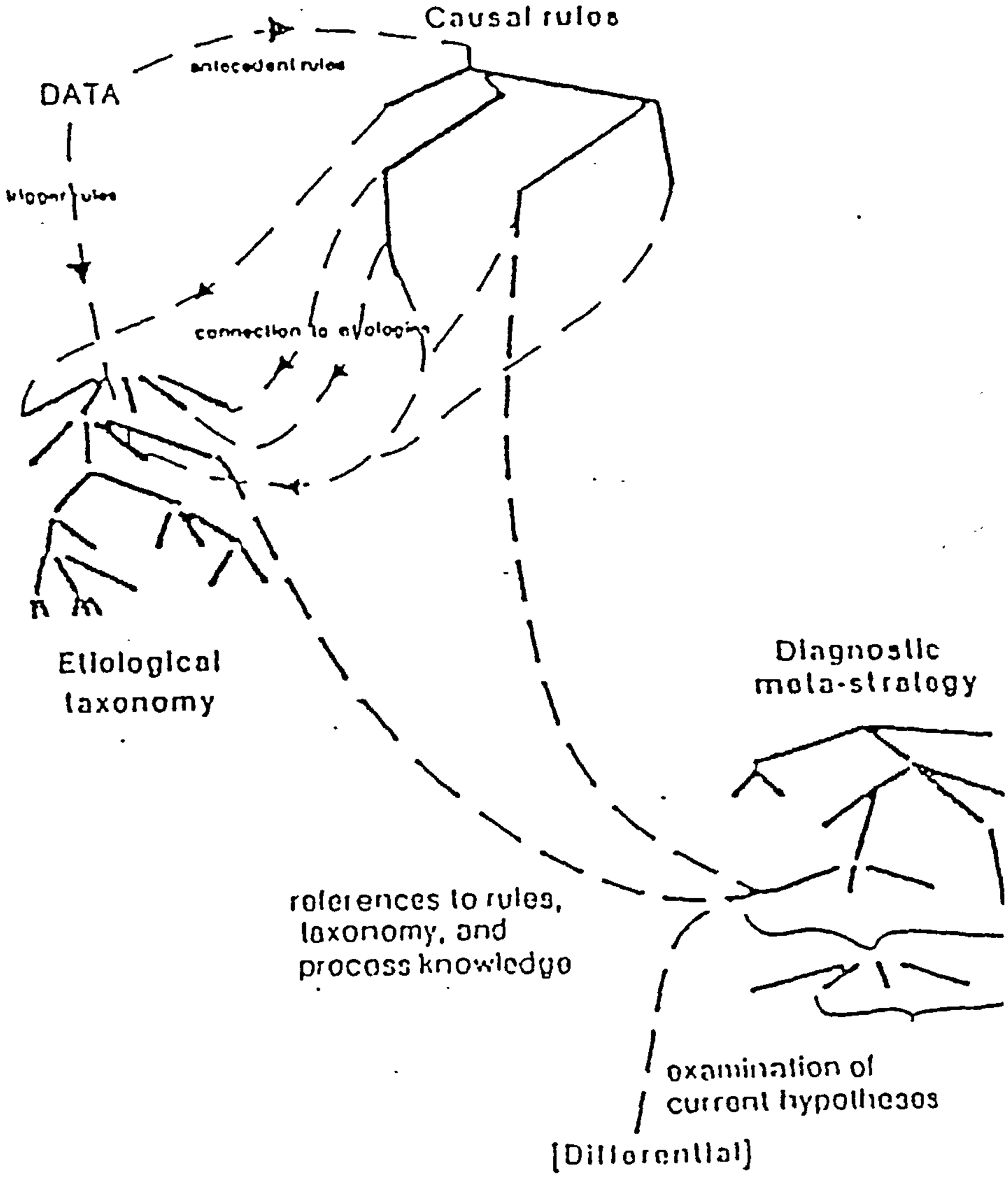

Figure 8-10: *Components of the NEOMYCIN system*

Access port

Procedural attachment : Procedural attachments are associated with metarules, and are accessed by their premise in applying the control knowledge (program) to that of domain knowledge (data). It may be interesting to note that procedural attachments replace arbitrary functions which were used before to provide access to domain relations.

WORKING MEMORY

Purpose: To provide a medium for organising the possibilities to consider (the differential), as well as for recording the trace of activities taken place over time (history). The differential ultimately should isolate the cause(s) of a complaint (or system fault).

Description: The working memory is extensively used by the "then part" of metarules in order for them to store "domain conclusions" (fig. 8-7). The "if part" of the metarule, on the other hand, examines the working memory and operates on it through the task focus. The focus, itself, can be a hypothesis, finding, or domain rule.

The continuous assertion and examination of domain conclusions, by the tasks, results in broadening and narrowing of the differential. A focusing activity which decides the ordering of invocation of subsequent tasks (subprocedure) by the control knowledge until a diagnosis is achieved.

The other part of the working memory is purely concerned with recording the history of all of the tasks which have been performed (fig. 8-7). This part is consulted by metarules, for example to determine if a particular hypothesis has been pursued.

Method

List manipulation : The method used here is a non-AI one. The working memory is a list which is being continuously updated. The updating takes the form of broadening and pruning the list such that it reflects the current diagnostic thinking, and the history of the tasks performed.

The whole operation is conducted through *randomly accessing* the list using the task focus, a process in which pointers to foci are inserted or deleted as required.

Design elements

Except for the overall list structure, all of the other elements are place holders for the type of elements already declared as part of the other two modules (above). This is so because the working memory uses instantiations of the elements already contained in the other two modules.

- *Task focus*
- *Hypotheses (making up the differential)*
- *Findings*
- *Domain rules*
- *Hierarchies*

  those observed amongst hypotheses/findings (cf 'domain knowledge')

- *History*

  A hierarchy of the names of the tasks which have already been performed . The hierarchy may also contain the task foci.

- *List*

  The overall structure of the working memory.

Composition principles

List of hierarchies : The overall list structure of the working memory has hypotheses/findings hierarchies hanging from its cells. This is because in considering a finding/hypothesis we may need to examine its ancestor and/or its children as well.

Access port

Task focus : The control module accesses (links up with) the working memory using the *task focus* .

9. Summary, Conclusion, and Future Plans

9.1. Summary and Conclusion

KADS methodology will provide a knowledge engineer with modelling tools to:

- Elicit knowledge from experts, and
- to 'formalise' that knowledge into a detailed design to be implemented on a computer system.

In both of these cases "interpretational models" play the central role, and should, therefore, be seen as KADS major contribution to the field of KBS. A library of such models has been developed within KADS, which will cover a population of KBS at an analysis level.

The development of KADS is equally concerned with populating the design space with models of equal usefulness to interpretation models. This will be discussed in a later section in this chapter.

At the moment, a descriptive language has been developed at the design level, in which a body of AI and conventional design methods and techniques have been incorporated. This language will require further development, so as to formalise the design activity more extensively. The formalisation should stop short of interfering with a designer's sense of creativity. Otherwise, KADS design language will become as unattractive to use as many conventional design methods have in the past.

KADS' other major attraction which is new in the field of KBS, is the capture of business needs side by side with the problem solving components at the analysis level.

This should make KADS a more commercially viable proposition, when it comes to developing systems within real life financial, management, and user constraints.

The use of KADS within different domains of expertise (fourteen in our case) has encountered successes and setbacks, with the former outweighing the latter. The prime areas in which KADS has been successful are:

(1) The use of IM in modelling a domain experise, thus eliciting knowledge (see Breuker, et al. 1983 [a,b,c], 1984; also Wielinga, et al. 1984) from an expert in a "model driven" fashion.

(2) The use of an interpretation model or a number of such models to create a workable conceptual model.

(3) The sharing of information across European countries through the use of expert knowledge contained in "conceptual models", statements of "business needs", and system design architecture.

(4) Following from (3), the ability to develop models for large systems in collaboration with partners at remote sites.

(5) The use of KADS to extend KADS itself. i.e. new IMs have been created by users of KADS, once they had learned how to develop one.

(6) The creation of design language usefully enhanced the power of KADS analysis, resulting in a number of working systems.

The important setbacks are as follows:

(1) Modality statements cannot, as yet, be captured sufficiently within conceptual models. It is intended that separate models should be developed for these statements. These models are thought to be orthogonal to conceptual models, since

modality statements cut across the functionality of a working system. This is because these statements will provide the communication links between the problem solving component of a system and the external world. The work on 'modality models' is at a seminal stage.

(2) The fourth layer of a conceptual model known as "strategic layer" has proved difficult to capture. This layer contains elements of dynamic planning of new task structures, as well as monitoring their behaviour. We are not very discouraged by this, since the problem is shared in general within the AI community. We bypass this layer by having a sufficient set of task structures, amongst which we can choose one best suited to a problem solving situation. Some domains tend to be satisfied by just having a sufficiently general task structure, without the need for the strategic layer.

(3) In smaller domains, the use of KADS will involve too much overhead. Experience shows that certain elements of KADS can still be used in such domains.

9.2. Future Plans

We shall divide this section into two parts:

(1) The role of prototyping in design,

(2) Modelling

(1) is concerned with the use of prototyping as a support activity in design. It is included here for completeness; the issue of prototyping within KADS will need to be pursued as an important future task. We, however, find it apt to indicate our position in this regard. We should hope that our discussion on prototyping would also prove beneficial to the future work concerned with this issue.

(2) will be addressing the issue of design models, as an extension to interpretation models. We hope that we can provide generic design models for the future users of KADS, so that these models may be used in conjunction with their corresponding generic epistemological counterparts (interpretation models) in analysis.

9.2.1. Prototyping in Design

Prototyping can be used within the two layers of KADS design methodology, in order to test the feasibility of the components of each layer. We have assumed that Functional blocks, and physical modules are the building blocks of the two respective layers of our design. We shall, therefore, suggest a set of possibilities for prototyping, based on these building blocks. The list is not an exhaustive one; it may be extended, as well as modified, in the course of future developments in KADS.

9.2.1.1 Prototyping in the Functional Layer

This should be done to establish the following:

(1) The correspondence between blocks and analysis output: each functional block should be treated as a separate entity, with I/O to/from each block assumed. The assumed I/O entries and exits need not, necessarily, be a very close reflection of the actual ones, since we are concerned primarily with the internal composition of the block under consideration. The block is, then, prototyped to ensure that: it functions as it should, relates to the analysis output as originally planned, and the method used in realising the block can be shown to correspond (isomorphically, semi-isomorphically, or non-isomorphically) with the analysis elements (goal statements, knowledge sources, and so on).

(2) Block Organisation: As a result of prototyping, we may decide to organise our blocks differently to that originally assumed. This may be due to the fact that

what seemed a good idea in the first instance, might prove to be difficult to implement. This is a classic example of trading off between 'the ideal,' and 'the possible'.

