University of Southampton

Logic Programming Based Information Management Tools for Hypermedia Systems

by

Muhammad Anwar-ur-Rehman Pasha

A thesis submitted for the degree of
Doctor of Philosophy

in the
Faculty of Engineering and Applied Science
Department of Electronics and Computer Science

August, 1995

To my creator, **Allah** *the most merciful and gracious,* who created me among the known intelligent creatures of the known universe.

University of Southampton

ABSTRACT

Faculty of Engineering and Applied Science

Department of Electronics and Computer Science
Doctor of Philosophy

Logic Programming Based Information Management Tools for Hypermedia Systems

by Muhammad Anwar-ur-Rehman Pasha

The availability of a large amount of on-line information has created a pressing problem of its maintenance and usability. This thesis presents an open architecture of a general framework, knowledge based hypermedia (*kbh*), in which a hypermedia system and a knowledge based system (kbs) are loosely coupled together. By adopting such a heterogeneous framework and an open architecture approach we aim to attack two problems. Firstly, to overcome a limitation of logic programming by providing loose coupling to conventional language applications which allows the power of logic programming techniques to be exploited for kbs development without undue consideration of interface questions. Secondly, by coupling a kbs to a hypermedia system, to provide domain independent kbs tools to provide additional facilities for managing the information stored in the hypermedia system.

This thesis presents a range of logic programming based kbs tools for managing information in hypermedia. The tools allow non-typical styles of presenting information in hypermedia systems and make the available information more applicable and more understandable. The kbs tools concern ways to assimilate and retrieve documents in hypermedia systems. A semi-automatic method to assimilate new documents is proposed. The method is rather general, allowing the documents to be labelled with a proof tree generated in an interactive session with the kbs. The method of labelling then allows retrieval of documents according to a natural generalisation ordering based on this label and obviates the need to explicitly add new link information when new documents are added.

A key benefit of *kbh* is that it offers a better environment for understanding on-line information than kbs or hypermedia individually. Particularly, the newly developed tools for information management offer a basis for solving disorientation, cognitive overhead and other navigational problems of hypermedia systems. These tools also provide active document features and an intelligent database retrieval environment which enhance the usability of hypermedia systems. In the reverse direction the presence of hypermedia increases the behavioural expressiveness of a kbs. The applicability of *kbh* and the newly developed tools are demonstrated in a legal domain.

Acknowledgements

No book is better than a good supervisor. I wish, I could have proper words to acknowledge your kind supervision, effective guidance, and moral support which make it possible to reach this stage. I want to appreciate the way you revealed the secret of logic programming to me. You always gave me right advice at the right time not only about this research but also about my personal problems. Being an overseas student, I faced many problems during my stay in the UK but you always help me to tackle them properly. I want to pay my special thanks for your comments on the draft of the thesis and for your help in improving it. I know these word are not enough, however, I will simply say "Paul, you are a nice, kind, friendly, and supporting supervisor".

I gratefully acknowledge the support of Multimedia Group of University of Southampton for permitting me to use their system. I am particularly grateful to Professor Dr. Wendy Hall for supervising me in the absence of my supervisor Dr. Paul Soper and for fruitful discussions during my research. I am grateful to Professor Dr. M. B. Sukhara[1] whose training about conducting theoretical research helped me a lot during my study. I should like to thank Ian Heath and Gary Hill for their moral and technical support during the development of a new filter. At the same time I should also like to thank LPA and specially Mr. Brian Steel for providing technical support about LPA Prolog. I want to say special thanks to Andy King, Geeta Abeysinghe, Syed Sahlah, and Bob Kemp for helping me learn Prolog programming.

I am grateful to my parent university - Islamia University, Bahawalpur, Pakistan - and government of Pakistan for providing financial support to conduct this study. I am also thankful to the chairman, Department of Electronics and Computer Science, University of Southampton for providing good research facilities within the department. I am also grateful to my friends here for their constant support and encouragement during this long period.

Above all, I give my heartfelt thanks to Shaheen, my wife. Over the years, she not only endured many days and nights as a research-widow but also sacrificed her

[1]Professor Dr. Muhammad Bellal Sukhara was my teacher, colleague and then became the Vice Chancellor of Islamia University Bahawalpur, Pakistan. Before starting this research, we had published many joint articles in international journals.

invaluable study time for me. By the end of this study she also completed her Ph.D in Special Education. Not only during my research, she always gave me every support and encouragement. All my successes are due to her support and encouragement.

While I am saying many thanks to every body, I want to say my heartfelt excuse to my children Muhammad Abdul_Rehman Pasha and Muhammad Shakaib Arsalan Pasha. Sorry, sons I could not spend time with you which you really deserve. I intend to repay this debt in the rest of my whole life. I am very thankful for your co-operation. You are the best children in this whole world.

Lastly, I want to speak about two things; firstly I want to pay my respects to my parents, family members and friends who prayed for my success. Secondly, I want to mention the name of a person whose research style and work inspired me the most. He is Marek Sergot; A well known researcher from logic programming group of Imperial College. I wish in any part of my life I could do some collaborative work with him. I am thankful to Marek Sergot for providing related literature about this research.

Contents

List of Figures

1 Introduction

This thesis concerns new proposals for the management of on-line information. It therefore addresses a pressing need because of the rapid increase in both the quantity and variety of on-line information. In this introductory chapter we start by introducing our proposal and setting it in the context of existing approaches. We next discuss, in Section 1.2, the legal domain which manifests many information management problems in a particularly rich form. In this section we justify and present some of the reasons why we have chosen this domain as a test-bed for our proposals. In Section 1.3 to 1.5 we set out the objectives of the thesis more precisely, outline our main contributions and list previous published work. Finally we outline the organisation of the remainder of the thesis.

1.1 Motivation

The first industrial revolution made it possible to deliver more blue collar work with fewer workers by substituting machine power for human muscle. The second industrial revolution made it possible to deliver more white-collar work with fewer workers by substituting computers for some brain functions [Hayes-Roth & Jacobstein 1994]. Currently, society is in a transitional phase; switching over from a post-industrial phase to an information revolution. A rapid growth of information is the key outcome of this transition. Consumer expectations based on advances in both computer and multi-media technology are changing the nature and the use of on-line information. On the one hand users are no longer only interested in finding information, but rather they want to use it for effective problem solving purposes [Fischer *et al.* 1989], [Bench-Capon *et al.* 1991], [Soper & Bench-Capon 1994]. On the other hand, it is no longer only text based which are available but also other forms such as bit images, sound, graphics and so on. The vigorous growth of on-line information and the changing nature of its use have created a pressing problem of its management; in the literature this is sometimes called the information crisis [Saarenpaa 1991].

On-line information management is a twofold problem. Firstly, how to store huge amounts of information so that any required piece of information can be retrieved easily and efficiently. Secondly, what kind of tools should be designed to allow users to retrieve only, or at least rather precisely, the required information and then to use it for practical purposes [Soper & Pasha 1994b]. A central motivation for this thesis is to address the second problem by combining the strengths of different information management technologies. For this we select two contrasting technologies: hypermedia whose strength is in handling unstructured information; and kbs whose strength is in handling structured information. More particularly we have focussed on developing kbs support for the rapidly changing hypermedia technology. Furthermore, we have based our approach to kbs development on logic programming (LP) because this paradigm most directly supports the reasoning needed for kbs. We now briefly review the basic categories of information management technology so as to put our proposal in context.

Currently, many information management systems are in operation in different domains. According to their functionalities, Minch [Minch 1989] has categorised them into five different groups: information retrieval systems (IRS), hypermedia systems, knowledge based systems (kbs), database management systems (DBMS), decision support systems (DSS). In this introduction we categorise information management systems even more simply: information retrieval; hypermedia; and knowledge based systems. Our point of view is that conceptually kbs subsumes some aspects of database management systems and that the term decision support systems usually refers to some kind of integration of the basic technologies.

- **Information Retrieval Systems**

The tremendous growth of on-line information and the advances in computer technology, which makes computers fast, cheap, reliable and easy to use, have propelled IRS into popularity despite their limitations. IRS store/retrieve data items written in unstructured formats in/from a database. For retrieval, users submit a query consisting of terms which interest them. This may be a single term or many terms conjoined with the system's logical connectors. Using some kind of search technique, such as full text scanning, signature file scanning, inversion file scanning and so on, these systems retrieve data items by matching the supplied terms with the contents of the stored data items. The systems then allow users to see the full contents of the retrieved data items [Salton & McGill 1983], [Frakes & Baeza-Yates 1992], [Faloutsos & Oard 1995].

The notion of information retrieval described above is based on Boolean search and could be called 'conventional' information retrieval. Its primary characteristic is that it is based on a search for terms and that these are usually words. Irrelevant retrieval and the query construction difficulty are the two commonly cited problems of conventional IRS [Blair & Maron 1985], [Bing 1988]. Let us look closely at these problems. Firstly, how do users construct their queries? Initially, users conceive the problem in their mind and consider all possible aspects of it. They extract aspects which they feel are the most important. Next, they select some suitable terms/words from their every day vocabulary which could express these extracted aspects effectively. Finally, they construct a request by filtering out the unnecessary terms/words. Frequently, users do not cope with this process properly, and so nor do they construct an effective query. Even when users construct a query, they are not sure that what they will get is what they want. Thus the query may cause irrelevant retrieval [Bing 1988], [Schild 1989].

