SOFSEM '93 - Hrdonov, Sumava, Czech Republic, 21.11. - 3.12.1993

KNOWLEDGE ENGINEERING

Michael Wilson

SERC Rutherford Appleton Laboratory, Chilton, Didcot, Oxon, OX11 0QX, UK

Abstract: Knowledge Engineering is the aspect of systems engineering which addresses uncertain process requirements by emphasising the acquisition of knowledge about a process and representing this knowledge in a Knowledge Based System. The discipline has moved from one producing small expert systems to one producing embedded KBS in larger computational solutions.
This transition has resulted in the development of methodologies that guide the knowledge engineering of a product. The KADS methodology is described to show how structured knowledge engineering of a problem can be based on documented techniques for project management, analysis, and design. However, such techniques still result in the development of independent systems in isolated domains and do not optimise the re-use of general knowledge. The CYC project to represent a portable set of general knowledge that can be used as a basis for any novel system is described.
MMI^2 is the third project described which addresses the engineering of knowledge about communication itself, rather than a discrete problem domain, into the interface of computer systems to improve the interaction with the user, as well as the solving of domain problems.

Keywords: Knowledge Engineering, KBS Methodology, KADS, Cyc, MMI^2, Multimodal System.

1. INTRODUCTION TO KNOWLEDGE ENGINEERING.

If systems engineering is the discipline of producing software and hardware solutions to users' information needs, then sub-disciplines can be defined to address development techniques designed to minimise identifiable risks. Software engineering provides the mechanisms for validating the implementation of well specified algorithms. Human Computer Interaction provides analysis and design techniques based on prototyping of the user interface to address aspects of systems where the risks are associated with the users' needs, or the system usability. Data engineering addresses the permanent storage of large amounts of data and the efficient retrieval of the relatively small portion required for any process. In contrast, knowledge engineering addresses the structure of complex but ill defined processes where the solution to defining the process is to define the knowledge involved in the process explicitly in a knowledge based system (KBS).

Conventional software development follows the waterfall life cycle model. This requires complete system requirements at the start of development. Errors later in development can be fixed at little cost; errors at the start of development incur large costs. If the risks of failure of the project are associated with the efficiency of the implementation of a system this is appropriate. If the risk of a project failing is due to the uncertainty of the algorithms to perform the functions required, user requirements or enterprise objectives then an approach which is flexible at the start of the process is appropriate. Conventional software engineering approaches produce efficiently implemented code to execute algorithms to perform required functions which will always produce the correct outcome for correct input. The knowledge engineering approach allows users and experts to describe requirements and methods to perform the required functions at a high level close to the one in which they think about the task: the Knowledge Level. These can then be presented back to them for validation of the content, and modification.

If algorithms to perform the required functions cannot be determined then heuristics which produce correct outcomes sufficiently often for some task requirements can be used - there may not be

sufficiently detailed domain theory to supply algorithms so human expertise in the domain can be used. If heuristic knowledge cannot be acquired which produces correct outcomes sufficiently frequently then the project should be terminated - there may not be domain expertise to acquire. Since this possibility continues after initial problem definition (including feasibility studies) into the acquisition of knowledge, then staged contracting should be used to protect the client, and the commitments made by the developer.

Knowledge Engineering (the process of developing KBS) differs from conventional software engineering mainly at the early stages of the life cycle when user requirements and functional methods (or Knowledge) are being acquired. The tools for implementation, user interface design, testing, maintenance and updating systems may differ, but the principles which govern all software systems are the same. Therefore, although the early stages of knowledge acquisition will involve a knowledge engineer and a (or more) domain experts, later stages will involve software engineers for implementation/integration.

Since the expert's knowledge must be represented at the Knowledge Level rather than at lower computationally efficient levels during knowledge acquisition, the computational representation at high levels normally persists throughout a KBS life cycle. Therefore the delivery process for KBS is not the same as for normal software. It is generally constrained into a triptych architecture including a knowledge base containing a representation of the acquired knowledge, an inference engine which reasons over this knowledge, and a data area to store the data from a particular "run" of the system. This allows the processing to be separated away from the knowledge and isolated in the inference engine, leaving the knowledge free of procedures below the knowledge level.

Knowledge based system development has passed through an academic research phase from the late 1960's until the early 1980's. It then entered a phase dominated by feasibility studies based on small expert systems; this involved many companies exploring the technology, and much publicity for the topic which promoted promises that were not met. In 1982 there were thought to be only 30 prototype knowledge based systems in existence. In this stage it tried to discard the Run-Debug-Edit life cycle of single University researchers and develop a prototyping approach which would support commercial development teams. The approach used was mainly that of prototyping small systems with expert system shells to capture the knowledge of human experts in professional jobs. The present, third phase, started in the late 1980's and involves the production of practical systems through an engineering approach. These use development methods which divide the problem spaces, large toolkits which supply modules to execute knowledge in many ways, and most frequently result in small KBS incorporated into larger products. By 1992 over 2000 commercial knowledge based systems were in use (Touche Ross, 1992).

1.1 CURRENT PROBLEMS IN KNOWLEDGE ENGINEERING

Given the changes from the age of small expert system to the age of commercial KBS integrated into systems Knowledge Engineering must address the issue of reliable methodology to meet the practical engineering objectives it now has. Secondly, the systems produced through knowledge engineering methods must be able to re-use not only abstract ideas, but also implementation level knowledge. To do this issues of portability and interoperability must be addressed. A consequence of addressing these two issues could be to lose the apparent freedom provided by expert systems and to become bound by the formalities of software engineering. To avoid this, knowledge engineering must maintain its influence on user interfaces and the ability of KBS to explain their reasoning. These three issues will be explored in detail in the body of the paper by considering three projects, one of which addresses each.

2. KBS DEVELOPMENT METHOD - THE KADS PROJECT

Many attempts have been made in the last decade to produce KBS development methods (reviewed in Wilson et al 89) of which some have emphasised the integration of KBS and conventional systems such as the Gemini method from the UK. Of these, the most complete method which provides a

detailed description of the knowledge analysis and design stages of knowledge engineering is the KADS system.

The KADS KBS development method is the most developed Knowledge Engineering method which has been used on many projects within Europe and is starting to be taken up in the USA. The method has been developed through two major CEC Esprit funded research projects since 1984 as well as various small projects in individual companies. Through this time, the notations used in the method have changed, and elements have been included or removed from the method. Even the expansion of the acronym has changed during its development from "Knowledge Acquisition and Documentation System" to "Knowledge Acquisition and Design System" among other things; the present version of KADS is termed "Common KADS", where KADS is used as a name rather than an acronym. In this paper an overview of the development method, and the theoretical basis for it will be presented. It is not possible in the space available to describe the complete development method. Those attracted by this summary can find details of the method in three books: Tansley & Hayball (1993) described the development method for the knowledge engineer or system developer; Hickman et al (1989) describes the approach for development managers including the incorporation of KADS into Boehm's spiral model of development management; Schreiber et al (1993) describes the theoretical basis for the KADS method.