(3) Classificatory Prototyping: Where we can make distinctions between certain classes of blocks, we may choose to develop prototypes for each class separately. Possible examples might be "data storage," "natural language parser," "ATN generator," and "Classification" blocks. We can test prototypes for each distinct class to ensure validity and appropriateness. We may then decide to amass a library of different classes of prototypes for corresponding functional blocks for future use, be it with some modifications.

(4) Methodological Prototyping: It is desirable to establish a link between the dynamic parts of the analysis and the set of available methods. For instance, we may choose to examine the connection between knowledge sources and their counterparts in methods. It is possible to experiment with a knowledge source (or a set of them) across different domains and find out about their design counterparts. We may then use this experience in developing systems using that knowledge source in other systems. We have an example of such a development for the knowledge source 'heuristic match' in the domain of ECGD (cf. Davoodi, 1987[a], see also appendix A). The idea can be extended to meta-classes, composite goal statements, and various other elements of the analysis output.

(5) User Reaction: By developing different prototypes for blocks addressing the internal and external views, users' reaction can be assessed. Based on this reaction, we can tune, or fine tune our design, as well as extend it. This is an important use of prototyping, since we can also use it as a means of rectifying the possible anomalies in the analysis output, against the user, expert, or

management requirements.

(6) Composite Prototyping: Having tested the functional blocks individually, we, may, then prototype the overall functional block hierarchy, and the corresponding methods. This should provide a handle on system coordination, as well as establishing the exact nature of I/O and other links between the blocks. This process may lead to further tuning of the blocks, resulting in a fully working hierarchy. This is always the last stage of prototyping in the functional layer before descending to the physical layer.

We shall, no doubt, be able to think of other cases for prototyping in this layer. We feel, however, that the list given provides a good insight into the role of prototyping and its usefulness within the functional layer.

9.2.1.2 Prototyping in Physical Layer

We shall mention briefly how prototyping may be used at this layer. Most of the uses of prototyping we enumerated for the functional layer can be applied to this layer as well, replacing modules for blocks.

Prototyping may be used to give us a clearer idea of how design elements may be distributed across modules. It can also be used to achieve an optimal modular configuration, also providing a clear idea of inter-modular (composition principle) and intra-modular (hierarchy / network of modules) configuration. Finally, we can use prototypes as handles for knowledge representation both globally (such as shells), and locally (such as frames, and rules). The latter is guided both by the available development kit, and by the available resources (machines and programmers).

We encourage the use of high culture (for a description of high and low culture shells, see Warden, 1987) shells and environments for prototyping, as well as for full system

development. Low culture shells (cf. ibid.) may be used for prototyping only when the developer is *certain* that it is the correct support tool to use for *intermediate* system assessment.

We shall, next, give some brief indications of our future plans in extending the design methodology in incorporating the modelling activity.

9.2.2. Generic Design Models

KBS development is ultimately a process of modelling. The modelling activity, itself, takes place at two global levels of analysis and design. We can, of course, consider tow other types of modelling , namely: 'psychological,' and 'administrative'. The former, roughly, relates to our conception of the expert and his environment in the process of knowledge elicitation. That is, we all the time tend to have a model of interviewee as we are proceeding with the elicitation of knowledge. This model is constantly being reshaped and modified in our minds' eyes. The administrative modelling activity is what takes place at the lifecycle model level. That is we purport to organise activities and phases of the system analysis and design within interconnected sequences, some of which may be overlapping.

Our talk of modelling here is concerned with that of analysis, and in particular with design. In considering the interesting expert domains in the 'real world,' we shall arrive at a relatively limited set of classifications. That is, irrespective of the fact that on the surface of it there are many expert domains to be conquered within the machine, we claim it possible for those domains to fall under a limited set of distinct classes of activities (see Chandrasekaran, 1987; Buchanan, 1987).

We should be able to cater for a future in which the knowledge engineer should be able to pick out of his 'toolkit' the appropriate "task-oriented" shell satisfying his

requirements. This is exactly what happens in the use of EMYCIN to help the development of NEOMYCIN. No doubt future systems will be built based on HERACLES which is the shell derived from the work on NEOMYCIN. Indeed, Clancey and his students are applying the shell in an engineering domain (cf. Clancey, 1985[b]) at the time of writing of this thesis.

The number of generic models such as HERACLES are very few, and they are mainly contained to a limited number of KBS / AI research centres. If we are to have any hope of turning KBS development from an artform into an *engineering process*, we need to make these generic models more accessible by making their history of development explicit.

This is the ultimate aim of KADS, in that we should provide a documented trace of the development for each of the future generic models. The trace should include the interpretation model(s), the possibility of different 'realistic' external requirements, and a *generic* design model. The idea of a generic design model is similar to that of interpretation models, in so far as such models can be applied to domains sharing an interesting expert feature. The major difference between the generic design and interpretation models, primarily, is twofold:

(1) A generic design model needs to indicate the incorporation of some possible envisaged external requirements; an interpretation model does not.

(2) Since design models are directly aimed at supporting real life tasks they are likely to embed within them a number of primitive interpretation models. The alternative would be that of having a design model which is based on a 'real life task' interpretation model, or a combination of both.

Our aim in design should be to provide such generic models, to be used in conjunction with the corresponding interpretation model(s) and external requirements. This should

make it possible for users of KADS to develop systems independently, as well as together, without having to have access to KADS 'gurus' for advice and consultation. The generic design models should be as much capable of being modified, or enhanced, as interpretation models are aimed to be.

9.3. KADS in Future Systems

The hallmark of second generation KBS is its ability to represent domain and reasoning knowledge separately (see, for instance, Steels, 1987). Such a separation will make it possible to have a representation of domain knowledge in its 'full' glory, thus avoiding problems associated with shallow knowledge representation. A *deep* representation of knowledge will make it possible to understand the working of a system better, and in turn to be able to extend and maintain it. It should also make it possible to use the system for both *teaching* and *learning* some expertise aspects

KADS has been developed with the need for *deep* knowledge representation in mind. This can be clearly seen in the way an expertise domain is represented within the KADS four layer model, by separating out the *domain layer* from the reasoning (control) part of the expertise.

KADS will also make it possible to bypass the expensive and generally unworkable paradigm of "incremental prototyping" by having a clear model of the expertise prior to any design and implementation. On the other hand, KADS will, in the future, make good use of prototyping as a support activity (see above).

KADS can equally be used in conjunction with a conventional methodology, to make practicable a modelling of combined 'heuristic reasoning' and 'database management' within one environment. The environment is supported by the KADS *lifecycle model*, and its modelling toolkit. Developing integrated systems which combine the power of

conventional database management and KBS technologies is a very current and challenging issue. We anticipate that this useful trend will carry on for the next decade and beyond, until such systems will become common practice. It is, also, possible to extend KADS to embed a conventional methodology within it, so as to support the development of integrated systems 'seamlessly'. We find that KADS' contribution to the future development of KBS and integrated systems is a major one. A contribution which manifests itself by taking out the development of such systems from the realm of an *artform* into that of an *engineering process*.

10. References

Abbret, G., and Burstein, M., *The KREME Knowledge Editing Environment*, Proc. of the Knowledge Acquisition for KBS Workshop, Banff, Canada, 1986.

Allen, J., and Anjewierden, A., *KADS Power Tools: User Interface Specifications*, ESPRIT P1098, Working Paper , 1987.

Alexander, J. H., Shulman, M. J., Rehfuss, S. J., and Messick, S. L., *Ontological Analysis: An Ongoing experiment*, Proc. of the Knowledge Acquisition for KBS Workshop, Banff, Canada, 1986.

Anjewierden, A., *PED - KADS Power Tools Protocol Editor*, ESPRIT P1098, Working Paper, April 24, 1987.

Anjewierden, A., and Allen, J., *KADS Power Tools: User Guide*, ESPRIT P1098,Working Paper, 1987.

Barthelemy, S., Edin, G., Toutain, T. (Cap Sogeti Innovation S. A.), and Becker, S. (SCS Organisationsberatung und Informationstechnik Gmbh), *Requirements Analysis in KBS Development*, ESPRIT P1098, Deliverable D3, 1987.

Bennett, J. S., *ROGET: A Knowledge-based consultant for acquiring the conceptual structure of of an Expert system*, Report no. HPP-83-24, Computer Science Department, Stanford University, 1983.

Bobrow, D. G.(ed.), *Qualitative Reasoning about Physical Systems*, Elseviers Science Publishers B.V., Amsterdam, 1984.

Booch, G., *Object Oriented Development*, IEEE Transactions on Software Engineering, vol. SE-12,2, 1986.

Brachman, R. J., and Schmolze, J. G., *An overview of the KL-ONE Knowledge Representation System*, in Cognitive Science, vol. 9, pp.171-216, 1985.

Brachman, R. J., *On the Epistemological Status of Semantic Networks*, in Associative Networks - Representation and Use of Knowledge by Computers, edited by N. V. Findler, Academic Press, New York, 1979.

Breuker, J., Wielinga, B., *Initial Analysis for Knowledge Based Systems - An Example*, **Report 1.3a**, ESPRIT Project 12, Memorandum 23 of the research Project: **The Acquisition of Expertise**, University of Amsterdam, Dept. of Social Science Informatics and Laboratory for Experimental Psychology, Weesperplein 8, 1018 XA Amsterdam, June 1983[a].

Breuker, J., Wielinga, B., *Analysis for Knowledge Based Systems - Part 1*, **Report 1.1**, ESPRIT Project 12, Memorandum 10 of the research Project: **The Acquisition of Expertise**, University of Amsterdam, Dept. of Social Science Informatics and Laboratory for Experimental Psychology, Weesperplein 8, 1018 XA Amsterdam, October 1983[b].

Breuker, J., Wielinga, B., *Analysis for Knowledge Based Systems - Part 2*, **Report 1.2**, ESPRIT Project 12, Memorandum 12 of the research Project: **The Acquisition of Expertise**, University of Amsterdam, Dept. of Social Science Informatics and Laboratory for Experimental Psychology, Weesperplein 8, 1018 XA Amsterdam, December 1983[c].

Breuker, J., Wielinga, B., *Techniques for Knowledge Elicitation and Analysis* **Report 1.5**, ESPRIT Project 12, Memorandum 28 of the research Project: **The Acquisition of Expertise**, University of Amsterdam, Dept. of Social Science Informatics and Laboratory for Experimental Psychology, Weesperplein 8, 1018 XA Amsterdam, July 1984.

Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billeaut, J-P (University van Amsterdam), and **Davoodi, M.**,Hayward, S. (STC), *Model Driven Knowledge Acquisition - Interpretation Models*, ESPRIT P1098, **Deliverable D1**, 1987.

Brodie, M. L., Mylopolous, J. (eds.), *On Knowledge Based Management Systems*, (Pub.) Springer Verlag, 1986.