Some other problems are related to the presence of synonyms and spelling mistakes in a query. An idea can be expressed in many ways in natural language and therefore a searcher and an author may use different terms to characterise the same concept [Blair & Maron 1985]. In natural language there are many words which look identical but they have different meaning depending on the context of their usage (homonyms). Many systems try to overcome such problems by using a thesaurus for expanding user requests. The problem with a traditional thesaurus, however, is that it is not context sensitive. The meaning of words mostly depends on their contexts. So the usage of traditional thesaurus is not an ideal solution. On a more basic level, spelling mistakes present in a request may cause an empty retrieval set. So the formulation of a request is acknowledged as being difficult and error-prone [Blair & Maron 1985], [Bing 1987], [Bing 1988].

Conceptual retrieval, to which this thesis strongly relates, is a very different notion in information retrieval. In conceptual retrieval data items are retrieved conceptually so as to minimise the chance of irrelevant retrieval. These systems first confirm the context of the term, where it has been used, and then retrieve only those data items which have similar information [Schild 1989]. Conceptual retrieval approaches [Hafner 1987], [Dick 1987], [Bing 1987], [Wildemast & de Mulder 1992], [Mulder *et al.* 1993], [Mulder & Combrink-Kuiters 1996] can offer a solution to the problem of irrelevant retrieval, however, inspite of a lengthy research effort they are in very early stages. As yet there is no practical or commercial product available. At this stage, it is quite difficult to declare that the systems developed using conceptual approaches will give a satisfactory result. Bing had two points to make about this. (i) About his own idea: "Whether the idea of a norm based thesaurus is viable, is still to be seen. It has interesting possibilities, but whether it will be successful for users will depend on whether the normative structure may be presented in a form which makes it readily and easily understandable by lawyers" [Bing 1988, p. 129]. (ii) About conceptual retrieval as a whole; " It is an open question whether a successfully implemented method for 'conceptual searching' really will increase the performance compared to traditional text retrieval systems. In principle this is an empirical question, which only may be answered if a method is available for controlled, comparative experiments" [Bing 1987, p. 44].

While we agree with Bing, that the value of conceptual retrieval must be judged empirically, we also believe that research into methods of conceptual retrieval is central to effective information management.

- **Hypermedia Systems**

Hypermedia systems are a relatively new technology in the information management community. These systems handle various kinds of on-line information. Some examples are text, bit map, video and sound. They offer a non-sequential style of navigation within the body of information and allow users to link relevant pieces of information together so that they can be traversed in later navigational sessions. Hypermedia systems offer a very powerful environment for handling unstructured on-line information, however, disorientation, cognitive overhead, lack of dynamic linking and lack of a reasoning facility are commonly cited problems of these systems. (Detailed discussion about hypermedia systems is given in Chapter 2.)

- **Knowledge Based Systems**

'Expert systems' is an alternative name of these systems. These systems are designed especially for particular tasks which seem to be very complex, time consuming and need a high level of human expertise. They work by exploiting data structures that represent knowledge acquired from human experts and simulate some of the functions of human experts when answering user queries. They have three main components: a user interface, a knowledge base, and an inference engine. Problem solving heuristic information is stored in the knowledge base using various formats, such as Horn clauses, semantic nets and frames. Using this information along with some kind of inference strategy - for example forward chaining or backward chaining - system answers queries about the problem domain. Such systems are in operation in various domains and perform a range of tasks. User-system communication and knowledge engineering are the two main problems for these systems [Susskind 1987b], [Jackson 1990], [Barragan & Barragan 1991], [Hayes-Roth & Jacobstein 1994]. Also these systems do not handle unstructured information so they can not be used as general purpose information management systems.

We have briefly outlined the different information management technologies and some of their key problems. Although information retrieval systems are commonly used for information management the query construction problem and irrelevant retrieval undermine their potential. Also they are unable to handle all the different kinds of available on-line information and are unable to offer a reasoning facility. On the other hand, kbs offer a reasoning facility, but they are unable to handle unstructured on-line information. In this area, hypermedia systems are emerging as the front-runner for on-line information management since they can handle the range of on-line information. Also conventional information retrieval techniques can be used in conjunction with hypermedia systems, for example incorporating string search in hypertext systems [Faloutsos *et al.* 1989], dynamic linking [Gloor 1991], generic linking [Davis *et al.* 1992], using conventional retrieval techniques for link creation [Zhuoxun *et al.* 1992]. But the use of such conventional retrieval techniques can cause irrelevant retrieval since it is not based on conceptual content [Soper & Pasha 1994c]. Lack of structure and reasoning are the two missing elements from hypermedia systems [Littleford 1991]. Combining kbs technology with hypermedia systems can provide these missing elements. It can also allow conceptual retrieval in hypermedia systems [Pasha & Soper 1995]. A limited number of attempts have been made in this direction but more research is required to build effective systems which can fulfil user demands and can handle the main on-line information management problems. Our research is an attempt in this direction, in particular to provide kbs support for hypermedia systems.

1.2 Legal Domain as a Test-Bed

Among others, one particular aim of our research is to design a general purpose information management technique that can be applied in various domains. Selection of a domain, as a test-bed for a general purpose technique, is a most important task which needs serious consideration. After considerable thought we found the legal domain to be suitable as a test-bed for our research. There are many reasons behind this decision and some of them are mentioned below.

Arguably it can be said that the professionals from the legal domain use more on-line information than any other domain's professionals [Gelbart & Smith 1991]. Also legal professionals are interested in the computerisation of a number of different aspects of legal practice, including retrieval of information and practical use of the available information. Typical commercial applications, for example WESTLAW, LEXIS, have concentrated on only one of these aspects. This lack of integration in commercial systems is not peculiar to law, but has been observed to be a general feature of the computerisation of information [Greenleaf *et al.* 1991]. Apart from the need for integration, legal information use has many underling problems - discussed below - which make it difficult to computerise fully.

Legal information usually contains many concepts which are not clearly defined or not defined at all, or have unspoken expectation. These concepts are open texture concepts. Formalisation of open texture concepts is recognised as problematic [Skalak & Rissland 1991], [Poulin *et al.* 1993]. At the same time, legal concepts are also dynamic; their meanings change as they are applied from case to case [Linzer 1988]. Even the logical connectors used in legal rules can be ambiguous, because their scope may not be clearly specified [Allen & Saxon 1987]. These issues seem to pose major constraints on computerising legal information fully. Our proposals make a contribution to provide help with open texture concepts, although this is not the main problem of the thesis.

Case reports are one of the most commonly used sources of information in the legal domain. The information in case reports is commonly used to help interpret open texture concepts. Also they provide precedents for new problems. As Skalak and Rissland [1991] mention, an important routine task of law practitioners is to find suitable cases which can be referred to in court to make their argumentation strong. Ashley and Aleven [1991] argue that during the search for suitable cases law practitioners try to find cases related to a particular issue which has been decided under similar circumstances. In addition to this they also consider some other factors including whether they are arguing for or against the result indicated by legal rules, whether the point at the issue has been decided in one's favour or not in a precedent, and whether there are alternative paths to the desired result. Retrieval of suitable cases according to these aspects is an important issue in computer based legal research and needs special consideration. The proceedings of the AI and Law conferences give a good account of current work being carried out in this direction [AIL 1987] , [AIL 1989], [AIL 1991], [AIL 1993], [AIL 1995].

In this section we have mentioned some of the problems of legal information and its consumers. Growing concern in legal research shows the importance of these problems and their solutions. Many attempts have been made to resolve them and many more are required to reach an ideal solution. We choose the legal domain with an aim to attack these problems and to find acceptable solutions for them. Although, we discuss the problems in the context of the legal domain, they can occur in other domains. We believe a solution to them will not be limited to the legal domain but will also be applicable in other domains.

1.3 Objectives

The main aim of this research is to develop new techniques for handling on-line information by merging existing information management technologies. More specifically the thesis is that merging two particular technologies, hypermedia and kbs, to build a composite system can: (i) Solve significant problems of information management; (ii) Provide an effective environment for information understanding; (iii) Provide an effective research platform for implementing new ideas.

The main component objectives of this thesis are given below:

- Review state of the art information management systems; particularly, hypermedia systems, focusing on the legal domain.
- Propose an architecture, called *kbh*, to merge hypermedia and knowledge based technologies.
- Implement the proposed *kbh* architecture including a bi-directional communicational channel (BCC).
- In the context of *kbh* propose and design a logic programming (LP) based kbs shell and a conceptual retrieval information management tool (called PCR).
- Show how PCR in the *kbh* framework can be used to provide additional facilities in hypermedia systems including *active document features*[1](sending queries to kb to see the practicality of available information) and *enhanced database retrieval* (structured retrieval).
- Develop and evaluate a prototype *kbh* system and tools for an application in the legal domain.

1.4 Contributions

Management of on-line information and the development of a system which can fulfil current information consumers' requirements were the two problems addressed in this thesis. We do not claim that we have solved both problems fully; however the outcome of this research could streamline future research in a positive direction which potentially could offer solutions to the current information crises. Some of the significant contributions of this thesis are highlighted below.

- We have developed a composite system, knowledge based hypermedia (*kbh*), which provides a loosely coupled environment between a kbs and a hypermedia system. The loosely coupled approach and the logic programming techniques used for the development of the kbs offer many advantages.