The central theme in KADS is modelling - the descriptions produced of the future systems and its domain are models. In Analysis one models the required behaviour of the system - specifying what the system will do. In Design, one models how the system is going to meet those requirements in terms of modules of program code - specifying the overall system architecture and how KBS components of the system will work (KADS does not support the design of non-KBS components). KADS includes the development activities required to produce seven models over the Analysis and Design Phases of development.

As well as modelling the behaviour of the system and the context within which it will work, KADS also provides a document framework for recording more general requirements and constraints. This should ensure that non-KBS areas integrate with the KBS components, and that requirements can be traced from different sources. The Analysis and Design phases of development are hierarchically decomposed into stages, which in turn are broken down into Activities. Each Phase, Stage and Activity is described in terms of the document which is the result of it. The complete breakdown and set of documentation is shown in Table 1.

As shown in table 1, the first stage of the KADS analysis is the Process Analysis. This starts by identifying the overall organisational process that the system will support. It then decomposes and describes this process using the conventional technique of Data Flow Diagrams (DFDs). This breaks down the process into high-level tasks which either the system or some other agent will perform. This stage is completed by assigning such agents to tasks and data stores. In particular, the knowledge bases tasks are identified. A Process Glossary complements the decomposition and distribution by acting as a data dictionary to the DFDs.

The results of the Process Analyses are taken by the Co-operation Analysis where the system boundaries are analysed in more detail, particularly, the user-system interface. The three required outputs of the analysis are: a detailed User Task Model describing the system's operations in terms of the user's tasks and the required interactions with the system; a System Task Model which describes the system's internal state changes and co-operation between internal system agents; a specification of the user-system interface itself (usually through a prototype). The fourth optional component is a User Model which describes the characteristics of the user agent.

The third and best developed model in KADS is the Expertise Analysis which investigates and describes the problem solving components of the knowledge based tasks identified in the Process Model. The Expertise Analysis builds up an Expertise Model from its four layers: the Domain Layer which describes the static factual knowledge about the domain; the Inference Layer which defines the inference steps which the KBS can make; the Task Layer which defines the basic problem solving tasks; and the Strategy Layer which defines how tasks are constructed, modified or chosen.

The development of the Expertise Model is the central knowledge engineering component of the KADS method which will be described in more detail after a summary of the remainder of the Analysis and Design Phases. Table 1 also includes the System Overview and the Constraints Analysis in the Analysis Phase which are 'maintenance type' activities. The System Overview maintains an up-to-date and overall description of the future system, whereas Constraints Analysis is just like conventional requirements analysis for mostly non-functional requirements.

Phase	Stage	Activity	Results
Analysis			Analysis Documentation
	Process Analysis		Process Model
		Analyse and Decompose Process	Process Decomposition+ Process Glossary
		Assign Tasks and Data Stores	Process Distribution+ Process Glossary
	Cooperation Analysis		Cooperation Model
		Analyse User Task(s)	User Task Model+ [User Model]
		Analyse System Task(s)	System Task Model
		Define User Interface	User Interface Spec.
	Expertise Analysis		Expertise Model
		Analyse Static Knowledge	Domain Structures
		Select Generic Task Model	Generic Task Model
		Construct Expertise Model	Domain layer+ Inference Layer+ Task Layer+ Strategy Layer+
	Constraints Analysis		Constraints Document
		Analyse Constraints	Environmental+ Technical+ Policy
	System Overview		System Overview Doc.
		Produce/Update System Overview	System Objectives+ System Functions+ System Structures+ Information Req's.
Design			Design Documentation
	Global Design		Global System Architecture
		Specify Sub-systems	Sub-system Definitions
		Specify Sub-system interfaces	Sub-system interface definitions
	KBS Design		KBS sub-system
		Define KBS Design Framework	KBS Design Framework
		Decompose KBS Functions	Function Blocks+ Domain/Data Model
		Assign Design Methods	Problem Solving Methods+ Knowledge Representation
		Assemble Design Elements	Module Specifications+ Module Interfaces

Table 1: All Stages, Phases, Activities and Results in the KADS KBS development method.

The KADS design phase is decomposed into two major stages of Global Design which breaks down the overall design problem into the KBS and non-KBS sub-systems, and the design of each of the

KBS sub-systems. KADS extends the model based approach to apply to these design stages and includes a library of models of generic problem solving approaches and algorithms, and knowledge representations which can be used to support the detailed KBS design. However, these will not be described in detail here, other than to say that they correspond to the models used in analysis which will now be described in more detail.

It was stated above that the Expertise Model and corresponding analysis was divided into four layers. This is true in that they are derived from Brahman's seven layer model of knowledge representation, but for practical purposes these layers can be viewed as being representations on a cycle of varying abstraction shown in Figure 1. The whole expertise model itself exists at a more abstract level than the task performed by a human in the domain, or the resultant system. In turn, although an instantiated expertise model is an implementation independent representation of the problem-solving capability of a prospective system, it is more concrete than an uninstantiated model would be. KADS provides a library of uninstantiated models for the inference and task layer called Generic Task Models which can be used to motivate the analysis and knowledge acquisition. Each of the four layers of the Expertise Model will be outlined before considering the process of developing them.

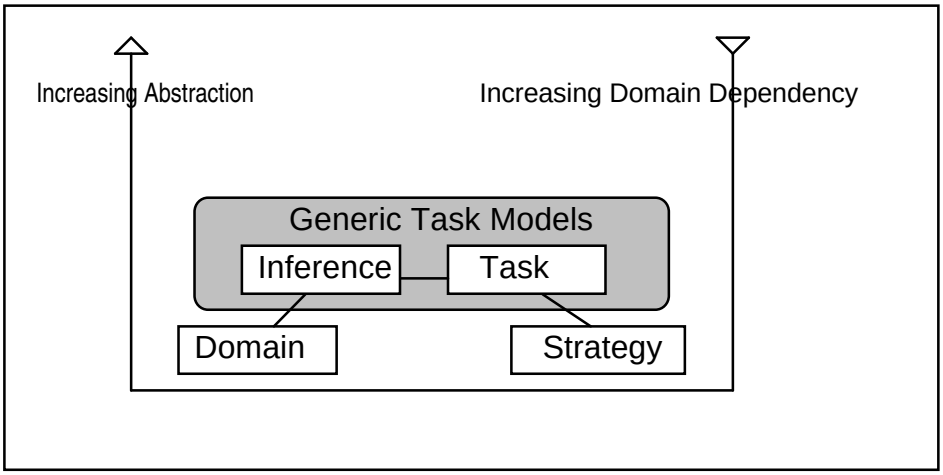

Figure 1: Levels of abstraction in the KADS Expertise Model

The Domain Layer is concerned with the definition of static domain knowledge, consisting of structures of domain concepts and relationships between concepts. This knowledge is termed 'static' since it is describes a domain while being neutral as to how it is used for inferencing. Domain concepts can be structured into ontologies using *isa* hierarchies, frames, rule sets or other representations. The structures of domain concepts are known in KADS as Domain Structures.