Brown, J. S., Burton, R. R., and de Kleer, J., *Pedagogical, natural language, and Knowledge engineering techniques in SOPHIE I, II, and III*, in **Intelligent Tutoring Systems**, (eds.) D. Sleeman and J. S. Brown, pp. 227-282, London, Academic Press, 1982.

Buchanan, B. G., *Expert Systems, in* **Expert Systems Tutorial Notes**, *IJCAI-87, Milan, Italy, 23 August, 1987.*

Buchanan, B. G., and Feigenbaum, E., A., *DENDRAL, and Meta-DENDRAL: Their applications dimension*, **Artificial Intelligence**, 11:5-24, 1978.

Buchanan, B. G., and Mitchell, T. M., *Model directed learning of production rules*, **Report. STAN-CS-77-597**, Computer Science Department, Stanford University, Calif.,1977; also in **Pattern-Directed Inference Systems**, (eds.) D. Waterman, and F.

Hayes-Roth, pp. 297-312, New York: Academic Press, 1978.

Buchanan, B. G., and Shortliffe, E. H., *RULE-BASED Expert SYSTEMS: The* MYCIN *Experiments of the Stanford Heuristic Programming Project*, , **Addison-Wesley Publishing Company, 1984.**

Bundy, A.(ed.), *A Catalogue of AI techniques*, (pub.) Springer-Verlag, New York, 1985.

Bundy, A., Byrd L., Lager G., Mellish C., and Palmer M., *Solving mechanics problems using meta-level inference*, in **IJCAI 6**, pp. 1071-1027, 1979.

Bundy, A., and Sterling, L., *Metalevel inference in algebra*, Department of Artificial Intelligence, University of Edinburgh, 1981.

Bylander, T., and Chandrasekaran, B., *Generic Tasks in Knowledge-Based Reasoning: The 'Right' Level of Abstraction for Knowledge Acquisition*, **Proc. of the Knowledge Acquisition for KBS Workshop**, Banff, Canada, 1986.

Bylander, T., and Mittal, S., *CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling*, **AI Magazine**, vol.7/2, pp. 66-67, 1986.

Clancey, W. J., *The epistemology of a Rule-Based Expert System: a Framework for Explanation*, **Artificial Intelligence**, vol. 20, no. 3, pp. 215-251, 1983.

Chandrasekaran, B., *Towards a Functional Architecture for Intelligence based on Generic Information Processing Tasks*, **Invited talk**, IJCAI, Milan, Italy, 28 August, 1987.

Charniak, E., and McDermott, D., *Introduction to Artificial Intelligence*, Addison-Wesley Publishing Company, 1985.

Clancey, W. J., *Acquiring, Representing, and Evaluating a Competence Model of Diagnostic Strategy*, **Report no. KSL-84-2**, Original draft, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Palo Alto, February 1984.

Clancey, W. J., *Representing Control Knowledge as Abstract Tasks and Metarules*, **Working Paper no. KSL 85-16**, Stanford Knowledge Systems Laboratory, Computer Science Department, Stanford University, Palo Alto, April 1985[a].

Clancey, W. J., *Acquiring, Representing, and Evaluating a Competence Model of Diagnostic Strategy*, Stanford Knowledge Systems Laboratory, Computer Science

Department, Stanford University, Palo Alto, 9 August, 1985[b]. (Also appearing in Contributions to the Nature of Expertise, Chi, Glaser, and Far (eds.), 1985).

Clancey, W. J., *Heuristic Classification*, in **Artificial Intelligence**, vol. 27, pp. 289-350, 1985[c].

Clancey, W. J., and Letsinger R., *NEOMYCIN: Reconfiguring a Rule-Based Expert System for Application to Teaching*, **proc. 7th IJCAI**, pp. 829-835, Vancouver, B. C., Canada, 1981.

Davis, R., and King, J., *An Overview of Production Systems*, in **Machine Intelligence**, pub: John Willey & Sons, New York, 1977[a].

Davis, R., and Buchanan, B. G., and Shortliffe, E., *Production Rules as a Representation for a Knowledge Based Consultation Program*, **Artificial Intelligence**, Vol. 8, no. 1, Feb. 1977[b].

Davis, R., *Interactive Transfer of Expertise: Acquisition of New Inference Rules*, **Artificial Intelligence**, vol. 12, no. 2, pp. 121-157, 1979.

Davis, R., *TEIRESIAS: Applications of Meta-level Knowledge*, in **Knowledge-based Systems in Artificial Intelligence**, (eds.) R. Davis and D.B.B. Lenat, McGraw-Hill, 1982

Davis, M., *The Mathematics of Non-Monotonic reasoning*, **Artificial Intelligence**, vol. 12, nos. 1&2, 1980.

Davoodi, M., *An Interpretation Model for the Generic Task of DESIGN of HARDWARE CONFIGURATIONS(DHC)*, ESPRIT P1098, **P1098 internal publication**, STC Technology Ltd., November, 1986[a].

Davoodi, M., *Real Life Application of an Interpretation Model*, ESPRIT P1098, **P1098 internal publication**, STC Technology Ltd., November, 1986[b].

Davoodi, M., *K1 - An Implementation of "Compare" KS in the context of ECGD domain*, the Consolidation Stream, , ESPRIT P1098, **P1098 internal publication**, STC Technology Ltd., 4 April, 1987[a].

Davoodi, M., *A Window to Design in KADS*, ESPRIT P1098, **P1098 internal publications**, STC Technology Ltd., EUROKOM text no. 185584, 13 July, 1987[b].

Davoodi, M., *NEOMYCIN: a KADS Perspective*, ESPRIT P1098, **P1098 internal publications**, STC Technology Ltd., September, 1987[c].

Davoodi, M., *KADS - a Methodology for KBS*, proc. first conf. for KBS in Government, Central Computer and Telecommunications Agency(CCTA), pp. 19-36, November 1987[d].

Davoodi, M. (STC), Bredeweg, B., Schreiber, G., and Wielinga, B. (UvA), *A Design Methodology for KBS - KADS*, ESPRIT P1098, Deliverable D8, November 1987.

Diederick, J., Ruhman, I., and May, M. *KRITON: A Knowledge Acquisition Tool for Expert Systems*, Proc. of the Knowledge Acquisition for KBS Workshop, Banff, Canada, Banff, Canada, 1986.

Diederick, J., *Knowledge-Based Knowledge Elicitation*, in Proc. of 10th IJCAI conf., pp. 201-204, Milan, Italy, 1987.

Eshelman, L., Ehret, D., McDermott, J., and Tan, M., *MOLE: A Tenacious Knowledge Acquisition Tool*, Proc. of the Knowledge Acquisition for KBS Workshop, Banff, Canada, Banff, Canada, 1986.

Feigenbaum, E. A., *Knowledge Engineering: the Applied Sides of Artificial Intelligence*, formerly Report no. HPP-80-21, Stanford Heuristic Programming Project, Stanford, CA., 1983.

Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J., *On generality and problem solving: a case study using the DENDRAL program*, in Machine Intelligence, (eds.) B. Meltzer and D. Michie, vol. 6, Edinburgh: Edinburgh University Press, pp. 165-190, 1971.

Frege, G., *The foundations of Arithmetic*, Trans: Austin, J. L., Oxford, Blackwell, 1953.

Frege, G., *Philosophical Writings*, Eds: Geach, P. and Black, M., 2nd edn., Oxford, Blackwell, 1960.

Frege, G., *Conceptual Notation and Related Articles*, trans. and ed: Tyrell Ward Bynum, Oxford, Clarendon Press, 1972.

Gains, B. R., and Shaw, M. L. G., *Induction of Inference Rules for Expert Systems*, in Journal of Fuzzy Set Systems, vol. 18, no. 3, University of Calgary, Dept. of Computer Science, Alberta, Canada, 1986.

Gale, W. A., *Knowledge Based Knowledge Acquisition for a Statistical Consulting System*, Proc. of the Knowledge Acquisition for KBS Workshop, Banff, Canada, Banff, Canada, 1986.

Gallaire, H., and Lasserre, C., *Metalevel Control for Logic Programming*, in **Logic Programming**, London, Academic Press, 1987.

Ginsberg, A., Weiss, S., and Politakis, P., *SEEK2: A generalised approach to automatic Knowledge based requirement*, in **proc. 9th IJCAI**, 1985.

Harmon, P., and King, D., *Expert Systems*, (pub.) John Wiley & Sons, Inc., 1985.

Hayes, P., *Computation and deduction*, in **proc. of MFCS Symposium**, Czech Academy of Sciences, 1973.

Hayes-Roth, F., Waterman, D. A., Lenat, D. B.(eds.), *Building Expert Systems*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1983.

Jackson, M. A., *Principles of Program Design*, Academic Press, 1975.

Johnson, P., and Gruber, S., *Specification of Expertise: Knowledge Acquisition for Expert Systems*, **Proc. of the Knowledge Acquisition for KBS Workshop**, Banff, Canada, Banff, Canada, 1986.

Kahn, G. S., Breaux, E. H., Joseph, R. L., and Del Clerk, P., *An intelligent Mixed-Initiative Workbench for Knowledge Acquisition*, **Proc. of the Knowledge Acquisition for KBS Workshop**, Banff, Canada, Banff, Canada, 1986.

Kampel, I., *A Practical Introduction to the New Logic Symbols*, (pub.) Butterworths, 1985.

Klinker, G., Bentolia, J., Genetet, S., Grimes, M., and McDermott, J. *KNACK - Report Driven Knowledge Acquisition*, **Proc. of the Knowledge Acquisition for KBS Workshop**, Banff, Canada, Banff, Canada, 1986.

Maes, W., and Demeyer, K. W., and Dupas, L. H., *Simpar - A versatile Technology Independent Parameter Extraction Program using a new Optimized-Fit-Strategy*, in **IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems**, Vol. 5, no. 2, pp. 320-5, 1986.

Maibaum, T. S. E., *Role of Abstraction in Program Development*, in **H.-J. Kugler(ed) Information Processing**, Elsevier, 1986.

MARCUS, S., *Taking Backtracking with a Grain of SALT*, **Proc. of the Knowledge Acquisition for KBS Workshop**, Banff, Canada, Banff, Canada, 1986.

McDermott, J., *R1: A Rule-Based Configurer of Computer Systems*, Carnegie-Mellon University Report CS-80-119, April 1980.

McDermott, J., *R1's Formative Years*, in AI Magazine, vol.2, pp. 21-29, Summer, 1981.

Morik, K., *Acquiring Domain Models*, in Proceedings of the Knowledge Acquisition for KBS Workshop, Banff, Canada, 1986.

Musen, M., Fagan, L. M., Combs, D. M., and Shortliffe, E. H., *Using a Domain Model to Drive an Interactive Knowledge Editing Tool*, Proc. of the Knowledge Acquisition for KBS Workshop, Banff, Canada, Banff, Canada, 1986.