[1]The term *active document features* was used by Bench-Capon and Soper in [Bench-Capon *et al.* 1991].

  - On the hypermedia side the presence of the kbs enhances the functionalities of the hypermedia system and addresses some of their fundamental navigational and retrieval problems. For example, the presence of the kbs inference mechanism allows conceptual retrieval and reasoning in hypermedia systems, making browsing more meaningful and more effective.
  - On the kbs side the presence of the hypermedia system addresses some of the fundamental problems of user-system interaction associated with current kbs. The hypermedia provides access to coherent texts concerning the application domain and, if it is multimedia, also to audio/visual sources. This can enhance the presentation of expertise and can increase the usability of kbs.
  - The *kbh* framework offers a balanced intelligence distribution between user and system. In the kbs intelligence lies primarily with the system; users can submit queries and can get answers but they do not know about the details of the primary sources. By contrast, in hypermedia, the intelligence lies largely with the users, who explicitly guide the system for information navigation. In the *kbh* framework the system and user work together for information navigation.

- We have developed an LP based, semi-automatic technique, **P**roof tree based **C**onceptual **R**etrieval (PCR), for conceptual retrieval which allows conceptual dynamic linking in hypermedia systems. PCR tags conceptual information about linked documents within the body of links. It offers many advantages and changes the basic nature of typical hypermedia links from simple GOTO's to conceptually enriched links. This makes navigation more conceptual and more effective.

- Providing a facility for conventional database retrieval would not normally be claimed as a big achievement, but combining such structured retrieval with PCR brings together two complementary retrieval ideas and offers a twofold advantage. On the one hand, it allows structured information retrieval in hypermedia. On the other hand, it appears to produce a considerably more effective retrieval compared to a typical database system alone. We named this combined retrieval *enhanced database retrieval.*

- The *kbh* framework is a hybrid model of kbs and hypermedia system. It allows users to submit queries to the kbs from hypermedia and to see the results through the kbs shell user interface. In this way the *kbh* framework offers a facility for hypermedia users to see the practicality of the available information in the system. This particular facility we call *active document features.*

- The prototype *kbh* is a hybrid of two important programming paradigms, imperative and declarative, and their corresponding languages (hypermedia uses 'C', kbs uses Prolog and bi-directional communication channel uses combination of both). The whole is developed using a modular based approach. Our idea of modular programming is to make a logical separation between parts of the system which perform their function separately from each other. This provides an easily upgradeable, open framework in which end-users can implement new ideas by exploiting the functionalities of imperative and declarative languages side by side. It also makes the *kbh* framework a suitable platform for research activities in various domains, such as industry, law, education and medicine.

1.5 Previous Publications

Some of the material of this thesis has been published previously in more or less detail. A list of published material is given below. The material covered splits roughly between the *kbh* framework, conceptual retrieval and applications as suggested in the titles.

- **Knowledge Based Hypermedia**
  Paul Soper, Wendy Hall, Muhammad A. Pasha and Ian Heath
  Proceedings of the Workshop on Logic Programming Support Environments,
  University of Edinburgh, UK. September 1993.

- **KBH: A Framework For Managing Legal Information**
  Paul J. Soper and Mohammed A. Pasha
  (Runner-up award for the best presented paper)

Conference Proceedings of the Information Technology and its Applications, Leicester, UK. April 1994.

- **Logic Programming Techniques for Handling Navigational Problems of Hypermedia Systems**
  Paul J. Soper and M. A. Pasha
  Proceedings of the 10th Logic Programming Workshop, University of Zurich, Switzerland. October, 1994.

- **Logic Programming Environment for Hypermedia Information Management**
  Paul Soper and M. A. Pasha
  Conference Proceedings of the 6th International Conference on Artificial Intelligence and Expert Systems Applications (EXPERSYS-94), Houston, Texas(USA). December, 1994.

- **Knowledge Based Hypermedia for Information Management**
  Paul Soper and Muhammad Pasha
  Conference Proceedings of The 14th annual conference of the British Computer Society Specialist Group on Expert Systems(ES-94), Cambridge, UK. December, 1994.

- **Integrating a Knowledge Base with an Open Hypermedia System and its Application in an Industrial Environment**
  I. Heath, W. Hall, R. Crowder, M. A. Pasha, P. Soper
  Proceedings of the Workshop on Intelligent Hypertext, Maryland, USA. December, 1994.

- **Conceptualising Hypertext Links Using Expert System Techniques**
  Muhammad A. Pasha and Paul Soper
  Conference Proceedings of the 7th International Conference on Artificial Intelligence and Expert Systems Applications (EXPERSYS-95), San Francisco, California (USA). November, 1995.

1.6 Thesis Structure

- **Chapter 2** discusses Hypermedia systems. Firstly, we briefly describe typical hypermedia systems, their basic concepts and a simple architectural model adopted in this thesis. Later, after discussing the advantages of hypermedia systems we discuss some well cited problems of hypermedia systems. Taking a practical example from the legal domain, we illustrate some limitations which current hypermedia systems are unable to overcome without including additional functionalities. This example concerns the retrieval of legal case reports. It will provide a key illustration of how our new information management proposals overcome these problems.

- **Chapter 3** starts by describing the various new research trends which combine the strengths of conventional information management technologies for on-line information management. Combining kbs and hypermedia is one of them. Using this approach, in this chapter, we propose a general purpose framework, knowledge based hypermedia (*kbh*), in which a logic programming (LP) based kbs and a hypermedia system are loosely coupled into a composite system. After discussing the architectural detail of *kbh*, we mention some of the main advantages of our framework, such as a relatively domain independent environment and provision of a reasoning facility for hypermedia systems. The reasoning facility, on the one hand, offers active document features (sending queries to the kb); and on the other hand provides a basis for conceptual retrieval. Lastly, the chapter gives a comparison of the *kbh* framework with related work and draws some conclusions.

- In **Chapter 4** we propose a semi-automatic conceptual retrieval technique - **P**roof tree based **C**onceptual **R**etrieval (PCR) - which allows conceptual retrieval in hypermedia systems by making hypermedia links conceptually enriched. This provides a basis for conceptual dynamic linking in hypermedia systems. PCR is an LP based technique. After giving a general introduction of logic programming, the chapter explains our proposal to generalise, in a schematic way, the idea of query-the-user (QTU) [Sergot 1982] to include three categories of knowledge: knowledge formalised in the kb (**kb_system**); knowledge in the user's head (**kb_user**); and knowledge in hypermedia documents (**kb_doc**). Interaction between the user and the other knowledge sources expands the boundary of **kb_system** without changing the totality of knowledge. Later, the chapter introduces our specific proposal for representing conceptual knowledge in documents, proof tree labels. The essential idea is that an appropriate query, related to the document's contents, is posed to the kbs. The execution trace generated in the interactive processing of the query is then used as a label for the document. We have adopted the term 'proof tree label' for such a trace.

The overall purpose of PCR is to find relevant documents, and the basic idea is to do this by matching proof tree labels. We have designed a proof tree label matching algorithm: by matching predicate names of the proof tree label nodes and using a weighing scheme, the algorithm generates a *matching weight* (MW) as a measure of similarity between two proof tree labels. The *percentage matching weight* (PMW) is used to give a clue to users about the level of relevancy between retrieved documents and their request. In this chapter, we justify our approach of matching predicate names for matching proof tree labels and the weighing scheme used to generate the matching weight. The proof tree label matching algorithm is also given in this chapter.

- **Chapter** 5 describes how PCR is applied to hypermedia systems. We start this chapter discussing two problems related to the use of PCR in hypermedia systems, how to inform users which definitions are available in the kb and how to inform users which concepts from a document are linked. We explain how we handle these two problems in our prototype system. Probably the most important contribution of PCR to hypermedia is to offer conceptual dynamic linking. This is done by storing conceptual information about destination anchors within the bodies of links so that during the linking process the system will link those documents which are conceptually similar/relevant to a user's request. In this chapter, we demonstrate how PCR offers conceptual dynamic linking in hypermedia systems. We also describe how our system provides *active document features* and *enhanced database retrieval* which is a combination of conventional database retrieval and PCR. The enhanced database retrieval appears to produce a considerably more effective retrieval environment. Lastly, we expand on the advantages of PCR for solving hypermedia problems, giving a concluding discussion and mentioning some issues which will inform the direction of future research.

- **Chapter 6:** The *kbh* framework combines two major technologies, hypermedia and kbs, but the communication channel (BCC) is also an essential part. It provides a message passing protocol through which these components can send and receive messages on a run-time basis. In this chapter we discuss software selection for the *kbh* framework; the hypermedia system and the kbs development tools. After discussing the important features and necessary background of the selected software, we describe the implementation and functioning details of BCC. We also mention some advantages of bi-directional versus uni-directional communication. Finally we argue that other AI technologies can also be linked with hypermedia in a similar way. If today's software industry adopts a similar approach, and provides in-built facilities in applications for bi-directional communication with other applications, it will be easier to build such composite systems.