The Inference Layer describes the basic inferential capability in terms of inference types, domain roles and inference structures linking these for tasks at the task layer. It identifies the inferences that can be made over the static knowledge in the domain layer for selected tasks. 'Inference types' are descriptions of the way in which domain concepts, structures or relations can be used to make inferences. For example, the 'classify' inference type takes an object with its attributes and derives the class of the object. Inference types direct the way in which static domain knowledge may be used, and provide 'handles' for the control of inference by the next, Task Layer. 'Domain roles' define the functions which domain structures may perform. For example, in a diagnosis task in the medical domain, HIV may be either an hypothesis to be verified or a solution resulting from a reasoning process. These are two different roles for a single domain concept within a problem solving process. the third component in the layer is the 'inference structure' which is a network of inference types and domain roles which constrains a reasoning process by explicitly describing which inferences can be made, and implicitly defining which cannot be made. Inference structures are depicted in a graphical form where domain roles are represented by rectangles and inference types by ellipses (for example, Figure 2).

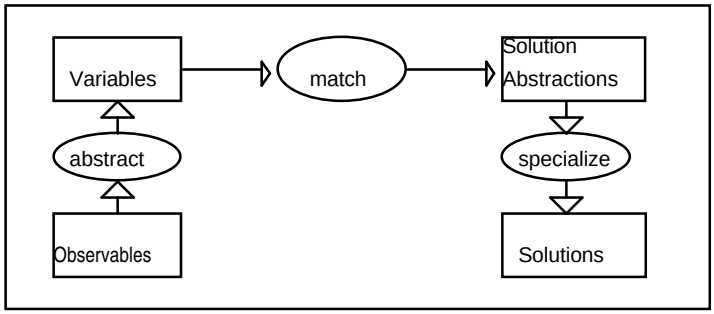

Figure 2: Inference Structure for Heuristic Classification

The third layer in the Expertise Model is the Task Layer which describes how the individual inferences within the Inference Layer may be sequenced in order to satisfy the required problem solving goals. The representation used is a task structure which can be defined statically (with a fixed control structure) or dynamically (e.g. as a result of planning at the inference layer). Task Structures are typically simple sequences of inferences wrapped in some conventional procedural control structures, such as selection (IF ... THEN... ELSE) and repetition (e.g. FOR, WHILE and REPEAT). However, they may include more complex controls structures such as parallelism, and temporal dependency.

The fourth layer of the Expertise Model is the Strategy Layer which provides the strategic knowledge to select, sequence, plan or repair the corresponding Task Structures. The following types of strategic knowledge may be described: Goal selection, Task structure selection, Goal sequencing, Task structure configuration, mode of KBS operation, inference control & repair.

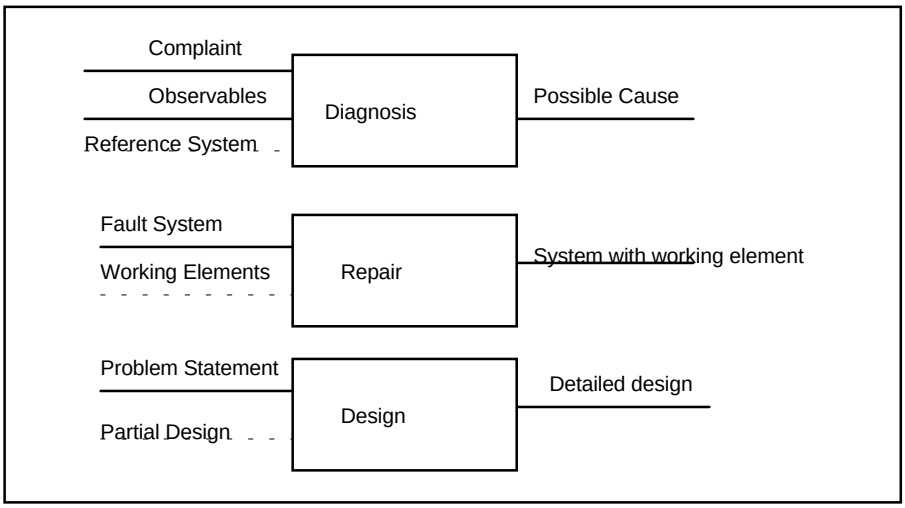

Figure 3: Input/Output descriptions of three task models (input on left, output on right; solid lines necessary, dashed lines optional).

So far I have described the KADS method as an engineering approach with a development structure with stages, resulting documents, models to be developed, and the decomposition of the Expertise Model has been described to show the notations which can be used. This allows Knowledge Engineers to approach problem solving tasks with poorly expressed algorithms in the same way as data intensive tasks may be approached by Data Engineers, or detailed algorithm implementation could be approached by Software Engineers. This moves knowledge engineering out of the prototyping stage in research departments and allows the integration of these techniques into serious software development. However, the major contribution of KADS takes it one stage further and addresses the identification of problem solving tasks and the re-use of task level descriptions through Generic Task Models.

- SYSTEM ANALYSIS
  - Identification
    - Diagnosis
      - Single Model Diagnosis
        - **Systematic Diagnosis**
          - **Localisation**
          - **Causal Tracing**
      - Multiple Model Diagnosis
        - **Mixed Mode Diagnosis**
    - **Verification**
    - **Correlation**
      - **Assessment**
    - **Monitoring**
    - Classification
      - **Simple Classification**
      - **Heuristic Classification**
      - **Systematic Refinement**
  - **Prediction**
    - **Predication of Behaviour**
    - **Prediction of Values**
- SYSTEM MODIFICATION
  - Repair
  - Remedy
  - Control
  - Maintenance
- SYSTEM SYNTHESIS
  - **Design**
    - **Hierarchical Design**
    - **Incremental Design**
  - Configuration
    - **Simple Configuration**
    - **Incremental Configuration**
  - **Planning**
  - **Scheduling**
  - Modelling

Table 2: The hierarchy of Generic Task Models in the KADS Task Library

Clancey (1985) had the insight that all heuristic classification tasks had a common inference structure, which is represented as a KADS inference structure in Figure 2. Since then several teams have attempted to categorise tasks at this level in order to provide a simple classification of the tasks addressed through knowledge engineering (e.g. Chandrasekaran, 1986, Steels, 1990 Wilson, 1989). KADS achieves this and provides a library of Generic Task Models (GTMs) which are models of problem solving tasks which are not tied to a domain. These GTMs are used to initiate and drive the knowledge acquisition process during Expertise Model development. For each task, the model describes it at the Task and Inference Layers (the shaded potion of Figure 1). Both the Domain and Strategic layers are strongly domain dependent and therefore little generic information can be given for them.

The library of GTMs is presented as a hierarchy. The top node represents tasks, the next layer has three entries: Systems Analysis which deal with an examination of the elements or structure of some entity; System Modification which deals with tasks which update or change some entity (often after a process of analysis and synthesis, but which modify the entity after finding a solution); thirdly, System Synthesis which deals with tasks which build up an entity from constituent parts. The full hierarchy of GTMs is presented in Table 2.