Neches, R., and Swartout, W. R., and Moore, J. D., *Enhanced Maintenance and Explanation of Expert Systems through explicit models of their Development*, in IEEE Transactions on Software Engineering, vol. 2, no. 2, pp 1337-51, University of Southern California, Institute of Information Sciences, Marina Del Ray, California, 1985.

Newell, A., *Heuristic Programming: Ill-Structured Problems*, in: Aronofsky (ed) Progress in Operations Research, (pub) Wiley, 1969.

Newell, A., *Physical Symbol Systems*, in Cognitive Science, vol. 4 pp. 135-183, 1980.

Newell, A., and Bell, c. G., *Computer Structures: Readings and Examples*, eds. Bell, C. G., and Newell, A., McGraw-Hill, McGraw-Hill Computer Science Series, New York, 1971.

Parnas, D. L., and Clements, P. C., *A Rational Design Process - How and Why to Fake it*, in IEEE Transactions on Software Engineering, vol. 12, no. 2, pp. 251-57, Dept of Computer Science, University of Victoria, Canada, 1986 Politakis, P., and Weiss, S., *Using Empirical Analysis to Refine Expert System Knowledge Bases*, in Artificial Intelligence, vol. 22, 1984.

Simon, H., *The Science of the Artificial*, MIT Press, Cambridge, MA., 1969.

Sembugamoortly, V., and Chandrasekaran, B., *Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems*, in Experience, Memory, and Reasoning, eds: Kolodner, J. L., and Riesbeck, C. K., pp.47-73, Lawerence Erlbaum, Hillsdale - NJ., 1986.

Shortliffe, E. H., *Computer-Based Medical Consultation: MYCIN*, (pub.) Elsevier, New York, 1976.

Shortliffe, E. H., Buchanan, B. G., and Feigenbaum, E. A., *Knowledge engineering for medical decision making: A review of computer based clinical decision aids*, in **Proc. of the IEEE 67**, pp. 1207-1224, 1979.

Steels, L., and Van de Valde, W., *Learning in Second Generation Expert Systems*, in **Knowledge Based Problem Solving**, ed. Kowalik, Prentice-Hall, Englewood Cliffs, NJ., pp. 270-95, 1986.

Steels, L., *The Deepening of Expert Systems*, in **AI Communications**, eds. Wielinga, B., and Steels, L., Artificial Intelligence Laboratory, Free University of Brussels, pub. North Holland, Aug. 1987.

Stefik, M., and Bobrow, D.G., *Object Oriented Programming: Themes and Variations*, in *AI Magazine*, Spring edition, pp. 40-62, Intelligent Systems Laboratory, XEROX Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, Ca. 94304, 1986.

Sterling, C. H., *International Communications and Information Policy*, ed. Sterling, C. H., Washington, D. C., Communications Press, 1984

Sternberg, R. J., *Sketch of a Componential Sub-theory of Human Intelligence*, in **Behavioural and Brain Sciences**, vol. 3, no. 4, pp. 573-84, Dept. of Psychology, Yale Univ., New Haven, CT, 1980

Sowa, J. F., *Conceptual Structures: Information Processing in Mind and Machine*, the Systems Programming Series, Addison-Wesley Publishing Company, 1984.

Sutcliffe, A., *Jackson System Development*, UMIST, Prentice Hall, 1st pub., 1988.

van Melle, W., *A Domain-independent Production rule system for consultation programs*, **Proc. 6th IJCAI**, pp. 923-925, Tokyo, Japan, 1979.

Warden, R., *Integrating KBS into information systems: the challenge ahead*, **proc. first conf. for KBS in Government, Central Computer and Telecommunications Agency(CCTA)**, pp. 19-36, November 1987.

Wielinga, B., and Breuker, J., *Interpretation of Verbal data for Knowledge Acquisition*, **Report 1.4**, Memorandum 27 of the research project **The Acquisition of Expertise**, University of Amsterdam, Dept. of Social Science Informatics and Laboratory for Experimental Psychology, Weesperplein 8, 1018 XA Amsterdam, June 1984.

Wielinga, B., and Breuker, J., *Models of Expertise*, in **Proc. of European conf. on Artificial Intelligence(ECAI)**, Brighton, England, July 1986.

Winston, P. H., *Artificial Intelligence*, MIT, Addison-Wesley Publishing Company, 2nd ed., July 1984.

Yourdon, E., and Constantine, L., *Structured Design: A discipline of Computer Program and System Design*, (pub.) Yourdon Press, 1978.

Appendix A: a KADS Prototype for COMPARE Knowledge Source:

11. A Brief Description

In this appendix we shall consider the implementation of *compare* knowledge source used within the inference structure of ECGD in chapter 7 (Davoodi, 1986[b]). The knowledge source is implemented in CRYSTAL which is a low level shell, a previous version of the KS has been implemented in KEE(Davoodi[a]).

The prototype has been initially used as part of the 'empirical' arm of the investigation into the nature of KADS design language. It provides an opportunity to examine the relationship between knowledge sources in the analysis output and their counterparts in implementation. At the end of this process, it is possible to abstract lessons from this to decide on the nature of some of the aspects of the design phase. The prototype should also provide a sense of empirical backing for the appropriateness of the analysis phase, and thus the conceptual model, for developing a kbs.

11.1. Implementation

The *compare* knowledge source will decide whether an exporter's application should be underwritten or not. The knowledge source takes as input the financial details of a case which are classed under the metaclass 'parameter', the output of the process are 'discrepancy' and 'decision' metaclasses. The type of a decision given is based on the value of 'discrepancy' which is the difference between the underwriting 'norm' (the second input to the knowledge source) and 'parameter'.

In the code for the prototype appearing in the next sub-section, four types of decisions are given:

- *NO*
- *YES*
- *REFER*
- *INSUFFICIENT DATA*

Each of the above decisions is based on the values given for each parameter by the user of the system.

11.1.1. The Rules

Crystal is a production rule based expert system shell, in which the satisfaction of each rule will depend on the satisfaction of every node rules hanging from it. This is a recursive process ultimately ending in the leaf rules which are the basic means of evaluating each branch of a rule. Having satisfied the relevant leaves, Crystal will then performs a backward chaining process until all the rules including the top rule(s) are satisfied.

In our case *compare* knowledge source comprises three clauses *compare1*, *compare2*, *compare3*, the satisfaction of one or more of which will result in a "YES" decision. A "NO" decision is given if all three clauses fail decisively, in marginal cases a

."REFER" decision is given. If there is not enough data to exploit any of the three 'compares,' then the prototype will signal that there is "INSUFFICIENT DATA".

Following is the listing of the rules, after which a series of runs of the prototype will appear.

```
[   1] Analyse case
         + IF    [  25] Initialise
         +   AND [  41] Instantiate Parameters
         +   AND [  10] Compare
         +   AND [   2] Analyse Decision
             AND        DO: Menu Question  Response$
                        Would you like to consider another case?
                          {   YES   }
                          {   NO    }

           AND          DO: Test Expression
                        Response$="YES"
           AND          DO: Restart Rule

         OR             DO: Display Form
                          This brings the consultation session to an end.
                          We hope that our services have been of help in
                          providing an underwriting "decision support".

                          CALCULATED RISK IS A SOUND BUSINESS STRATEGY
                           BENEFITING OUR CLIENTS, AND ENSURING  A
                            CONTINUED SERVICE FROM US IN YEARS
                                    TO  COME!

[   2] Analyse Decision                                          Sp
         + IF    [  46] Top Decision Order
         +   AND [  24] Final Decision
         +   AND [  15] Decision Priority

[   3] Buyer Competence in Region                                Sp
             IF         DO: Test Expression
                          (buyer_competence<=10)&(buyer_competence>=0)

[   4] Buyer Exposure in Region                                  Sp
             IF         DO: Test Expression
                          (buyer_exposure<=10)&(buyer_exposure>=0)

[   5] Buyer Honesty in Region                                   Sp
             IF         DO: Test Expression
                          (buyer_honesty<=10)&(buyer_honesty>=0)

[   6] Buyer Market Condition in Region                          Sp
```

```
IF          DO: Test Expression
              buyer_market_condition<=10
  AND         DO: Test Expression
              buyer_market_condition>=0
```

```
[   7] Check1 All Parameters Present                    Sp
       IF          DO: Test Expression
                     ((buyer_honesty>0)&(buyer_competence>0))
         AND         DO: Assign Variable
                     D1_All_Parameters_Present$:="YES"
```

```
[   8] Check2 All Parameters Present                    Sp
       IF          DO: Test Expression
                     ((buyer_exposure>0)&(buyer_market_condition>0))
         AND         DO: Test Expression
                     ph_competence>0
         AND         DO: Assign Variable
                     D2_All_Parameters_Present$:="YES"
```

```
[   9] Check3 All Parameters Present                    Sp
       IF          DO: Test Expression
                     ((buyer_exposure>0)&(buyer_competence>0))
         AND         DO: Test Expression
                     ((buyer_market_condition>0)&(ph_competence>0))
         AND         DO: Assign Variable
                     D3_All_Parameters_Present$:="YES"
```

```
[  10] Compare                                          Sp
       + IF    [  19] Evaluate Discrepancies
       +   AND [  11] Compare1
       +   AND [  12] Compare2
       +   AND [  13] Compare3
```

```
[  11] Compare1                                         Sp
       + IF    [   7] Check1 All Parameters Present
         AND         DO: Test Expression
                     D1_All_Parameters_Present$="YES"
       +   AND [  16] Evaluate Decision1

         OR          DO: Assign Variable
                     Dec1:=-1
```

```
[  12] Compare2                                                    Sp
        + IF    [   8] Check2 All Parameters Present
          AND          DO: Test Expression
                       D2_All_Parameters_Present$="YES"
        +   AND [  17] Evaluate Decision2

          OR           DO: Assign Variable
                       Dec2:=-1


[  13] Compare3                                                    Sp
        + IF    [   9] Check3 All Parameters Present
          AND          DO: Test Expression
                       D3_All_Parameters_Present$="YES"
        +   AND [  18] Evaluate Decision3

          OR           DO: Assign Variable
                       Dec3:=-1


[  14] Decide Buyer Exposure vs Market Condition                    Sp
            IF         DO: Test Expression
                       ((D2_b_e>=0)&(D2_b_m_c>=0))
             AND       DO: Assign Variable
                       Dec2:=2

           OR          DO: Test Expression
                       D2_b_e>=0
             AND       DO: Assign Variable
                       Dec2:=1

           OR          DO: Assign Variable
                       Dec2:=0

[  15] Decision Priority                                           Sp
            IF         DO: Test Expression
                       Decision$="YES"
             AND       DO: Display Form
                       We Recommend a   "YES"  decision.

                       The recommendation is based on careful analysis of
                       the client's financial attributes.  We find that
                       the data volunteered would leave the underwriter
                       within the acceptable / reasonable bounds of
                       business risk.

           OR          DO: Test Expression
                       Decision$="REFER"
             AND       DO: Display Form
                       We should like to pass judgement on this case, and
                       "REFER" the decision completely to the under-
```

writer. Meanwhile, the underwriter should be able to refer to the system's knowledge base and use processed data in arriving at his decision. Our reason for REFERRING the case to the underwriter stems from the fact that the financial attributes volunteered are not sufficiently favourable / un-favourable to warrant a definite recommendation. Updated data may be submitted for reconsideration.