- **Chapter 7** discusses the development of our prototype kbs shell for the *kbh* framework. Its development is heavily influenced by the expanded context of *kbh*. The shell is not a single entity in *kbh*; rather it is the inference engine together with a variety of information management tools and a user interface. Our idea is that the shell is extendible to include additional useful tools. In developing the shell we adopted a modular style, mainly because this was useful for handling complexity and existing code. Also, it makes the shell open-ended allowing end-users to add new modules. The shell has six main modules: Message Analyser, Labeller, Matcher, User-Interface and System Library. All modules are developed using a meta-level programming approach. The Message Analyser is the central module. It is responsible for handling inter-module communication and performing tasks requested through the hypermedia manager. The implementation and functioning details are given in this chapter.

  In this chapter we propose a 'help' facility for users when answering unclear system questions. Our contribution is to display the documents in which the concepts asked about have been explained. This can help users to understand these concepts. In our approach the connection between relevant information and concepts is not hard-wired, rather the system computes it dynamically. We argue that our approach is more flexible than designing simple questions or linking text files with such questions.

- **Chapter 8** describes a case study carried out in the legal domain using our prototype *kbh*. The general motivation of the case study is to check the feasibility and functionalities of the newly developed information management tools and the *kbh* framework. The case study is based on leaflets about British nationality registration provided by the Home Office [Home Office 1991c], [Home Office 1991b] and [Home Office 1991a].

  Through the case study we demonstrate the functionalities of the prototype and try to support our claims made about the *kbh* framework in this thesis. Finally we argue that the *kbh* framework offers a significant improvement over kbs or hypermedia alone. On the one hand, it offers an advisory system like environment, and, on the other hand, it provides an effective environment for information exploration.

- **Chapter 9** presents our conclusion and highlights some areas for future research directions.

2 Hypermedia Systems

2.1 Introduction

Hypermedia systems, relatively a new technology, offer an alternative style of information handling compared to other information management systems. Conventional information retrieval systems deal with unstructured information but expect a structured query from users about their interest. Hence these systems face query language limitations and irrelevant retrieval problems which undermine their efficiency. Typical hypermedia systems do handle unstructured information, but, in contrast, they do not require any formal query. Hence, they do not face query language difficulties or irrelevant retrieval problems.

In the hypermedia community, use of inconsistent terminology is a common occurrence. This is expected in such a dynamic research area and often reflects important underlying differences of opinion. In Section 2.2, so as to be clear about the basic terminology used in this thesis, we briefly describe typical hypermedia systems, their basic concepts (Section 2.2.1) and the simple architectural model adopted in this thesis (Section 2.2.2).

It is important to mention that across the inter net the World Wide Web (WWW) is a commonly used hypermedia system. It is often used to access information stored on different sites. WWW provides various search engines, which like typical information retrieval systems retrieve information using conventional retrieval techniques. In this respect WWW is a good example of combining hypermedia and IRS technologies. Inspite of the importance of the WWW, we have focussed in this thesis on general characteristics of hypermedia systems rather issues specific to the WWW and therefore the discussion in this chapter does not refer explicitly to WWW hypermedia systems. Nevertheless the outcome of our research, which is to explore the potential of kbs technology to support hypermedia systems, is also applicable to the WWW.

Hypermedia systems offer a non-sequential style of navigation through the body of information. They allow associations (links) between relevant pieces of information to be made. During a navigation session, users can traverse the body of information through these links and can retrieve desired information easily. Also, these systems can handle various kinds of on-line information, such as text, graphics, sound, bit images, video and so on. In Section 2.3 we mention some of the advantages of hypermedia systems.

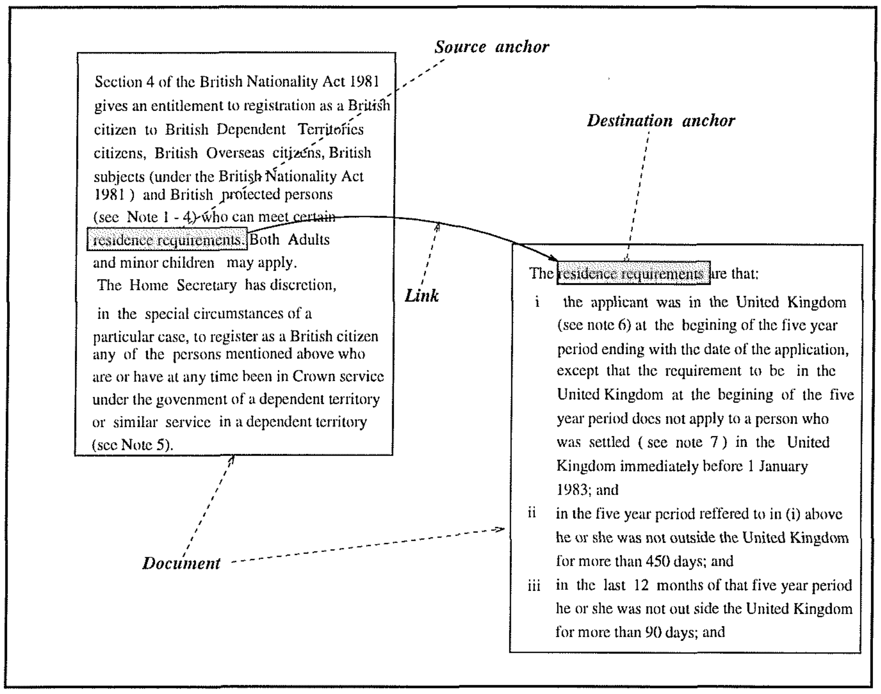
Hypermedia systems offer many attractive features, but they face some serious architectural and navigational problems. In Section 2.4 we discuss the problems of hypermedia systems taking into account currently proposed approaches. In Section 2.5, taking a practical example from the legal domain, we illustrate some limitations which current hypermedia systems are unable to overcome without including additional functionalities. This example concerns the retrieval of legal case reports. It will provide a key illustration of how our new information management proposal overcomes these problems. Section 2.6 gives our conclusion.

2.2 Hypermedia Systems

The term hypertext was used first by Theodore Nelson to describe the idea of a non-sequential style of reading/writing [Nelson 1967]. But this idea has been around since 1945 when Vannever Bush, Scientific advisor to Roosevelt, put it forward. In his article [Bush 1945], Bush proposed a system that would allow the users to store books, records, communications and so on and to retrieve any portion of information quickly based on associated cross-references. It is the recent development of computer technology, however, that makes it possible to implement this idea and now the development of hypertext systems is a valuable contribution to information technology [Fountain *et al.* 1990].

The aim of hypertext systems is very simple and practical: to provide an effective and easy to use environment for integrating and managing on-line information. By offering a non-sequential style of navigation - navigate through the contents of information via different paths compare to conventional text that typically allows sequential or hierarchical access - these systems provide a helpful environment for information understanding [Heath 1992].

The availability of multi-media information technology broadens the horizon of these systems. Currently they not only handle textual information but also support other kinds of on-line information; such as bit images, sound, video and graphics. The systems which handle various kind of online information are termed hypermedia systems. The attractive features of hypermedia systems are making them very popular among users who deal with on-line presentation of large amounts of loosely structured information, for example on-line documentation or computer-aided learning [Nielsen 1990b]. The functionalities they offer and their increasing popularity point to their importance as the basis of future information management systems [Marchionini & Shneiderman 1988].



Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories citizens, British Overseas citizens, British subjects (under the British Nationality Act 1981) and British protected persons (see Note 1 - 4) who can meet certain residence requirements. Both Adults and minor children may apply.
The Home Secretary has discretion, in the special circumstances of a particular case, to register as a British citizen any of the persons mentioned above who are or have at any time been in Crown service under the govenment of a dependent territory or similar service in a dependent territory (see Note 5).

The residence requirements are that:

i the applicant was in the United Kingdom (see note 6) at the begining of the five year period ending with the date of the application, except that the requirement to be in the United Kingdom at the begining of the five year period does not apply to a person who was settled (see note 7) in the United Kingdom immediately before 1 January 1983; and

ii in the five year period reffered to in (i) above he or she was not outside the United Kingdom for more than 450 days; and

iii in the last 12 months of that five year period he or she was not out side the United Kingdom for more than 90 days; and

Figure 2.1: Basic concepts of hypermedia systems

Fielded hypermedia systems offer various kinds of tools for readers and writers[1]. Using these tools, users can navigate through the body of information from an introductory level to higher level or can make association between relevant pieces of information for future navigation. Some of the basic concepts of hypermedia systems

[1] We will use the term users for both readers and writers. Some authors, however, do make distinction between readers and writers.

are briefly discussed in the next section; for a detailed account readers could consult [Conklin 1987], [Jonassen 1989], [Nielsen 1990a], [McKnight *et al.* 1991].

2.2.1 Basic Concepts

In this section we explain some of the basic concepts of hypermedia systems as shown in Figure 2.1. We want to mention that, due to the popularity of hypermedia systems, in literature the term hypermedia is commonly used in place of hypertext; this practice is being followed in this thesis. The term hypertext will explicitly be used where the system only handles textual information.

- ***Document:*** An elementary component of hypermedia systems is a *document*; some systems use the term *card*. A *document* contains a chunk of information which may be in any form like text, graphic, audio signal, video signal. The size of a document depends on the architecture of the system as some systems impose a limitation on its size.

- ***Anchor:*** The second important concept is an *anchor*. An anchor is a selected area of a document which shows users interest. There is no limit on its size; it may be a word, a phrase, a sentence, a paragraph, or a full document. Users can select an anchor called *source anchor* and can connect it with another anchor called *destination anchor*. Both anchors may be selected either from the same document or from different documents. A source anchor must be connected with a destination anchor. This type of connection is known as *one-to-one* mapping. However, many systems allow more than one destination anchor giving rise to *one-to-many* mappings or *many-to-many* mappings.