For each GTM the input and output descriptions are presented (see Figure 3) to aid the knowledge engineer in selecting the appropriate GTM for a system by encouraging him to consider the knowledge based tasks in the Process Model in terms of their inputs and outputs. GTM's are used as a starting point for developing Expertise Models. The selection of the GTM and the Analysis of Static Knowledge are the first two stages of the expertise analysis and are performed in parallel. Once the GTM is selected the specialisation of its terminology to that of the problem domain collected in the Analysis of Static Knowledge can begin. This population is then continued as the model develops through step-by-step refinement, eliciting knowledge from documents, task simulations and through structured interviews. It is likely that the expertise model will require some form of prototyping to experimentally test the hypothesis that the knowledge acquired is that appropriate for the task, and that it is sufficient.

3. COMMON KNOWLEDGE - THE CYC PROJECT

KADS presents a methodology to guide the construction of knowledge based systems and to motivate the analysis and knowledge acquisition. The Expertise model is divided into four layers, and abstract implementation independent libraries are provided for these to guide the knowledge engineer in identifying the structures, inferences and classes of domain knowledge expected for different tasks. This method reduces the uncertainty in knowledge acquisition and provides reuse of abstract descriptions of knowledge. However, KADS allows the use of many different forms of knowledge representation and each new system developed will be constructed from scratch as far as the implementation is concerned. The second major developments in knowledge engineering are to facilitate knowledge re-use or sharing at the level of implemented knowledge representations.

There are currently three major problems in re-using implemented knowledge. Firstly, there are a wide variety of approaches to knowledge representation, and knowledge that is expressed in one formalism cannot be directly incorporated into another formalism which prevents knowledge sharing at the implemented representation level. Secondly, if the knowledge was shared by communication between existing modules rather incorporating source code, interface protocols for interoperability would be required which do not exist. Thirdly, for knowledge sharing in either of these ways, or at a more abstract level, the ontologies would have to be compatible which they are not; for example, one type hierarchy might split the concept Object into Physical_Object and Abstract_Object, but another might divide Object into Decomposible_Object and Non_Decomposible_Object.

There are currently several large projects addressing the issues of knowledge sharing by addressing each of these three cases. In the USA, the ARPA & NSF Knowledge Sharing Effort (KSE) has been considering all of them since 1989 (Neches et al, 1991) by designing re-usable knowledge library structures using common knowledge representation schemes with translators between them, extensions to knowledge bases to support interoperability, portable reasoning processes to operate on the knowledge representation. The common reasoning elements, and interoperability supports will not be considered further here as we focus on re-usable common knowledge. Brachman and Levesque (1984) have shown that if a representation system is sufficiently expressive to be useful, then the inferences that a reasoner draws over it will be incomplete. The mechanisms employed to control this incompleteness will be similar to the domain roles and inference structures in the KADS methodology placed over the static domain knowledge, to allow the reasoning processes access to the knowledge.

The re-usable knowledge library structures of the KSE use a four layer structure with topic-independent fundamental models (or time, causality, etc.) at the bottom, application independent models of domain structures above this, then the application dependent layer of domain specific extensions to this developed on the shared layers, and finally the run time instances at the top layer created during the running of a knowledge base.

The KSE draws on public funds at several research institutes in universities and companies (Stanford, Maryland, Unysis, Bell Labs, etc.) who must compromise on the approach taken, with the consequence that it is too diverse to allow a clear explanation here. A second project at a single

commercial company the MCC Corporation using their own commercial funds is developing a very similar approach to re-usable knowledge by developing a general ontology corresponding to the bottom two layers of the KSE layered structure(Lenat et al, 1986; Lenat & Guha, 1990; Guha and Lenat, 1990, 1993). Consequently, the major difference between the KSE and Cyc projects is that the first advocates shared reusable knowledge bases built at different sites incorporating controlled heterogeneity, whereas Cyc advocates the development of a single representation using a single language as a homogenous and more manageable structure (although the Cyc ontology would be entirely suitable as one of the components reused by the KSE approach).

The purpose of the Cyc project is to produce an artefact which is an ontology of general, common-sense knowledge which can be used as a basis for knowledge engineering domain task solution, for guiding automated knowledge acquisition & machine learning, and for natural language processing. There are published disputes between the KSE and Cyc project representatives, and between the Cyc and more theoretical researchers. Cyc is engineering an ontology today, even though some of the philosophical problems about its structure, or AI problems about reasoning over it have not been theoretically solved (Quine, 1969).

Like KADS, Cyc is another long term project which started in the early 1980's and is continuing into the late 1990's, involving about 200 person years of effort. The size and duration have the consequence that the approach taken at the outset in many things is not still being used, as changes have had to be made to overcome problems. The major example of this is in the representation language used: CycL. This started as a frame system of mainly declarative knowledge (Lenat et al, 1986; Lenat & Guha, 1990) incorporating numerical certainty factors; it has become a constraint language representation divided between purely declarative knowledge (the Epistemological Level) and procedural inference knowledge (Heuristic Level) which has abandoned numerical certainty factors using reasoning by argumentation as a default reasoning mechanism (Guha and Lenat, 1990, 1993). The expressive power required to represent a general common-sense knowledge base has necessitated that the current representation language be like a first order predicate calculus (including disjunction, negation, universal and existential quantification, etc.) with some second order extensions (including both reification of individual proposition and reflection of the inferential processes in the language; modal operators, limited quantification over predicates, a context mechanism, etc.).

The current Cyc system contains about 2 million independent assertions explicitly represented as axioms in the CycL. For example, the following axiom states that "the human resources department of a company plays the primary role of mediating and hiring employees" (in this, (#%ForAll x S P) can be read as $(\forall x \in S)$ P(x); (#%ist c P) states that assertion P holds true in context c.):

```
(#%ist  #%LargeCorpInternalsMt
        (#%ForAll x (#%HumanResourcesDepartment #%allInstances)
                (#%actsInCapacity x #%mediatorInProcesses
                        #%EmployeeHiring #%MainFunction)))
```

To get a feel for the size and structure of the ontology, this is one of several thousand axioms in the LargeCorpInternalsMt microtheory, which is one of about two dozen microtheories which make up Cyc's knowledge of organisations. This is a substantial enlargement on the 100 to 200 rules normally found in a stand alone expert system built in a shell in the mid 1980's, or even the 7000 rules found in the most famous commercially used KBS, Digital Corporation's product configuration system XSEL.

The Cyc project is producing this large ontology structured into two layers. The upper layer which a user would interact with is the Epistemological Level, below this is a procedural layer containing specialised inference procedures as well as stated knowledge called the Heuristic Level. To access the Epistemological Level the user (or user knowledge base program) would use the CycL to describe statements, and a functional interface to control the operations on these. There are six operations at the functional interface (there are strong similarities between this two layered approach, the functional interface and the representation language and that used in the MMI^2 system described in the next section):

Assert ($\sum xKB \rightarrow KB$) : given a sentence and a knowledge base, the result is a knowledge base in which the sentence is an axiom.

Unasssert ($\sum xKB \rightarrow KB$) : Given a knowledge base containing an axiom, the result is the knowledge base with the axiom removed (this is the undo of assert).

Deny ($\sum xKB \rightarrow KB$) : Given a knowledge base containing an axiom, or from which a sentence is theorem (derivable by inference) the result is a knowledge base where the sentence is neither an axiom, nor is it derivable by inference.