OR DO: Test Expression
Decision$="NO"
AND DO: Display Form
" NO "
The financial attributes volunteered in this case point to an UNACCEPTABLE amount of risk to be undertaken by the underwriter. I would, therefore, suggest that the case should be rejected. If you find, on revision and further investigation of the case that some of the attributes can be improved, you should resubmit the case for reconsideration. Meanwhile, the processed data within the system can be referenced for future purposes.

OR DO: Test Expression
Decision$="Insufficient Data"
AND DO: Display Form
"INSUFFICIENT DATA"
A number of financial attributes are missing, and it, therefore, is not possible to make a realistic assessment of the case. The missing data are those attributes for which you entered "0" (for UNKNOWN).
You can resubmit the case once you have sufficient data on the missing attribute values.

[16] Evaluate Decision1 Sp
IF DO: Test Expression
((D1_b_h>=0)&(D1_b_c>=0))
AND DO: Assign Variable
Dec1:=2

OR DO: Test Expression
((D1_b_h>=-2)&(D1_b_c>=0))
AND DO: Assign Variable
Dec1:=1

OR DO: Assign Variable
Dec1:=0

```
[  17] Evaluate Decision2                                    Sp
         IF          DO: Test Expression
                      D2_ph_c>=0
       +   AND [  14] Decide Buyer Exposure vs Market Condition

         OR          DO: Assign Variable
                     Dec2:=0
```

```
[  18] Evaluate Decision3                                    Sp
         IF          DO: Test Expression
                      ((D3_b_c>=0)&(D3_ph_c>=0))
       +   AND [  23] Exposure vs Market Condition

         OR          DO: Assign Variable
                     Dec3:=0
```

```
[  19] Evaluate Discrepancies                                Sp
       + IF    [  20] Evaluate Discrepancy1
       +   AND [  21] Evaluate Discrepancy2
       +   AND [  22] Evaluate Discrepancy3
```

```
[  20] Evaluate Discrepancy1                                 Sp
         IF          DO: Assign Variable
                      D1_b_h:=buyer_honesty-N1_b_h
         AND          DO: Assign Variable
                      D1_b_c:=buyer_competence-N1_b_c
```

```
[  21] Evaluate Discrepancy2                                 Sp
         IF          DO: Assign Variable
                      D2_b_e:=(10-buyer_exposure)-N2_b_e
         AND          DO: Assign Variable
                      D2_b_m_c:=buyer_market_condition-N2_b_m_c
         AND          DO: Assign Variable
                      D2_ph_c:=ph_competence-N2_ph_c
```

```
[  22] Evaluate Discrepancy3                                 Sp
         IF          DO: Assign Variable
                      D3_b_e:=(10-buyer_exposure)-N3_b_e
         AND          DO: Assign Variable
                      D3_b_m_c:=buyer_market_condition-N3_b_m_c
         AND          DO: Assign Variable
                      D3_b_c:=buyer_competence-N3_b_c
         AND          DO: Assign Variable
```

D3_ph_c:=ph_competence-N3_ph_c

[23] Exposure vs Market Condition Sp
IF DO: Test Expression
((D3_b_e>=0)&(D3_b_m_c>=0))
AND DO: Assign Variable
Dec3:=2

OR DO: Test Expression
D3_b_e>=0
AND DO: Assign Variable
Dec3:=1

OR DO: Assign Variable
Dec3:=0

[24] Final Decision Sp
IF DO: Test Expression
Decision=0
AND DO: Assign Variable
Decision$:="NO"

OR DO: Test Expression
Decision=1
AND DO: Assign Variable
Decision$:="REFER"

OR DO: Test Expression
Decision=-1
AND DO: Assign Variable
Decision$:="Insufficient Data"

OR DO: Test Expression
Decision=2
AND DO: Assign Variable
Decision$:="YES"

[25] Initialise Sp
+ IF [36] Initialise Parameters
+ AND [35] Initialise Norms
+ AND [27] Initialise Discrepancies
+ AND [31] Initialise Intermediate Decisions
+ AND [26] Initialise Decision Class

[26] Initialise Decision Class Sp
IF DO: Assign Variable

Decision$:="NO"

[27] Initialise Discrepancies Sp
+ IF [28] Initialise Discrepancy1
+ AND [29] Initialise Discrepancy2
+ AND [30] Initialise Discrepancy3

[28] Initialise Discrepancy1 Sp
IF DO: Assign Variable
D1_b_h:=0
AND DO: Assign Variable
D1_b_c:=0
AND DO: Assign Variable
D1_All_Parameters_Present$:="NO"

[29] Initialise Discrepancy2 Sp
IF DO: Assign Variable
D2_b_e:=0
AND DO: Assign Variable
D2_b_m_c:=0
AND DO: Assign Variable
D2_ph_c:=0
AND DO: Assign Variable
D2_All_Parameters_Present$:="NO"

[30] Initialise Discrepancy3 Sp
IF DO: Assign Variable
D3_b_e:=0
AND DO: Assign Variable
D3_b_m_c:=0
AND DO: Assign Variable
D3_b_c:=0
AND DO: Assign Variable
D3_ph_c:=0
AND DO: Assign Variable
D3_All_Parameters_Present$:="NO"

[31] Initialise Intermediate Decisions Sp
IF DO: Assign Variable
Dec1:=0
AND DO: Assign Variable
Dec2:=0
AND DO: Assign Variable

Dec3:=0
AND DO: Assign Variable
Response$:="NO"

[32] Initialise Norm1 Sp
IF DO: Assign Variable
N1_b_h:=8
AND DO: Assign Variable
N1_b_c:=6

[33] Initialise Norm2 Sp
IF DO: Assign Variable
N2_b_e:=4
AND DO: Assign Variable
N2_b_m_c:=4
AND DO: Assign Variable
N2_ph_c:=6

[34] Initialise Norm3 Sp
IF DO: Assign Variable
N3_b_e:=2
AND DO: Assign Variable
N3_b_m_c:=2
AND DO: Assign Variable
N3_b_c:=6
AND DO: Assign Variable
N3_ph_c:=8

[35] Initialise Norms Sp
+ IF [32] Initialise Norm1
+ AND [33] Initialise Norm2
+ AND [34] Initialise Norm3

[36] Initialise Parameters Sp
IF DO: Assign Variable
buyer_competence:=0
AND DO: Assign Variable
buyer_exposure:=0
AND DO: Assign Variable
buyer_honesty:=0
AND DO: Assign Variable
buyer_market_condition:=0
AND DO: Assign Variable
ph_competence:=0

[37] Instantiate Buyer Competence Sp
IF DO: Display Form
Your judgemental value for:
"BUYER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< buyer_competence >
buyer_competence

+ AND [3] NOT Buyer Competence in Region
AND DO: Restart Rule

OR DO: Succeed

[38] Instantiate Buyer Exposure Sp
IF DO: Display Form
Your judgemental value for:
"BUYER EXPOSURE"
0 for UNKNOWN, In this case values other than "0" should be seen as representing their complementary meaning. That is the higher the value, the lower will be it contribution: 1..2 for LOW, 3..4 for below average, 5..6 for average and above, 6..10 for HIGH.
Please Enter a Value between "0" to "10" Inclusive
< buyer_exposure >

buyer_exposure

+ AND [4] NOT Buyer Exposure in Region
AND DO: Restart Rule

OR DO: Succeed

[39] Instantiate Buyer Honesty Sp
IF DO: Display Form
Your judgemental value for:
"BUYER HONESTY"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< buyer_honesty >
buyer_honesty

```
+   AND [   5] NOT Buyer Honesty in Region
    AND         DO: Restart Rule

  OR            DO: Succeed
```

```
[  40] Instantiate Buyer Market Condition                Sp
         IF          DO: Display Form
                       Your judgemental value for:
                            "BUYER  MARKET CONDITION"
                       0 for UNKNOWN, 1..2 for LOW
                       3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
                       8..10 for HIGH


                       Please Enter a Value between "0" to "10" Inclusive
                                    < buyer_market_condition>
                     buyer_market_condition
```

```
+   AND [   6] NOT Buyer Market Condition in Region
    AND         DO: Restart Rule

  OR            DO: Succeed
```

```
[  41] Instantiate Parameters                            Sp
         IF          DO: Display Form
                       You are invited to enter the judgemental values
                       against parameters allowing the system to·re-
                       commend a decision.  We have allowed for a wide-
                       spread of these values between 1 to 10 to enable
                       you to provide a close approximation.  In cases
                       where the data is lacking, you can enter 0 for
                       'unknown'.

+   AND [  37] Instantiate Buyer Competence
+   AND [  38] Instantiate Buyer Exposure
+   AND [  39] Instantiate Buyer Honesty
+   AND [  40] Instantiate Buyer Market Condition
+   AND [  42] Instantiate Policyholder Competence
```

```
[  42] Instantiate Policyholder Competence               Sp
         IF          DO: Display Form
                       Your judgemental value for:
                            "POLICY HOLDER COMPETENCE"
                       0 for UNKNOWN, 1..2 for LOW
                       3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
                       8..10 for HIGH
```

Please Enter a Value between "0" to "10" Inclusive
< ph_competence >
ph_competence

+ AND [45] NOT Policyholder Competence in Region
AND DO: Restart Rule

OR DO: Succeed

[43] Introduce
IF DO: Display Form
UK Exporter Underwriting Decision Support
S Y S T E M

The system is aimed at supporting decision making by underwriters at ECGD within the Welsh office. Data required is that describing the exporter(potential policy holder(p/h)), and his overseas buyers. Cases will be underwritten in which the aggregate risks arising from the expoter and his clientels are judged reasonable.

[44] Main line control
+ IF [43] Introduce
+ AND [1] Analyse case

[45] Policyholder Competence in Region Sp
IF DO: Test Expression
(ph_competence<=10)&(ph_competence>=0)

[46] Top Decision Order Sp
IF DO: Assign Variable
Decision:=max(Dec1,max(Dec2,Dec3))

[47] CRYSTAL MASTER RULE
+ IF [44] Main line control

We shall provide six runs of the system, the first two of which will result in "YES" decisions, the next two will result in "NO" decisions. The fifth run will give "REFER" decision and the last one will signal to the user evidence of "INSUFFICIENT DATA" for any decision to be made.