- ***Link:*** The next important concept is *link*. The association between source anchor and destination anchor is known as a link. Different systems store link information in different ways. Some tag this information within the main body of the original documents, as hidden characters, whereas other systems keep it in a separate place called a Link Base. In this thesis we call the link information *body of the link*.

  Links are just like hard wired connections between associated anchors and provide navigational paths. These paths can be traversed by activating the source anchor, usually done by a pointing device like a mouse, and following the link to the destination anchor. In this way the information can be traversed in a non-sequential manner. Different systems offer different kinds of links which provide different functionalities. The functionalities provided by links implicitly define the characteristics of the systems. Readers can get detailed information about the links offered from a system's reference manual.

Links are the main strength of hypermedia systems. These links my be inter-document or intra-document. According to [Tomek & Murer 1992], links can be categorised into two main classes: *cold* or *passive*; and *hot* or *active*. In passive links the destination anchor is always a piece of information. In active links it may be a piece of code which is responsible for performing some action on a piece of information. The active link feature is not available in all systems. It is available in those systems which support script or action language such as HyperCard [Apple Computers 1987] and KMS [Akscyn *et al.* 1988].

2.2.2 A Simple Model of a Hypermedia System

In this section we set down a simple architecture of a typical hypermedia system. For the purpose of this thesis we can accept Conklin's definition [Conklin 1987] of what constitutes a hypermedia system. According to his description a hypermedia system has three essential elements:

- ***Database:*** An underlying system to manage and to store the hyper-documents (collection of text, digitised speech, audio recording, graphics, animation). The database can be thought of as a network of textual and graphical data items (documents).
- ***A Windowing System:*** A user interface that provides a means of interaction with the system and tools for displaying required information.
- ***Links:*** The ability to create links between anchors, inter-document or intra-document, within the system. These links may be simply GOTO's, or may contain a piece of code responsible of performing some action.

Many of the fielded hypermedia systems satisfy these criteria. The additional functionalities which they provide distinguish them one from another.

2.3 Advantages of Hypermedia Systems

Among others, two main advantages of hypermedia systems are non-sequential style of navigation and handling information stored on multi-media devices. Hypermedia systems offer various navigational tools. Using these tools, users can create links between relevant pieces of information. Users can build their own links or simply annotate someone else's document. Because source and destination anchors are treated equally it is easy to go forwards and backwards among the anchors. Also, both hierarchical and non-hierarchical organisation can be imposed on unstructured information. Providing the table of contents, a local view, a global view, or mixing local and global views, browsing through the whole body of information can be made easy.

Some systems embed information about links within the contents of the documents which limits the use of information stored in CD-ROM like devices. But, many systems now keep link information separate from documents and transfer it indirectly. This technique makes the system more flexible and allows linking of information stored on read-only memory devices.

We have mentioned some advantages of hypermedia systems, but to date, hypermedia technology has been applied only to a limited extent in domains like medicine, industry and marketing. The potential of this technology, however, is vast and needs more exploration. Already, however, the increasing popularity of hypermedia systems, especially on the WWW, suggests they are the front-runner for future information management systems.

2.4 Problems of Hypermedia Systems and Current Proposals

In spite of the above mentioned advantages of hypermedia systems, there are still many problems which remain unresolved and need special consideration. Two well-cited problems are cognitive overhead and disorientation [Conklin 1987], [Halasz 1988], [Jonassen 1989], [Halasz 1991], [Nielsen 1990b], [Fauth 1995]. Some of the problems related to hypermedia links are briefly discussed below.

The main function of links is to allow users to establish permanent association between pieces of information which they think are relevant to each other, so that these relevancy associations can be traversed during later navigational sessions. It has been observed that a rich network of links introduces the problem of getting lost. One study showed that 56% of users complained about this [Nielsen & Lyngak 1989]. A number of navigational aids have been suggested in [Nielsen 1990b] such as: facilities for backtracking, graphical display of relative position of the current anchor through global and local overview diagrams, creation of prominent landmarks, use of time stamps and foot prints, and history of traversal of documents. But this issue has not been resolved yet and remains a major problem for hypermedia systems [Mahapatra & Courtney 1992].

Some of the other issues related to links, raised in [Tomek & Murer 1992] which need special consideration, are mentioned below:

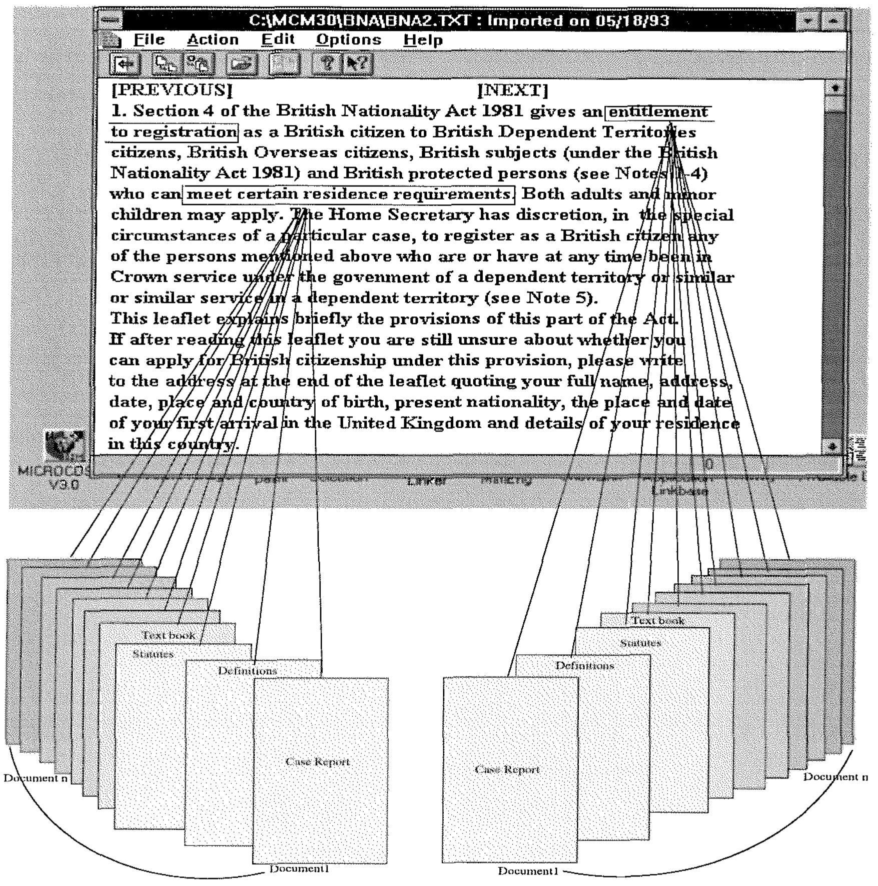
- What type of behaviour should these links have? Whether they should be *cold* providing the same information every time, or be *hot*, their destinations being automatically calculated according to user responses.
- How should these links be created? Manually or dynamically?
- Should the links be simple GOTO's or should they have some conceptual information about the destination anchor?
- What information should the system display about destination anchors if it has *one-to-many* option?
- How should the system help users in selecting one of many links available at a given point?
- What kind of information about destination anchor should the system provide before users decide to activate traversal?

In a typical hypermedia system the link is no more than a simple GOTO [Dam 1988]. So systems can only retrieve those documents which have predefined links. This particular weakness of hypermedia links has been discussed by several authors [Garg 1988], [Bruza & van der Weide 1992], [Hall 1994]. To overcome this weakness, some systems allow dynamic linking which can reduce the overhead of manual linking [Davis *et al.* 1992], [Gloor 1991]. In dynamic linking documents are captured at run time according to the needs of the user. This offers a more flexible way of navigating through the information. Dynamic links are so far only implemented in a few systems [Faloutsos *et al.* 1989], [Gloor 1991], [Davis *et al.* 1992], [Zhuoxun *et al.* 1992]. The approaches taken in these systems, often called generic/computed linking, are usually based on conventional retrieval techniques and therefor introduces the associated problems (e.g. irrelevant retrieval problem); discussed in Section 1.1. So generic/computed linking is only the start on solving the dynamic linking problem.

The heart of the problem is that hypermedia links do not have any conceptual information about the contents of the linked documents, so systems are unable to calculate associations between conceptually relevant documents dynamically. The current literature suggests a growing need for conceptual association between documents [Agosti & Marchetti 1992], [Arents & Bogaerts 1993]. Conceptual association could reduce the chance of disorientation and handle many problems related to hypermedia information retrieval. Layered hypermedia models, as proposed in [Agosti *et al.* 1991], [Bruza & van der Weide 1992], [Bruza 1993] are recent attempts in this direction. The common feature of these approaches is to build different layers showing an index of the concepts at the top of the actual information. The approaches provide partial solutions to the problems mentioned above. But their main weakness is that they do not adequately explain how to choose which layers will be suitable for a particular application. In the next section we illustrate the limitations of typical hypermedia systems and the currently proposed approaches for conceptual linking.