Justify ($\sum xKB \rightarrow sentences$): If a sentence is true in a knowledge base, then the result is the minimal subset of the knowledge base from which it can be derived.

Ask ($\sum xKB \rightarrow truth-value \vee bindings$): Given a sentence and a knowledge base, the result is the truth value of the sentence, and the values for any free variables (similar to a Prolog goal).
Bundle: This provides a macro operation for sets of the other functions. Bundle allows some optimisation to take place on the ordering of the functions and their evaluation which would not be possible if they were executed independently.

At the Epistemological Level interaction is through these interfaces where sentences are expressed in CycL. The user views the knowledge base as a single KBS. Beneath this layer, the Heuristic Level efficiently implements reasoning procedures over what are divided knowledge bases called microtheories. The details of these inference procedures and the divisions between microtheories will not be explained here beyond saying that they resemble the inference and task layer interactions in the KADS development method. An important consequence of this microtheory division is that inferences are drawn from within microtheories and are not complete over the entire ontology (that is, they do not infer all the facts which can be derived from a general inference rule over the body of domain knowledge). Therefore, the inference mechanisms can be efficient, and importantly different tasks will result in different inferences from the same underlying domain knowledge. Indeed, an assertion can be true in the context of one task while being false in the context of another. The Epistemological Level is the one through which user, or control program interaction is undertaken.

This diversion into the internal structure of Cyc has been to show the form of interaction with it. The purpose in discussing Cyc is to illustrate the development of large portable ontologies which can be used as the basis for KBS in specific domains. The ontology of Cyc is organised around the concept of categories (also called classes or collections). The categories are organised in a generalisation/specialisation hierarchy (a directed graph rather than a tree since each category has several direct generalisations) (see Figure 4).

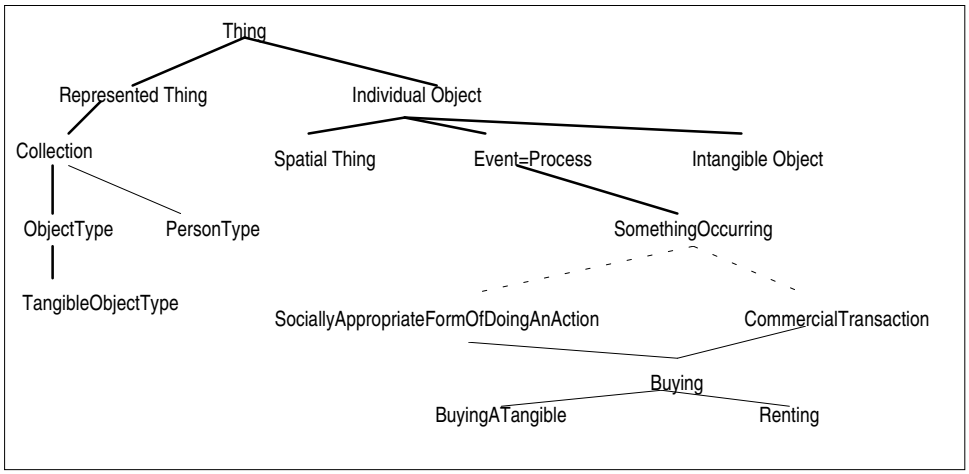

Figure 4: Some of Cyc's collections.

The major distinction in most ontologies is that between 'instances' (INST) and 'subtypes' (ISA) (i.e. MarvinMinsky INST ComputerScientist, ComputerScientist ISA Person). This simple distinction can be very confusing if considered in terms of mathematical sets. Cyc maintains the relations 'instance' (and the inverse 'instanceOf') and 'spec' (and the inverse 'genis'). If one looks at the distinction between IndividualObject and Collection, IndividualObjects include elements such as Fred, TheWhiteHouse and TheFourthofJuly1990 (that is non-sets), which can have parts but not instances. Collections in contrast have instances and are akin to sets, e.g. Chair (the set of all chairs), Buying (the set of all buying events). A second confusion which can arise concerns the legality of the attachment of attribute predicates to objects. For example, age and weight cannot be attached to Chair since it is a Collection, although they must be attachable to instances of the collection such as Chair905. As well as collections of individuals, Cyc also employs collections of collections, such as PersonType which includes Person, ComputerScientist, and Texan, which are themselves collections which include MarvinMinsky, etc.

This description could continue through the details of mass and count nouns, the representation of time, events and processes, the representation of proper nouns, the compositionality of objects into compound objects, and many other issues which knowledge engineers, linguists and philosophers can debate at length. Unfortunately, such a lengthy description would still not provide enough detail to use the Cyc system, nor would it enlighten you as to the underlying issues of knowledge engineering. Each of these and many other issues can be debated. Cyc has chosen a solution to each of them. It has also chosen to represent in its ontology a large number of general items. Not all items in the world have been represented, and not all the solutions chosen to these classic problems will be agreed with by all academics. Cyc has been developed to be a usable system for KBS development taking an engineering approach to these issues. A more important question then is, how does Cyc know when it has enough information or workable solutions to these problems?

As with most engineering solutions the answer is that it works so far. As with somebody jumping off a 20 storey building, they can say "So far, so good" until the last few inches of the fall. Will Cyc fail catastrophically when faced with a real problem, or will it degrade gracefully? The project has been running for nearly ten years. They have introduced a large number of rules and a large number of objects into the system. They aim to represent most of the knowledge found in a single volume desk sized encyclopaedia. They have had to change the representation language considerably since the project started, but it has been stable for several years now. They have had to add in new high level concepts into the ontology, but the tools exist to do this in man hours rather than man years. They have performed various test developments as the project continues and the ontology now appears to be stable enough to cope with them. Nobody knows the properties of the problem they are addressing enough to know if they are about to encounter a catastrophe. However, informed judgement suggests that they will not.

Cyc is not yet a commercially available tool. It is currently being used at about 20 sites who are developing research systems around it. The ontology is still being grown to meet the needs of these and to cover the planned scope of general knowledge. A recent test system was to produce an interface to a distributed heterogeneous database system for motor car retailing. The development process was faster than comparable planned projects and resulting system was both usable and efficient. Soon Cyc will start being used more generally as a practical ontology by knowledge engineers to provide a portable pre-existing ontology which will facilitate the re-use of low level knowledge in the way the KADS facilitates the reuse of abstract models of task knowledge to guide KBS development.

4. INTELLIGENT USER INTERFACES - THE MMI^2 PROJECT

The remaining attractive feature of early 1980's expert systems which has not been considered is that of the user interface and explanation. To many, expert systems introduced windows and mice into computing. Although, these user interface techniques existed before expert systems, integrated AI toolkits were amongst the first to popularise them. Now they are prevalent in most office systems. Expert systems also appeared intelligent because they could explain their reasoning and appeared to

communicate intelligently about the problem they were solving. As with the two previous topics of methods and ontologies, this was not a generalisable property which would scale up to large systems where the domain was complex or broad.