11.1.2. The First Run

In this run clauses one and two of *compare* are satisfied, note that the satisfaction of one clause would have given the decision of "YES".

The first run:

UK Exporter Underwriting Decision Support
S Y S T E M

The system is aimed at supporting decision making by underwriters at ECGD within the Welsh office. Data required is that describing the exporter(potential policy holder(p/h)), and his overseas buyers. Cases will be underwritten in which the aggregate risks arising from the expoter and his clientels are judged reasonable.

You are invited to enter the judgemental values against parameters allowing the system to recommend a decision. We have allowed for a widespread of these values between 1 to 10 to enable you to provide a close approximation. In cases where the data is lacking, you can enter 0 for 'unknown'.

Your judgemental value for:
"BUYER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 7 >
buyer_competence

Your judgemental value for:
"BUYER EXPOSURE"
0 for UNKNOWN, In this case values other than "0" should be seen as representing their complementary meaning. That is the higher the value, the lower will be it contribution: 1..2 for LOW, 3..4 for below average, 5..6 for average and above, 6..10 for HIGH.
Please Enter a Value between "0" to "10" Inclusive
< 5 >

buyer_exposure

Your judgemental value for:
"BUYER HONESTY"

0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 8 >
buyer_honesty

Your judgemental value for:
"BUYER MARKET CONDITION"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 5 >
buyer_market_condition

Your judgemental value for:
"POLICY HOLDER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 7 >
ph_competence

We Recommend a "YES" decision.

The recommendation is based on careful analysis of the client's financial attributes. We find that the data volunteered would leave the underwriter within the acceptable / reasonable bounds of business risk.

Would you like to consider another case?

YES

11.1.3. The Second Run

In this run clause three of *compare* is satisfied to result in another "YES" decision.

The second run:

You are invited to enter the judgemental values against parameters allowing the system to recommend a decision. We have allowed for a widespread of these values between 1 to 10 to enable you to provide a close approximation. In cases where the data is lacking, you can enter 0 for 'unknown'.

Your judgemental value for:
"BUYER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 6 >
buyer_competence

Your judgemental value for:
"BUYER EXPOSURE"
0 for UNKNOWN, In this case values other than "0" should be seen as representing their complementary meaning. That is the higher the value, the lower will be it contribution: 1..2 for LOW, 3..4 for below average, 5..6 for average and above, 6..10 for HIGH.
Please Enter a Value between "0" to "10" Inclusive
< 8 >

buyer_exposure

Your judgemental value for:
"BUYER HONESTY"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 0 >
buyer_honesty

Your judgemental value for:
"BUYER MARKET CONDITION"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 7 >
buyer_market_condition

Your judgemental value for:
"POLICY HOLDER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 8 >
ph_competence

We Recommend a "YES" decision.

The recommendation is based on careful analysis of the client's financial attributes. We find that the data volunteered would leave the underwriter within the acceptable / reasonable bounds of business risk.

Would you like to consider another case?

YES

11.1.4. The Third and Fourth Runs

In the next two runs none of the three of *compare* clauses are satisfied thus resulting in a "NO" decision in both cases.

The third run:

You are invited to enter the judgemental values against parameters allowing the system to re-commend a decision. We have allowed for a wide-spread of these values between 1 to 10 to enable you to provide a close approximation. In cases where the data is lacking, you can enter 0 for 'unknown'.

Your judgemental value for:
"BUYER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 6 >
buyer_competence

Your judgemental value for:
"BUYER EXPOSURE"
0 for UNKNOWN, In this case values other than "0" should be seen as representing their complementary meaning. That is the higher the value, the lower will be it contribution: 1..2 for LOW, 3..4 for below average, 5..6 for average and above, 6..10 for HIGH.
Please Enter a Value between "0" to "10" Inclusive
< 6 >

buyer_exposure

Your judgemental value for:
"BUYER HONESTY"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 0 >
buyer_honesty

Your judgemental value for:
"BUYER MARKET CONDITION"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 1 >
buyer_market_condition

Your judgemental value for:
"POLICY HOLDER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 3 >
ph_competence

" NO "

The financial attributes volunteered in this case point to an UNACCEPTABLE amount of risk to be undertaken by the underwriter. I would, therefore, suggest that the case should be rejected. If you find, on revision and further investigation of the case that some of the attributes can be improved, you should resubmit the case for reconsideration. Meanwhile, the processed data within the system can be referenced for future purposes.

Would you like to consider another case?

YES

The fourth run:

You are invited to enter the judgemental values against parameters allowing the system to recommend a decision. We have allowed for a wide-spread of these values between 1 to 10 to enable you to provide a close approximation. In cases where the data is lacking, you can enter 0 for 'unknown'.

Your judgemental value for:
"BUYER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 3 >
buyer_competence

Your judgemental value for:
"BUYER EXPOSURE"
0 for UNKNOWN, In this case values other than "0" should be seen as representing their complementary meaning. That is the higher the value, the lower will be it contribution: 1..2 for LOW, 3..4 for below average, 5..6 for average and above, 6..10 for HIGH.
Please Enter a Value between "0" to "10" Inclusive
< 9 >

buyer_exposure

Your judgemental value for:
"BUYER HONESTY"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 2 >
buyer_honesty

Your judgemental value for:
"BUYER MARKET CONDITION"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 0 >
buyer_market_condition

Your judgemental value for:
"POLICY HOLDER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 3 >
ph_competence

" NO "
The financial attributes volunteered in this case point to an UNACCEPTABLE amount of risk to be undertaken by the underwriter. I would, therefore, suggest that the case should be rejected. If you find, on revision and further investigation of the case that some of the attributes can be improved, you should resubmit the case for reconsideration. Meanwhile, the processed data within the system can be referenced for future purposes.

11.1.5. The Fifth and Sixth Runs

The fifth run will result in a "REFER" decision, and the sixth one will indicate that the data available is insufficient to make a decision.

In this case clauses two and three will fail and clause one will result in a "REFER" decision.

The fifth run:

You are invited to enter the judgemental values against parameters allowing the system to recommend a decision. We have allowed for a widespread of these values between 1 to 10 to enable you to provide a close approximation. In cases where the data is lacking, you can enter 0 for 'unknown'.

Your judgemental value for:
"BUYER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 6 >
buyer_competence

Your judgemental value for:
"BUYER EXPOSURE"
0 for UNKNOWN, In this case values other than "0" should be seen as representing their complementary meaning. That is the higher the value, the lower will be it contribution: 1..2 for LOW, 3..4 for below average, 5..6 for average and above, 6..10 for HIGH.
Please Enter a Value between "0" to "10" Inclusive
< 6 >

buyer_exposure

Your judgemental value for:
"BUYER HONESTY"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 6 >
buyer_honesty

Your judgemental value for:
"BUYER MARKET CONDITION"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 1 >
buyer_market_condition

Your judgemental value for:
"POLICY HOLDER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 0 >
ph_competence

We should like to pass judgement on this case, and "REFER" the decision completely to the underwriter. Meanwhile, the underwriter should be able to refer to the system's knowledge base and use processed data in arriving at his decision. Our reason for REFERRING the case to the underwriter stems from the fact that the financial attributes volunteered are not sufficiently favourable / unfavourable to warrant a definite recommendation. Updated data may be submitted for reconsideration.

Would you like to consider another case?

YES

In this case every one of the *compare* clauses are able to deliver a decision due to lack of data, thus "INSUFFICIENT DATA" will be returned as the final decision.

The sixth and final run:

You are invited to enter the judgemental values against parameters allowing the system to recommend a decision. We have allowed for a wide-spread of these values between 1 to 10 to enable you to provide a close approximation. In cases where the data is lacking, you can enter 0 for 'unknown'.

Your judgemental value for:
"BUYER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 6 >
buyer_competence

Your judgemental value for:
"BUYER EXPOSURE"
0 for UNKNOWN, In this case values other than "0" should be seen as representing their complementary meaning. That is the higher the value, the lower will be it contribution: 1..2 for LOW, 3..4 for below average, 5..6 for average and above, 6..10 for HIGH.
Please Enter a Value between "0" to "10" Inclusive
< 6 >

buyer_exposure

Your judgemental value for:
"BUYER HONESTY"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 0 >
buyer_honesty

Your judgemental value for:
"BUYER MARKET CONDITION"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 1 >
buyer_market_condition

Your judgemental value for:
"POLICY HOLDER COMPETENCE"
0 for UNKNOWN, 1..2 for LOW
3..4 for below AVERAGE, 5..7 for AVERAGE and ABOVE
8..10 for HIGH

Please Enter a Value between "0" to "10" Inclusive
< 3 >
ph_competence

"INSUFFICIENT DATA"
A number of financial attributes are missing, and it, therefore, is not possible to make a realistic assessment of the case. The missing data are those attributes for which you entered "0" (for UNKNOWN).
You can resubmit the case once you have sufficient data on the missing attribute values.

Would you like to consider another case?

NO

This brings the consultation session to an end.
We hope that our services have been of help in providing an underwriting "decision support".

CALCULATED RISK IS A SOUND BUSINESS STRATEGY
BENEFITING OUR CLIENTS, AND ENSURING A
CONTINUED SERVICE FROM US IN YEARS
TO COME!

Appendix B: KBS, from Requirements to Design:

12. BOTAID

We shall use an example kbs in order to demonstrate the KADS methodology in terms of its analysis and design approach. Since we have had a number of case studies hitherto mostly dealing with the 'analysis' aspects of KADS, we shall put greater emphasis on the design aspects in this example.

BOTAID is a system intended to AID a BOTanist in identifying The correct species of plants, given that some or all of the characteristics of those plants are known. The following is the statement of requirements of the system:

- *The system should be able to perform the task of a botanist in identifying the spicies of a plant by applying the flora rules to the given characteristics of the plant.*

12.1. Analysis Phase

12.1.1. The Conceptual Model

The first three layers of the conceptual model are listed below. The fourth layer of 'flexible strategy' is not included since the task at hand does not require it.

12.1.1.1 Domain Layer

This will contain:

- entities/* representing plants */
- attribute & value pairs/* making up definitions of different types of plants */
- subsume relations/* relations between entities */
- heuristics for attribute selection.

12.1.1.2 Inference Layer

This will contain two knowledge sources, and corresponding metaclasses. It can be represented linearly as follows:
FINDINGS (/ attribute value pairs */)* ---> **classify** ---> *SOLUTION (/* object class*/)*
FINDINGS ---> **match** ---> *QUESTION (unknown discriminating attribute(s) *)*

```
Classify Plant
        obtain_data(object attributes)      /* Goal Statement */
        classify(object attributes)         /* knowledge source */
IF solution
THEN
        display(object class)               /* Goal Statement */
ELSE
        select_request(attribute)           /* request is guided by 'match' */
        display_request                     /* Goal Statement */
        Classify Plant                      /* Starting to do recursion */

Explain
        display(trace(classify))            /* Goal Statement */
```

12.1.2. External Requirements

External requirements are formulated as follows:

(1) The system should be able to acquire the flora rules from an existing database.