2.5 An Illustration of the Limitations of Hypermedia Systems

In this section we illustrate, through a practical example, that current hypermedia approaches are not well suited for handling legal information, so there is a strong need to provide additional functionalities to handle this kind of information. Figure 2.2 shows a text taken from a leaflet published by the British Home Office for the general public [Home Office 1991c]. Let us suppose, a user wants to link the available information with the concept *entitlement to registration.* The legal domain offers different information sources such as text books, summaries, statutes and case reports. A typical hypermedia system does not allow splitting of the available information



[PREVIOUS] [NEXT]
1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories citizens, British Overseas citizens, British subjects (under the British Nationality Act 1981) and British protected persons (see Notes 1-4) who can meet certain residence requirements. Both adults and minor children may apply. The Home Secretary has discretion, in the special circumstances of a particular case, to register as a British citizen any of the persons mentioned above who are or have at any time been in Crown service under the govenment of a dependent territory or similar or similar service in a dependent territory (see Note 5).
This leaflet explains briefly the provisions of this part of the Act.
If after reading this leaflet you are still unsure about whether you can apply for British citizenship under this provision, please write to the address at the end of the leaflet quoting your full name, address, date, place and country of birth, present nationality, the place and date of your first arrival in the United Kingdom and details of your residence in this country.

Figure 2.2: Linking documents with concepts in a typical hypermedia system

according to its source. In the basic approach users link all available information with the selected concept without making any distinction as shown in Figure 2.2. Linking information from all available sources with a single concept can easily lead to an overwhelming number of links to the concept, even for one type of source such as case reports this can be the case. This makes it quite difficult to find required information during a navigational session.

A solution to this problem using the layered model approach is that the user creates (say) two layers. The top layer shows an index of terms representing different information sources such as text books, case reports and so on. Each indexed term then acts like an anchor which links with the actual documents of the bottom layer[2]. In a navigational session when a user follows a link from a particular anchor, the system

[2]The layer model is similar to a hierarchical structure but some researchers like to use this term for their system. Actually they claim that their models have conceptual linking in the body of information.

first displays the index of the information sources. The user selects the source type required and then the system displays all linked documents of the selected type. But, in practice, the problem is not so simple to solve.

The legal domain supplies hundreds of case reports related to *entitlement to registration*. As one can see from the text of Figure 2.2 the concept *entitlement to registration* covers different groups of people who are eligible to apply. Again, hundreds of case reports will be available related to each group. One can argue that introducing another layer could solve this problem. In the new layer the user builds another index of terms representing different groups of people. During the navigational session a reader selects his/her group of interest and the system displays the documents related to that particular group. But the problem is not solved yet.

It is obvious from the text of the figure that for *entitlement to registration* the client must meet certain residence requirements. These again have many other conditions which need to be satisfied. At the same time case reports cover many other aspects like court decisions, clients' factual situation, cited cases and so on. It is quite difficult to decide how many and what layers are required to link a case report. At the same time introducing a large number of layers will not only make navigation complicated but may also reduce users' interest to explore the information. It will also increase the chances of getting lost. Handling this kind of complex information using layered or hierarchical models is not ideal. As Nielsen comments it is not possible to find out the usability of this kind of layer structure without knowing how many layers will be suitable [Nielsen 1990b].

A second approach to handling this kind of complex information is through the use of script languages, like HyperTalk, provided by hypermedia systems. But typically such languages are not powerful enough to handle the complicated problems which arise. The limitations of these languages are discussed in [Nielsen 1990b].

Another problem is that all case reports related to *entitlement to registration* also contain information about the concept *meet certain residence requirements*. As typical hypermedia systems are unable to create links dynamically between conceptually relevant documents so users explicitly create links between anchor *meet certain residence requirements* and case reports having such information. This certainly puts extra load on users.

In this section we have illustrated a basic problem with current hypermedia systems: that they are unable to give sufficient assistance to users in selecting suitable links from one-to-many or many-to-many mappings. A typical hypermedia system displays very little information about linked documents, for example title or file name. To get an idea about the contents of documents, readers need to traverse them which is a painstaking problem. We have also pointed out that recent proposals such as layered model and scripting fail to solve the problem or become too complex to apply to typical tasks in the legal domain. This shows the strong need of new techniques which can handle such complex problems effectively. In this thesis we attempt to solve similar kinds of complex problems.

2.6 Conclusion

In this chapter we discussed the relatively new technology of hypermedia systems. These systems offers a non-sequential style of navigation which provides a useful environment for information management. Also, they can handle various kinds of on-line information in various media. The functionalities they offer and their increasing popularity point to their importance as the basis of future information management systems. However, these systems are facing some serious navigational and architectural problems, such as disorientation, cognitive overhead and lack of conceptual dynamic linking.

Conceptual linking is emerging as a new topic in hypermedia systems. Conceptual linking can handle hypermedia problems, in principle, but as far as we are aware previously no adequate proposal has been put forward to make this effective in practice. Among others, layered models are a relatively new approach to offer conceptual linking, but we have discussed their limitations. So there is a need to explore new techniques to handle information stored in hypermedia systems.

ElHani says in his paper [ElHani 1992, p. 24]: " Hypermedia is unable to answer or to solve a problem, that a learner makes up himself, unless it was foreseen by the author. Furthermore, hypermedia has no capability either to analyse unexpected responses nor to make decisions in such circumstances. It has also no facility of modelling the learner. All presentations are pre-defined. It means that hypermedia is lacking a reasoning and inference process. ...That is why it should be extended towards expert systems".

We also believe that extending hypermedia towards kbs technology and providing kbs tools in hypermedia will not only provide reasoning facility in hypermedia systems but also fulfil the demands of users to find the desired information with less effort. The presence of kbs tools can also provide additional functionalities such as active document features, conceptual and structural retrieval that make hypermedia systems more effective and more useful. In the next chapter we propose a new composite system, Knowledge Based Hypermedia (*kbh*), that combines the strengths of kbs and hypermedia technologies and offers an effective and powerful environment for information management.

3 Knowledge Based Hypermedia (*kbh*)

3.1 Introduction

Up till now, we have highlighted some of the fundamental problems of information management. The inadequacy of existing techniques has challenged researchers to explore new approaches to on-line information management. The use of AI techniques, particularly kbs, has emerged as a popular approach. Jones in her paper [Jones 1991] argued that conventional information retrieval can be impractical or inappropriate whereas kbs approaches have the potential to provide effective systems. Croft comments [Croft 1993, p. 11]: "In the past few years we have begun to understand, at least conceptually, how traditional information techniques and knowledge-based techniques could be integrated into a single framework ... Knowledge-based techniques require a much larger investment in human effort and have yet to establish clearly their usefulness in the context of retrieval. On the other hand, they hold the promise of a much more effective system, and this promise motivates the growing body of research in the area".

Many studies have been carried out merging kbs and IRS technologies. Some examples are SCALIR [Rose & Belew 1989], I^3R [Croft & Thompson 1987], CanSearch [Pollit 1987], FLEXICON [Gelbart & Smith 1993], a case-based approach to information retrieval [Daniels & Rissland 1995]. Likewise there are many examples of studies which merge kbs and hypermedia technologies. Examples of this approach are SHADOW: a technical support system for software engineering [Schlumberger 1989], an expert system for the domain of labour law [Hamfelt & Barklund 1990], ExperText [Rada & Barlow 1989], HRExpertext [Diaper & Rada 1991], DataLex [Greenleaf *et al.* 1991], Animated Hypertext [Bench-Capon *et al.* 1991], [Soper & Bench-Capon 1994] an Expertext system for building standard [Casson & Stone 1992]. Some other studies carried out in health care are reported in [Rada 1995] and [Timpka *et al.* 1995]. In this thesis, we have selected the merging of kbs and hypermedia for further investigation. Our basic reason is, as we argued in the previous chapter, that hypermedia is a powerful technology for on-line information management which is producing encouraging results. Furthermore it seems to be clear that the technology is being taken-up rapidly and so seems likely to be a front-runner in the future of information management. Apart from this pragmatic motivation, the advantages of kbs techniques in hypermedia have been widely recognised [Marchionini & Shneiderman 1988], [Minch 1989], [Kibby & Mayes 1989], [ElHani 1992], [Soper *et al.* 1993], [Pasha & Soper 1995], [Arents 1995]. Information retrieval systems as a general technology for information management are not so attractive because they can not handle different media. But regardless of this, IRS is an essentially orthogonal technology which can be added in a straightforward way to both hypermedia and kbs [Greenleaf *et al.* 1991], [Gloor 1991], [Zhuoxun *et al.* 1992].

There are four main ways to combine kbs and hypermedia technologies:

1. A kbs system provides hypermedia features within its main framework and uses them for explanation purposes; no reverse communication. It is a uni-directional communication and only one technology exploits the functionalities of the other. Many existing kbs shells use this option [Konstantinou *et al.* 1993]. The system proposed in [Hamfelt & Barklund 1990] is a practical example of this approach.

2. A hypermedia system provides kbs facilities for query evaluation purposes. No communication from kbs to hypermedia. It is again uni-directional communication. The hypermedia only offers an additional kbs facility to users. This approach was explored in [Schwabe *et al.* 1990].

3. An integrated system provides both kbs and hypermedia features and at run-time uses them for system level tasks. This offers bi-directional communication and both technologies exploit the functionalities of each other. As an integrated approach it tends not to be an open system. This approach was proposed in [Rada & Barlow 1989] under the name *ExperText*. Later with some changes, there was *HRExpertext* [Diaper & Rada 1991].