To explain the issues of co-operative user interfaces to KBS further, an example system will be described. The MMI^2 system was developed as the result of another large research project started in 1989 taking 60 person years of effort with the purpose of demonstrating the architecture and development method required to produce large scale co-operative interfaces to KBS (Binot et al, 1991). The demonstration task used in this system is that of designing local and wide area computer networks for institutions such as hospitals or universities. The overall architecture of the MMI^2 system is shown in Figure 6.

Knowledge engineers have tried to retain the intelligent interaction properties of expert systems by introducing explicit co-operative user interfaces onto domain KBS which themselves involve explicit knowledge of communication (see Wilson & Conway, 1991 for a review). These interfaces use multimodal systems to divide the presentation and input modes from the dialogue management and domain reasoning functions of the target system. Multimedia systems are an active area of development with new commercial products being released monthly. However, although multimedia systems can present graphics, video, or text, and can take keyboard, mouse or pen based input, they rely on specialised underlying representations for each medium. Multimodal systems construct their output from a single common logical meaning representation and decide the most effective and efficient way to communicate this to users, then construct the output form for the user. Similarly, they allow users to input information in a single mode (e.g. typing command language) or a combination of modes simultaneously (e.g. combining natural language input with mouse clicks as in "Put <click on object>that there <click on object>."). Figure 5 graphically shows the distinction between the combined or independent use of modes, the simultaneous or successive use of modes, and the use of an abstract meaning representation, or specific data representations by multimodal and multimedia systems.

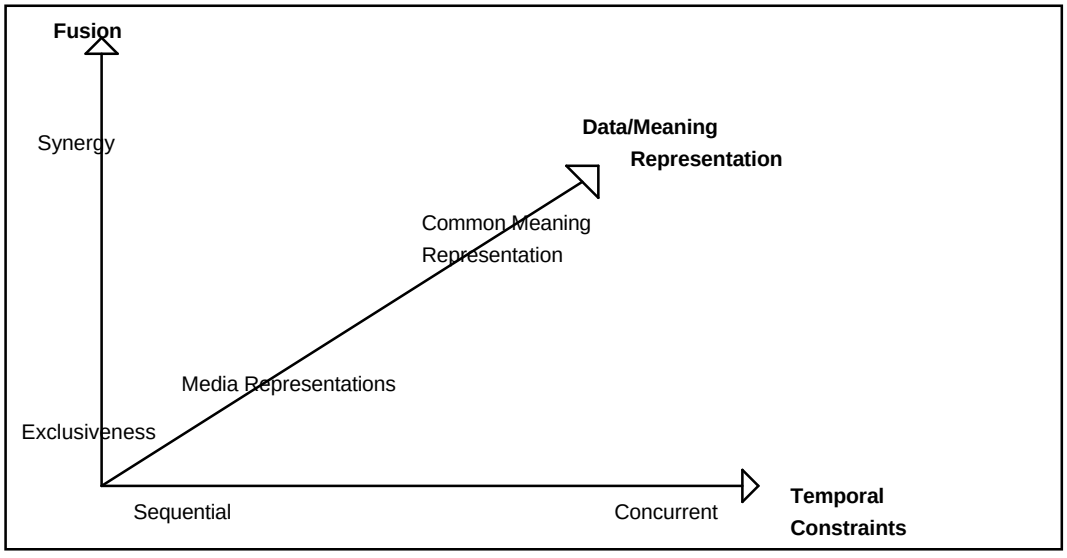

Figure 5: Space of interaction architectures for multimedia & multimodal user interfaces on three scales of Fusion of input and output media, Temporal Constraints on different media, and the abstractness of the Data/Meaning representation behind the media.

The freedom provided to systems and users by multimodal systems firstly relies upon the use of a common abstract meaning representation for all information sent to or received by all modes. The representation used for this must be able to express all such information, therefore it requires the expressiveness of the CycL language illustrated in the previous section. In the MMI^2 system the language is called the Common Meaning Representation (CMR) and is a first order logic with second order extensions, employing promiscuous reification of objects, actions and events. This language is used to pass between the Mode and Dialogue management layers of the MMI2 architecture, allowing

a clear interface where different modes can realise (generate images, language, etc.) any logical CMR description.

The mode layer of the MMI^2 system includes a window manager, and several modes. Each mode has a generator to produce the mode from system generated CMR and a parser to produce it from user input. The modes supported are English, French and Spanish natural language, Command Language, Audio, graphics for CAD diagrams, business graphics (charts, tables, pie charts, hierarchies), direct manipulation by the user on these, pen based gesture one these and the text modes. In addition to the modes, tools are required to render the output and receive input. The natural language modes use conventional natural language processing techniques, the graphics mode uses explicit knowledge about the design of graphic presentations drawn from design theory and designer's expertise to produce effective and efficient presentations (Chappel & Wilson, 1993).

The second necessity for a multimodal system which is synergistic and concurrent is that there is a common reference context for all objects. MMI^2 contains a Context Expert which stores all objects referred to in the dialogue and which provides the Dialogue Manager with candidates to resolve diexis and anaphora. The consequence of this is that each mode can refer to objects mentioned in other modes. These two requirements are sufficient to support multimodal interaction, but they do not in themselves support co-operative dialogue.

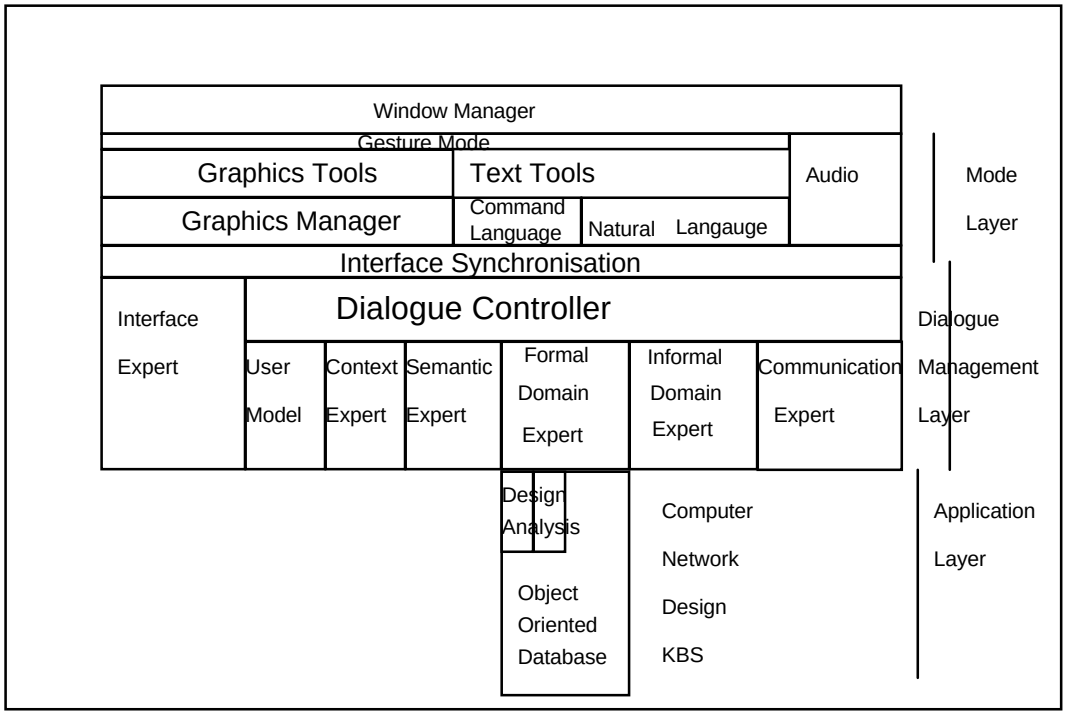

Figure 6: Architecture of the MMI^2 Multimodal Man-Machine Interface for Knowledge Based Systems.