(2) The explanation should be a graphic path of the reasoning process.

(3) The user should be to converse with the system in a semi-natural language fashion.

12.2. Design Phase

12.2.1. Functional Layer

Figure 12-1 depicts the functional block structure supporting the analysis output of BOTAID. At The top level of the structure (level 1) BOTAID (Flora Expert) has four functional blocks, the aggregate of which satisfy the analysis *internal* and *external* views. The detail of each block, where appropriate, has been mapped out in terms of the functional blocks in the next level (level 2). We find that level 3 is the right level of abstraction for the functional blocks, thus we shall refrain from decomposing the blocks any further.

The description of each block is as follows:

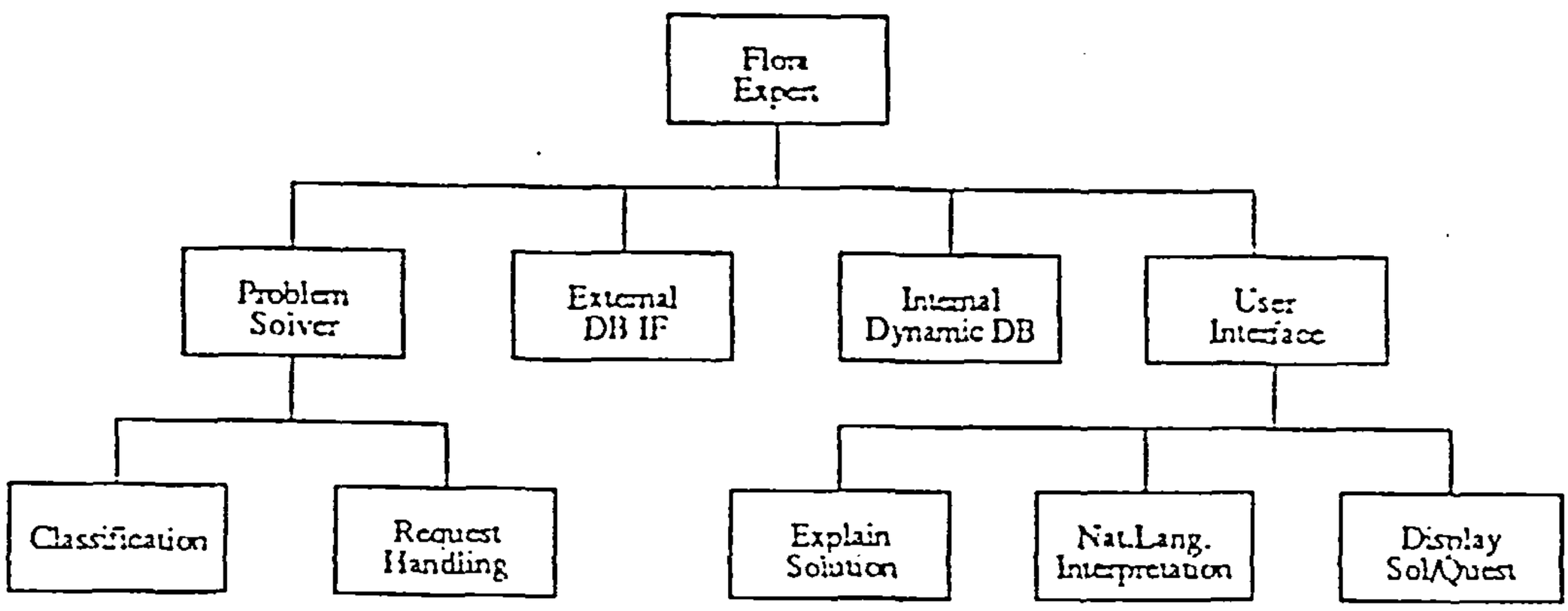

Figure 12-1: *The 'consist of' functional block diagram for BOTAID*

Level 1

User Interface

Sub Function of:	BOTAID(Flora Expert)
Function Type:	data I/O, explanation
Relation to Analysis:	obtain_data, display, and explanation goals
Input:	solution, attribute, solution path
Output:	attribute value pairs
External Interface:	user i/o in terms text and graphics display
Controls:	see sub-functions(below)
Controlled By:	see sub-functions(below)

Problem Solver

Sub Function of:	BOTAID(Flora Expert)
Function Type:	problem solving
Relation to Analysis:	match and classify knowledge sources
Input:	attribute value pairs, flora rules
Output:	solution, solution path, attribute
External Interface:	-
Controls:	see sub-functions(below)
Controlled By:	see sub-functions(below)

Internal dynamic database

Sub Function of:	BOTAID(Flora Expert)
Function Type:	data storage
Relation to Analysis:	place holder for findings and solution
Input:	attribute value pairs, solution path
Output:	attribute value pairs, solution path
External Interface:	-
Controls:	-

Controlled By: classification, request handling, explanation

External database interface
Sub Function of: BOTAID(Flora Expert)
Function Type: data I/O
Relation to Analysis: external requirement (1)
Input: plant class
Output: definition of sub-classes of the input plant class
External Interface: reads external flora database
Controls: -
Controlled By: classification

Level 2

Explain solution
Sub Function of: User Interface
Function Type: explanation
Relation to Analysis: explanation goal
Input: Solution path
Output: -
External Interface: responds to user's request for explanation
by giving a graphical representation of
the solution path
solution path path request to internal database
Controls: solution path request to internal database
Controlled By: -

Natural Language Interpretation
Sub Function of: User Interface
Function Type: data I/O
Relation to Analysis: obtain_data goal
Input: -
Output: attribute value pairs
External Interface: reads user's text
Controls: activates classification
Controlled By: -

Display solution / question
Sub Function of: User Interface
Function Type: data I/O
Relation to Analysis: display goal
Input: attribute, solution
Output: -
External Interface: display attribute question or solution path
Controls: -
Controlled By: activated by classifier or request handler

Classification
Sub Function of: Problem Solver
Function Type: problem solving

Relation to Analysis:	knowledge source classify
Input:	attribute value pairs, class definitions
Output:	solution
External Interface:	-
Controls:	asks external database for sub-class definitions asks internal database for attribute value pairs IF no solution -> activates request handler ELSE activate display function
Controlled By:	natural language interpretation

Request Handling

Sub Function of:	Problem Solver
Function Type:	problem solving
Relation to Analysis:	match knowledge source
Input:	class definitions, attribute value pairs
Output:	attribute
External Interface:	-
Controls:	asks internal database for attribute value pairs; activates display function
Controlled By:	classification

Figure 12-2 depicts the input-output relationships between the functional blocks at the lowest level (level 2). This type of figure is a useful add-on to the textual description of the functional blocks. It is a useful idea not to have any more than six to eight blocks to such a figure. If a greater number of blocks need to be represented in a such a way, then zooming in / out facilities should be used to make The diagram readable.

Figure 12-3 shows the control of the functional blocks in terms of a JSD diagram (Jackson, 1975). The conceptual model of BOTAID has a fixed task structure, in which case a JSD representation as that show below is sufficient to represent control.

12.2.1.1 Selection of Methods

We shall not describe every single method being used by the blocks, rather those which are significantly important and interesting.

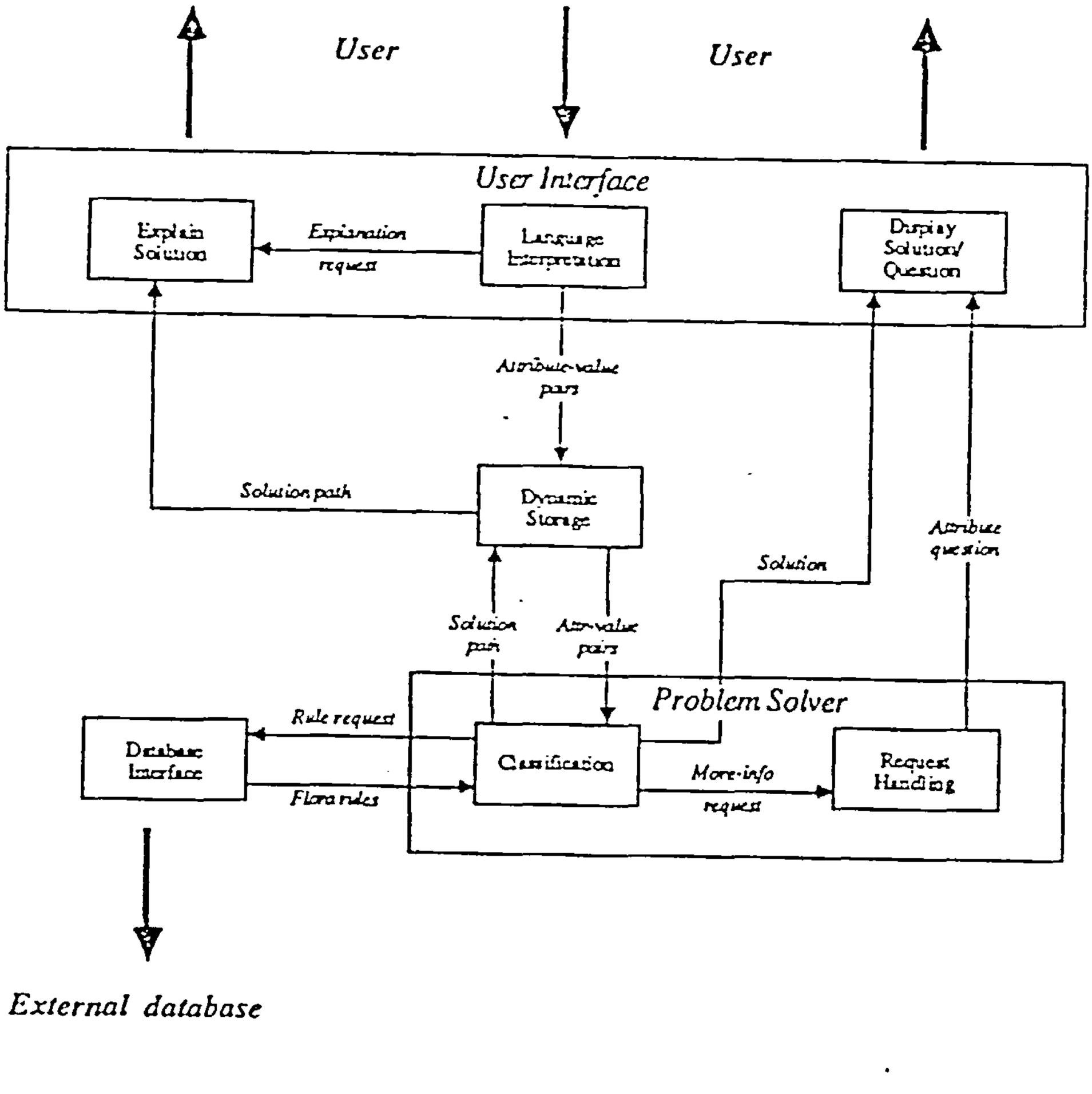

Figure 12-2: *Input-Output relationships between BOTAID lowest functional blocks*

Method: Hierarchical Classification

Description: Classification by refinement

Reference: Clancey(1985[c]), for instance

Assoc. method: Classification

Design Elements: Classifier procedure
Class definitions + subsumption relationships
Attribute value pairs

Method: Production System

Description: Situation, Conclusion / Action pairs

Reference: Buchanan, et al.(1984), for instance

Assoc. method: request handling

Design Elements: Rule interpreter
Heuristic rules for attribute selection

Method: ATN parsing

Description: Language parsing using
Augmented Transition Networks

Reference: Charniak, et al.(1985), for instance

Assoc. method: Natural Language interpretation

Design Elements: text string
ATN grammar
ATN parser
lexicon
parse tree

12.2.2. Physical Layer

12.2.2.1 Architecture

We shall use a simple skeleton architecture to describe BOTAID at the physical level. The architecture will consist of six (see fig. 12-4) physical modules:

- *knowledge base*
- *external database interface*
- *working memory*
- *inference procedure*
- *monitor*
- *user interface*

A number of physical modules consist of sub-modules. The main modules are described below.