4. A composite system using independent kbs and hypermedia systems, in

which the systems communicate with each other bi-directionally on a run-time basis through a message passing protocol, but remain as separate identities. We call this an *open architecture approach*. This approach was explored independently with different features in two papers [Bench-Capon *et al.* 1991] and [Greenleaf *et al.* 1991].

The fourth way is the basis of our approach. In Section 3.2, using this approach, we propose a general purpose framework, knowledge based hypermedia (*kbh*), in which a logic programming (LP) based kbs and a hypermedia system are loosely coupled into a composite system. The architectural detail of *kbh* is given in Section 3.3.

The open architecture approach allows the *kbh* framework to offer a relatively domain independent environment; in the sense that the hypermedia manager and the kbs shell of *kbh* remain relatively domain independent, whereas the hypermedia document base and kbs knowledge base (kb) are developed according to the application. The framework also provides a reasoning facility for hypermedia systems which on the one hand offers active document features (sending queries to the kb); and on the other hand provides a basis for conceptual retrieval. The advantages of the *kbh* framework are discussed in Section 3.4. This section also discusses the advantages of an open architecture approach compared to an integrated approach.

We have mentioned above that many authors have proposed systems which combine kbs and hypermedia technologies. A comparison of the *kbh* framework with related work is given in Section 3.5.

3.2 Proposal for Knowledge Based Hypermedia

Knowledge based hypermedia (*kbh*) is a general purpose framework in which a LP based kbs and a hypermedia system are loosely coupled into a composite system. The two components of the system, the hypermedia manager and the kbs shell, communicate with each other through a message passing protocol on a run-time basis and remain independent entities.

The motivations for the development of this framework are:

- To improve the performance of hypermedia systems and kbs's by loosely coupling them into a composite system. On the hypermedia side the presence of the kbs can enhance the functionalities of hypermedia systems and address some of their fundamental navigational and retrieval problems. On kbs side the presence of the hypermedia system can address some of the fundamental problems of user-system interaction associated with current kbs.

- To build a general framework which offers a practical intelligence distribution between user and system. In the kbs intelligence lies primarily with the system; users can submit queries and can get answers but they do not know about the details of the primary sources. Contrarily, in hypermedia the intelligence lies largely with the users who explicitly guide the system for information navigation. A composite system can make this distribution more practical.
- To build an open framework that provides a useful research platform for both hypermedia and kbs communities which can be used in various domains, and which can be altered or upgraded easily according to researchers' requirements.
- To build an open framework that allows to develop LP based tools which provide additional features and, in particular, allow conceptual dynamic linking, structured retrieval and active document features in hypermedia systems.
- To build an open framework that offers an environment in which users can exploit the useful features of declarative and imperative programming languages side by side.

A conceptual model of the *kbh* framework is shown in Figure 3.1. It is implemented using a modular based open architecture approach which provides an alternative, and very flexible, way of introducing additional structure into hypermedia. The structure, which may concern a whole range of features from semantic support to text formatting, can be modelled in the kb. Because of the open architecture the resulting kbs component, which is being developed in an LP style, is composable with suitable present or future hypermedia systems.

In the *kbh* framework, the hypermedia manager is the top-level interface to the kbs. The main aims of doing this are:

- To give due weight to the primary source documents, e.g. textual information, as we believe that such sources prepared for human readers, in particular texts,

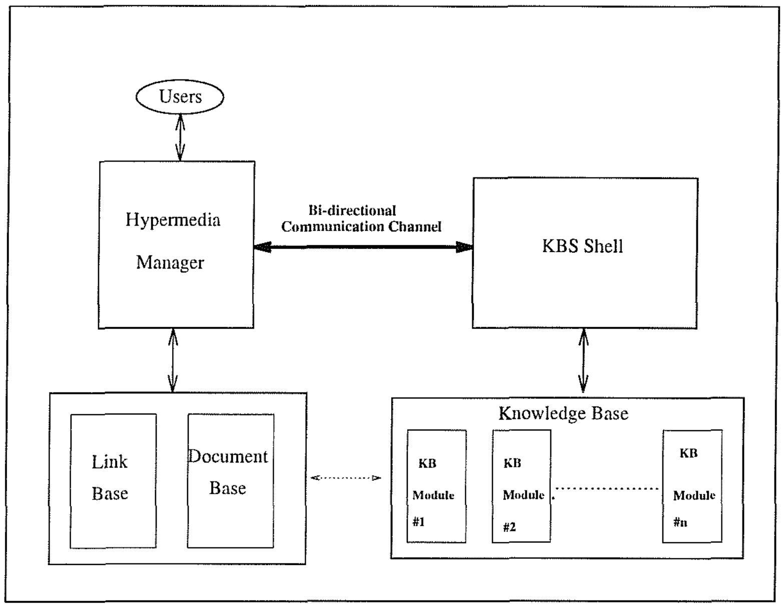


Figure 3.1: Conceptual model of knowledge based hypermedia (*kbh*).

are the most powerful means of information communication;

- To overcome the communication difficulties of kbs, as their non-friendly user-interfaces make them difficult for naive users to understand their behaviour.

The *kbh* framework allows users to utilise the functionalities of hypermedia, kbs, or both together. The communication between the hypermedia and the kbs is established by introducing a new kind of link between hypermedia and knowledge base. In typical hypermedia links lead to other parts of the hypermedia. In the *kbh* framework links can also lead to the knowledge base. Such links gives rise to a query to the knowledge base via the kbs shell. Therefore, in the *kbh* framework, users can navigate through the whole body of information in the document base using the hypermedia manager's navigational tools and also can send queries to the kbs and receive responses.

It is important to note in Figure 3.1 that there is no direct link between the hypermedia document base and the kbs knowledge base. All communication is via the hypermedia manager and the kbs shell. Nevertheless the intended use of *kbh* is that the knowledge base should model some useful aspects of the hypermedia document base. The dashed line in Figure 3.1 indicates this informal relation. What aspects of the document base are modelled in the knowledge base can vary widely and depend on the application. It will be noticed that the kb is divided into modules. This is because both different parts of the document base and different aspects of those parts can be modelled in different modules of the kb.

We want to explain two important points here. The first point concerns the distinction between the *kbh* framework and other systems built using an open architecture approach. The similarity between *kbh* and the models proposed in [Hamfelt & Barklund 1990], [Schwabe *et al.* 1990] and [Bench-Capon *et al.* 1991] is that all three are built using an open architecture approach. They use third party hypermedia systems and an LP approach for the kbs development. The model of [Schwabe *et al.* 1990] differs mainly in having a uni-directional communication channel between component systems, designed for providing active document features in a hypertext environment. The model of [Hamfelt & Barklund 1990] differs mainly as it only uses a hypertext facility for kbs explanation purposes. Communication from hypertext to kbs is not mentioned which apparently shows that the system has only a unidirectional communication, from kbs to hypertext. The model of [Bench-Capon *et al.* 1991] concerned the communication channel as bi-directional, and hence corresponds to our model of *kbh*, but it did not implement the direction in which the kbs sends messages to the hypermedia system. However, in the *kbh* framework the communication between component systems is bi-directional. Both hypermedia and kbs can send and receive messages to and from each other on a run-time basis and therefore can use the functionalities of each other for system level tasks. DataLex is another system developed using an open architecture approach. It is developed in an imperative language but as a whole it does not have much similarity with *kbh* (see Section 3.5).

The second point concerns the use of hypertext in a kbs environment. The use of hypertext for displaying textual information is a recent trend in the computer science research. Particularly, it is becoming very popular in the legal domain. Many systems offers this facility, such as:

- FLEXICON: A legal information management system [Gelbart & Smith 1993].
- ASSESS: a decision support system for the New Zealand Accident Rehabilitation and Compensation Insurance Corporation [Dayal *et al.* 1993].
- Nomos-Advisor: A prototyped expert system based on Italian VAT Law [Konstantinou *et al.* 1993].

- MPC: Malicious Prosecution Consultant; a frame-based, case-based reasoning expert system operating in the domain of the tort of malicious prosecution [Kowalski 1991].
- IKBALAS II: An object-Oriented legal kbs performing statutory interpretation in the area of accident compensation. It is a rule-based and case-based reasoning system. [Vossos *et al.* 1991].
- Loge-expert: an expert system in Quebec Housing law [Paquin *et al.* 1991].
- An expert system for the legal domain of labour law [Hamfelt & Barklund 1990].

The point which we want to highlight is that from the literature about these systems it appears that by introducing hypertext facilities in their systems, the system developers have provided an "ideal" environment for retrieving required information. None of them have mentioned any navigational or other problems of hypertext systems. The hypertext system seems to be a *magic bullet* which solves the problems related to legal information management. In contrast to this, if we review the literature about hypermedia research from 1987 to date, many authors has mentioned two main problems with hypermedia systems, disorientation and cognitive overhead, as needing to be solved to make these systems effective in the practical world. In addition to this, in Section 2.5, we illustrated some practical problems related to handling legal information which typical hypermedia systems are unable to solve. Our point about the above systems is, therefore that they are too application specific and/or not sufficiently documented to be considered as solution to the main problems of information management in this area. The main aim of the *kbh* framework is not only to provide a hypermedia facility in a kbs environment, but also to build kbs tools which solve hypermedia navigational problems in practice.