MMI^2 does not include a broad knowledge base of common-sense knowledge such as that being developed in Cyc. However, like Cyc it must include more than just the limited domain knowledge for the demonstrator application for designing computer networks. Two other domains of knowledge are represented: the domain of the user, and the domain if the interface itself.

The User Model contains a model of the beliefs of the user (Chappel et al, 1992). It monitors all messages passing between the mode and dialogue management layers in CMR and extracts from them beliefs which the user holds (both correctly and incorrectly with respect to the knowledge stored in the KBS in MMI^2 which are assumed to be correct), and the intentions of the user. This user model then acts as a server to other parts of the system which require knowledge of the user, such as the graphics manager for planning effective graphics communication, the natural language generators for generating text, and the communications and informal domain experts. The user model is also

available for the evaluation or interpretation of predicates in CMR about the user (e.g. to answer questions such has "Who am I?").

The Interface Expert contains information about the interface itself. These are available to answer questions about the interface and its capabilities, but also for the evaluation of predicates in the CMR about the interface. For example, if the user asks the system to "draw a bar chart of the cost of computers on the network" then concepts such as BarChart are not network design concepts, but interface concepts; so that their evaluation is against the domain of the interface rather than network design.

The third domain is obviously the domain of the application itself, containing knowledge of computer network design. This application is accessed through the formal domain expert which provides a functional interface consistent with the User Model and Interface Expert. The application itself consists of an object oriented database which stores and object and instance hierarchy for the computer network domain similar to a subset of the Cyc ontology. In addition there are two sets of domain heuristics, for design, and for the analysis of a design. These were developed following the KADS methodology using task models for Hierarchical Design and Heuristic Classification operating on the domain objects in the object-oriented database. Above the Heuristic Level of the application itself, the Formal Domain Expert parallels the Epistemological Level in Cyc, or the Inference Layer in the KADS Expertise Model.

The functional interface provided to the knowledge bases in the Formal Domain Expert, User Model and Interface Expert includes three operations which correspond to the Assert, UnAssert and Ask functions in the Cyc functional interface: Assert, Retract and Goal.

The use of three domains of knowledge through a common functional interface allows the dialogue controller to be efficiently implemented and the system to be extensible through this common interface. The different domain allow users freedom to act and ask about more than just the core domain itself and provide some impression of cooperativity. Unfortunately, the addition of these two domains alone do not achieve the structure of a co-operative human-human conversation. To do this requires a representation of the Strategy and Task Levels from the KADS Expertise Model and knowledge of communication argumentation to present the information to the user co-operatively. An example will illustrate this point more clearly. When a user wishes to design a network they must state some essential requirements such as a description of the building, the number of machines required, and the cost of installation. There are also many optional requirements such as demands to promote extensibility, and constraints on some environments being hazardous to network performance (e.g. X-ray exposure). If the user fails to state all of the essential requirements, and asks the system to perform a design, then it will merely fail to produce a design. It is necessary to provide an explicit high level representation of the task model to ensure that the user is prompted for requirements, and that the requirements stated are not contradictory. These task plans are represented in the Informal Domain Expert. Before Goal predicates with free variables are interpreted against the Formal Domain Expert they are passed to the Informal Domain Expert for informal evaluation that pre-conditions on the task stage have been met. If they have not, then messages are passed back to the Dialogue Controller.

In order for the Dialogue Controller to express these or other statements to the user, they must be structured into complex messages to present the argument that is required in an effective way. To do this MMI^2 includes a communication planning module which turns sequences of functional interface level formulae into large CMR structures which can be passed to the modes by planning the argumentation structure of the message using knowledge of communication itself.

These facilities allow the user to communicate co-operatively with the system to achieve their task, and to investigate the system and the state of a static design. The aspect of explanation which expert systems possessed was to explain the reasoning they used. To provide this a set of explanatory predicates are provided which operate on temporary representations produced by the design and analysis heuristics to provide backward chaining traces of their reasoning (Justify, Expand, Elaborate, Define, Exemplify, Analogy) the output of these is also re-structured by the communication expert to produce co-operative responses to the user.

This description of MMI^2 has covered the architecture and five principles required for co-operative multimodal interaction:

1) The use of as single common meaning representation.

2) The use of a common context space for referents.

3) The use of multiple domains of expertise to support knowledge of the user and interface as well as the domain.

4) The use of pragmatic task plans in the informal domain expert to guide dialogue.

5) The use of expertise about communication planning in the communication expert, graphics manger and natural language modes to produce high quality, effectively planned communications.

5. CONCLUSION

In this paper the role of knowledge engineering has been described as the explicit representation of knowledge used in processes at the knowledge level, and the representation of inference processes which operate over it. Knowledge engineering has progressed from the 1980's stage of producing ad hoc small expert systems in shells to producing reliable KBS which can be incorporated into larger software and hardware systems.

The major foci of research at present are the establishment of development methods to ease knowledge acquisition and the integration of KBS with conventional systems development, and the development of portable, re-usable knowledge in the form of rules and ontologies. KADS has been described as a knowledge engineering development methodology which has a theoretically derived expertise model which identifies implementation independent representations for knowledge, and provides libraries to support the modelling activities of knowledge engineers. There are several other development methods within large corporations, or soon to be published by government agencies, but they follow the same overall structure.

In the development of implementation specific re-usable knowledge, the Knowledge Sharing Effort is developing standards for interworking heterogeneous knowledge bases following the model of heterogeneous database interaction. In contrast, the Cyc project is developing a standard ontology which can be used as the basis for many KBS projects, through the addition of domain specific layers on top of it, built using a compatible knowledge representation language. Using a broad common-sense core of knowledge such as that produced in Cyc as a basis for projects would be a start in reducing the brittleness of knowledge bases - that is their collapse of reasoning outside the very narrow domain in which they operate.

The remaining selling point of expert systems was the ability to explain themselves and present a co-operative user interface which gave the impression that the system was intelligent in a wider sense. The third strand of work described addresses the use of KBS within the user interface to support co-operative interaction through the representation of knowledge about communication itself and the user agent, and the use of a common meaning representation to support multimodal interaction.