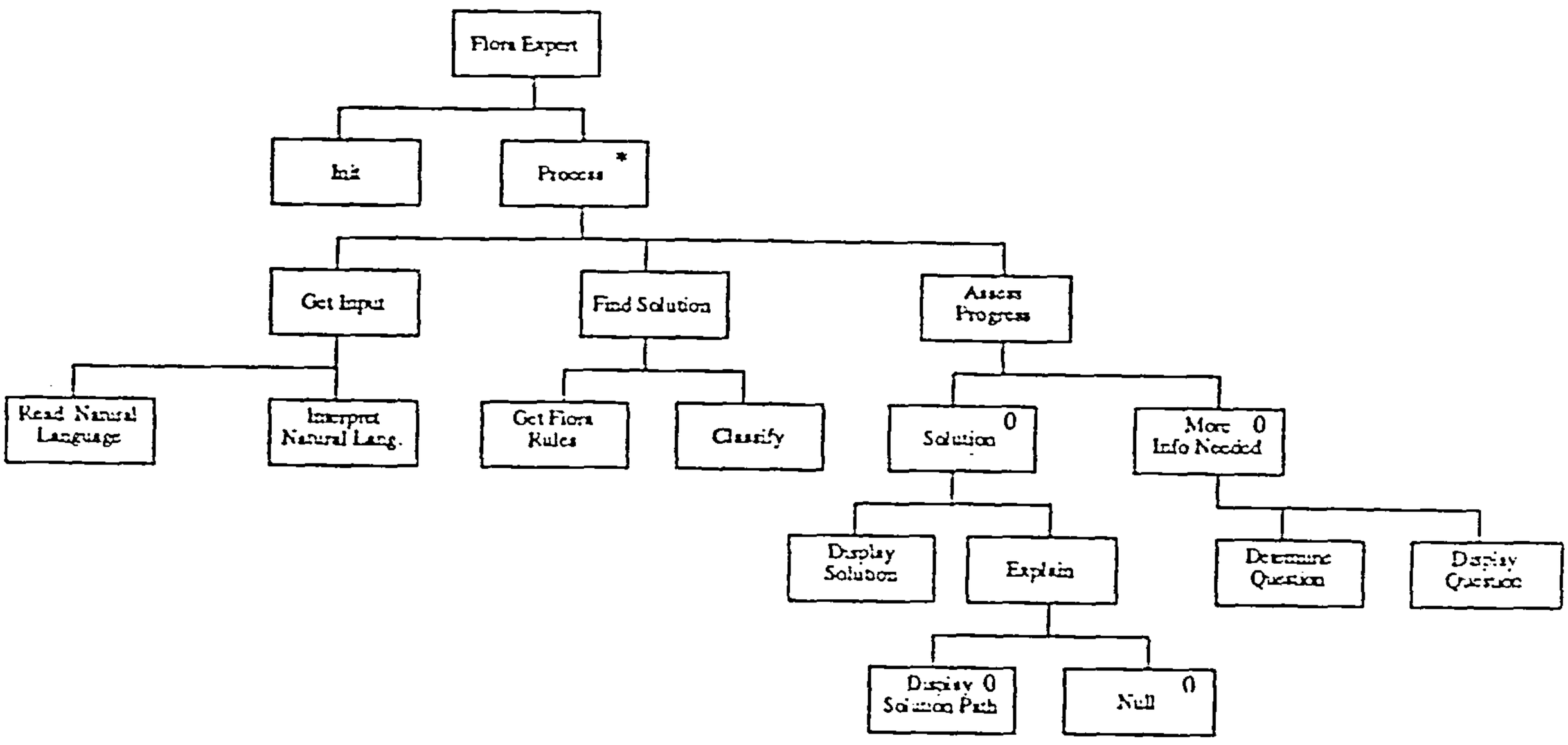

Figure 12-3: *JSD diagram of control in BOTAID*

Module: **knowledge base**

Description: Stores the static knowledge needed by the kbs

Design elements: ATN grammar
lexicon
production rules

Comp. principle: The three design elements will form three independent sub-modules of **knowledge base**

Access port: ATN grammar: start node of the network
lexicon: a word
production rules: condition part of rule

Module: **external database**

Description: Stores the static knowledge needed by the kbs

Design elements: flora class definitions

Comp. principle: dependent on the external database

Access port: node in the subsumption hierarchy

Module: **working memory**

Description: Stores global data

Design elements: attribute value pairs
solution path

Comp. principle: virtual (pointer to a) subsumption hierarchy

Access port: global access

Module: **inference procedures**

Description: Stores the 'active' design elements for BOTAID

Design elements: ATN parser
rule interpreter
classifier

Comp. principle: The three design elements will form three independent sub-modules of **inference structure**

Access port: ATN parser: text string
rule interpreter: request
classifier: attribute value pairs +
node in subsumption hierarchy

Module: **monitor**

Description: implements the overall control

Design elements: -

Comp. principle: see the control diagram in the functional description

Access port: new case description

Module: **user interface**

Description: handles every interaction with the user

Design elements: display explanation
display solution / question
read text + text string

Comp. principle: three independent sub-modules

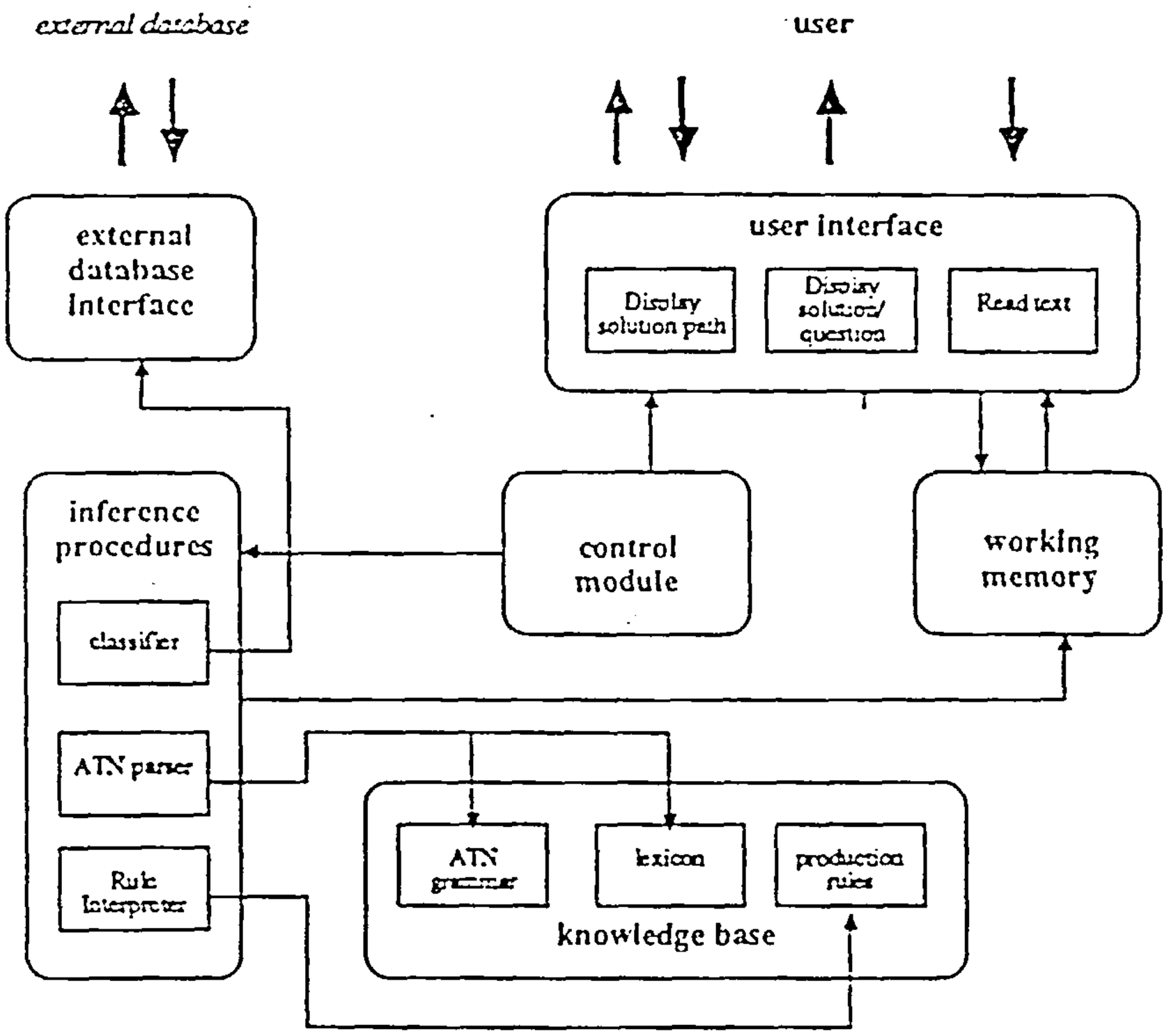

Figure 12-4: *Architecture of BOTAID*

Access port: display explanation: solution path
display solution / question: solution / question
read text + text string: user input

The choice of implementation vehicle or environment may be constrained by the type of software available. But if such a constraint is not present, the major guideline is the physical modules architecture which will indicate the use of an environment such as KEE. This will provide a high degree of textual and graphic interface, whilst also providing the means to develop a production system as well as 'objects' to represent some of the static knowledge.