In all three of these areas, development has been made in very large projects with many secondary developments testing the ideas and constructions presented, and the main project adjusting the artifact in the light of real use. This is essential if knowledge engineering is to develop because of the use of heuristics. KADS divides general domain knowledge into a lower layer than inferences or the tasks which control them. Cyc divides the Epistemological Layer of domain knowledge from the Heuristic Layer which control inferences over it for tasks. MMI2 divides the worlds of interface knowledge from that of domain knowledge, and divides the inference mechanisms of design and analysis from the underlying domain ontology. All three of these examples represent domain

knowledge and then employ heuristic reasoning over it. In all three cases different heuristic reasoning processes are used over the same underlying domain knowledge layer to perform different tasks. Since these are heuristic reasoning systems they are not complete logical theorem provers. They do not compute logical closure over the set of axioms - that is, they do not infer all the facts which can be derived from a general inference rule over the body of domain knowledge. The heuristics limit the search space and permit the reasoning to be computationally tractable. All three approaches illustrate the finding of Brachman and Levesque (1984) that if a representation system is sufficiently expressive to be useful, then the inferences that a reasoner draws over it will be incomplete. The consequence in all three of these systems is that they do not attempt to draw the complete set of inferences. They are only heuristically complete in that they effectively draw those inferences that a task demands. Different tasks will result in different inferences from the same underlying domain knowledge. Indeed, an assertion can be true in the context of one task while being false in the context of another. This enables local consistency without demanding global consistency across on entire method, knowledge base or ontology. In order to evaluate whether the set of inferences is complete for a task it is necessary to evaluate if the heuristic reasoning processes are available to support each task. This requires an engineering, statistical sampling of tasks and their required inference heuristics to be performed and checked against each approach (method or ontology).

The obvious aspect of knowledge engineering which this paper has not described is that of knowledge acquisition itself. Knowledge acquisition is usually the process of understanding the problem to be solved, the elicitation of knowledge from one or more experts in the problem area, the encoding of this knowledge in some mediating representation or prototype, and the refinement of this mediating representation by the expert until the body of knowledge is deemed complete and correct for the identified problem This was perhaps the area of greatest practical benefit in the 1980's and one in which techniques of knowledge elicitation have been taken into conventional software requirements capture and validation. The techniques of knowledge elicitation are still required when using a development method such as KADS, extending existing ontologies such as Cyc, or developing intelligent communication system such as MMI^2. There are many existing textbooks which list knowledge elicitation techniques and which provide best practise guidance to those who intend to use them (e.g. Diaper, 1989).

This paper has tried to identify and describe areas of knowledge engineering where there is active, important current research, and which appear to be able to be combined together to extend a discipline which has started to produce reliably engineered products, and overcome the excessive publicity and exaggerated promises which the scientific discipline in which it is grounded (Artificial Intelligence) repeatedly attracts.

Although the era of exaggerated expectations of expert systems has been passed. Although knowledge engineering has begun to conform to engineering principles by adopting methodologies. Although knowledge representations have become formalised in logic. Although frame systems have been adopted as object oriented databases. Although knowledge acquisition techniques have been adopted as conventional requirements capture techniques. Although the awareness for the enterprise problem rather than the computer solution has been adopted into mainstream informatics. Although the sensitivity to the user interface has become standardised in business computing. Although semantic typing and constraints have been adopted into deductive database technology. The fundamental distinction between knowledge engineering and mainstream computer science remains. The heritage of Wittgenstein's conflict with Russelian mathematical philosophy remains. The insight from psychology which was the foundation of artificial intelligence persists in knowledge engineering today. Reasoning over large amounts of data which results in interesting results will be incomplete. Algorithms are not sufficient and heuristic reasoning is required. Controlled reasoning and segmented knowledge bases may be required, but the inference process will not be universal, it must be tailored for a task; tailored for a purpose.

6. REFERENCES

[1] Binot, J-L., Falzon, P., Perez, R., Peroche, B., Sheehy, N., Rouault, J. and Wilson, M.D.: Architecture of a multimodal dialogue interface for knowledge-based systems. Proceedings of the Esprit '90 Conference, pp 412-433. Kluwer Academic Publishers: Dordrecht, 1990.

[2] Brachman, R.J. and Levesque, H.J.: The tractability of subsumption in frame based description languages. In Proceedings of the Third National Conference on Artificial Intelligence, 34-37. Menlo Park, Calif.: AAAI, 1984.

[3] Chandrasekaran, B.: Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert System Design. *IEEE Expert* 1(3): 23-30, 1986.

[4] Chappel, H., Wilson, M. and Cahour, B.: Engineering User Models to Enhance Multi-Modal Dialogue. In J.A. Larson and C.A. Unger (Eds.) Engineering For Human-Computer Interaction. Elsevier Science Publishers B.V. (North-Holland): Amsterdam, pp 297-315, 1992.

[5] Chappel, H. and Wilson, M.D.: Knowledge-Based Design of Graphical Responses. In Proceedings of the ACM International Workshop on Intelligent User Interfaces, pp 29-36. ACM Press: New York, 1993.

[6] Clancey, W.J.: Heuristic Classification, *Artificial Intelligence*, 27, 289-350, 1985.

[7] D. Diaper: *Knowledge Elicitation: principles, techniques and applications.* Chichester: Ellis Horwood, 1989.

[8] Guha, R.V. and Lenat, D.B.: Cyc: a midterm report, *AI Magazine*. 11 (3), 32-59, 1990.

[9] Guha, R.V. and Lenat, D.B.: Re: CycLing paper reviews, *Artificial Intelligence* 61(1) 149-174, 1993.

[10] Hickman, F.R., Killin, J.L., Land, L., Mulhall, T., Porter, D. & Taylor, R.M.: *Analysis for Knowledge-Based Systems: a practical guide to the KADS methodology.* Ellis-Horwood: Chichester, 1989.

[11] Lenat, D.B., Prakash, M., and Shepard, M.: Cyc: using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine,* 6, 65-85, 1986.

[12] Lenat, D.B. and Guha, R.V.: *Building Large knowledge Based Systems,* Addison-Wesley: Reading, MA, 1990

[13] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., Swartout, W.R.: Enabling Technology for Knowledge Sharing. *AI Magazine*. 12 (3), 36-56, 1991.

[14] Quine, W.V.: Natural Kinds. In Ontological Relativity and Other Essays (chapter 5). New York: Columbia University Press, 1969.

[15] Schreiber, G., Wielings, B. & Breuker, J.: *KADS: A principled Approach to Knowledge Based Systems Development.* Academic Press: London, 1993.

[16] Steels, L.: Components of Expertise. *AI Magazine* 11(2) 29-49, 1990.

[17] Tansley, D.S.W. & Hayball, C.C.: *Knowledge Based Systems Analysis and Design: A KADS Developer's Handbook.* Prentice-Hall: London, 1993.

[18] Touche Ross: *Knowledge Based Systems: Survey of UK Applications.* Department of Trade and Industry: UK, 1992.

[19] Wilson, M.D., Duce, D.A. and Simpson, D.: Life cycles in Software and Knowledge Engineering: A comparative review. *Knowledge Engineering Review,* **3** (4), 189-204, 1989.

[20] Wilson, M.D.:Task Models for Knowledge Elicitation. In D. Diaper (Ed.) *Knowledge Elicitation: principles, techniques and applications*, 197-220. Chichester: Ellis Horwood: 1989.

[21] Wilson, M.D. and Conway, A.: Enhanced Interaction Styles for User Interfaces, *IEEE Computer Graphics and Applications*, **11**, 79-90, March 1991.