3.3 Architectural Details of the *kbh* Framework

In this section we describe the architectural details of the *kbh* framework as shown in Figure 3.2. The *kbh* framework has three main modules; a hypermedia module, a bi-directional communication channel module (BCC), and a kbs module. The functionalities of these modules are given below:

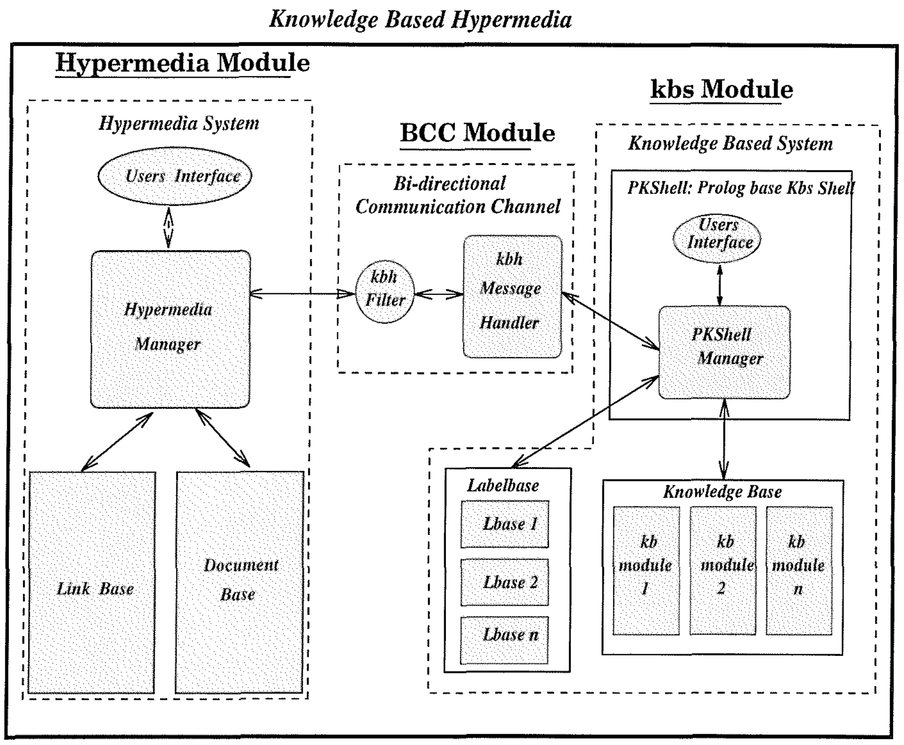


Figure 3.2: Architectural model of knowledge based hypermedia.

1. **Hypermedia Module**

   We are using a third party hypermedia system, Microcosm [Fountain *et al.* 1990], being developed in an imperative language using an open architecture approach. The function of the hypermedia module is to store/display unstructured information and to provide tools for navigation. It has four main components:

   (a) *User-Interface*

   This is a windowing system used for displaying documents and initialising the system's actions. An action means an activity that a user wants to be carried out; for example to create a link, to display a link, to hand over control to the kbs, or to send queries to the kbs.

   (b) *Hypermedia Manager*

   This controls all the activities related to the hypermedia module and performs actions initiated from the *User Interface*. Its primary function is to support navigation through the document base. It is also responsible for processing in-coming/out-going messages from/to the BCC module.

(c) ***Document Base***

All documents entered into the system are stored here. Users can create different *Document Bases* for different applications and can ask the system to load a particular one during system initialisation. Users can also remove an active *Document Base* and load a new one during a running session.

(d) ***Link Base***

This stores information about links created using the hypermedia manager's tools. Two main advantages of keeping link information separate from documents are: information stored on diverse memory devices can be linked; link information can be manipulated independently. Users can also create different link bases for different applications.

2. **Bi-Directional Communication Channel (BCC) Module**

This module is responsible for communication between the kbs and hypermedia Modules. The latter modules of the *kbh* framework are developed in two different programming paradigms, the kbs in a declarative style (using Prolog) and the hypermedia in an imperative style (using 'C'). A main task of the BCC is to build a bridge between these two languages. It has two sub-modules; *kbh_Filter* and *kbh_Message Handler*.

(a) ***kbh_Filter***

The code of *kbh_Filter* is written in 'C' under Windows environment. It provides communication between the hypermedia manager and *kbh_Message Handler*. Its main function is to process messages coming from the hypermedia manager and pass them to the *kbh_Message Handler*, and conversely to process messages coming from the *kbh_Message Handler* and pass them to the Hypermedia manager.

(b) ***kbh_Message Handler***

kbh_Message Handler's code is written in 'C' and Prolog under Windows environment. Its function is to process messages coming from *kbh_Filter* and pass them to the kbs shell, and conversely to process messages coming from the kbs shell and pass them to *kbh_Filter*.

3. **Knowledge Based System (kbs) Module**

   This is implemented using an LP approach. Some of the reasons for selecting LP are mentioned below:

   - It offers an expressive language, predicate logic, which makes it a suitable language for knowledge representation.
   - It offers a powerful framework for reasoning since it is based on predicate logic.
   - It offers a declarative style of programming.
   - Using meta-programming it allows control knowledge to be separated from domain knowledge, which allows to building a domain independent kbs shell.

   As already mentioned one of the main motives behind the *kbh* framework is to build a domain independent research platform that can provide a useful environment for future research. As such it should have a flexible skeleton in which different components can be altered easily and independently to meet users' requirements. This goal has been achieved jointly through the modular approach of the *kbh* framework and the LP approach to the kbs which keeps control knowledge separate from domain knowledge. As is clear from Figure 3.2 and Figure 3.3, the kbs part is a collection of independent modules communicating bi-directionally. Users can tailor any module or can add new ones for additional functionalities. Hence, the kbs exemplifies the openness of the system.

   The kbs module has three main sub-modules as shown in Figure 3.2.

   (a) **PKShell**: Prolog based kbs **Shell**

   This has two main modules, *User-Interface* and *PKShell Manager* as shown in Figure 3.3.

   i. ***User-Interface***

   This provides a user-system dialogue facility. It serves as a mediator in the communication between users and the *PKShell-Manager*. It translates user queries into corresponding formal

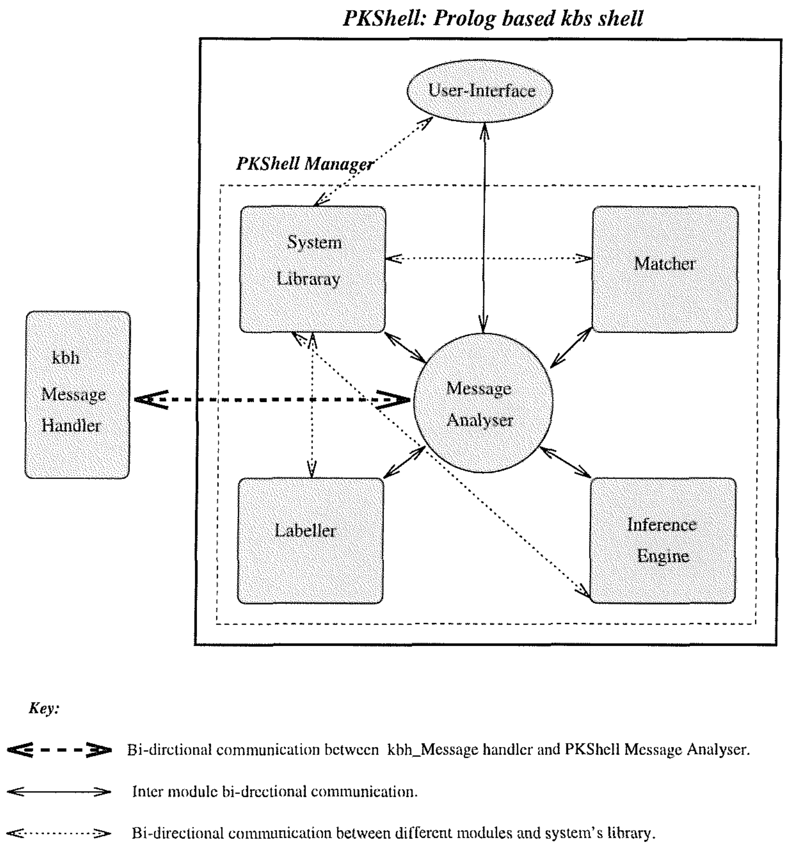

PKShell: Prolog based kbs shell
User-Interface
PKShell Manager
System Libraray
Matcher
kbh Message Handler
Message Analyser
Labeller
Inference Engine
Key:
Bi-dirctional communication between kbh_Message handler and PKShell Message Analyser.
Inter module bi-drectional communication.
Bi-directional communication between different modules and system's library.

Figure 3.3: The conceptual model of PKShell.

queries and displays results. It also communicates with other modules to perform specific tasks.

ii. *PKShell Manager*

This has five sub-modules used for various purposes. A brief description is given below. A detailed discussion is given in Chapter 7.

- *Inference engine*: This is a meta-interpreter built using backward chaining, Query-the-User, and other LP based techniques. It evaluates queries, generates results and builds proof trees. What we call 'proof trees' are essentially stored explanation traces and are central to PCR, our conceptual retrieval technique (Explained in Chapter 4).
- *Labeller*: Its function is to generate suitable labels constructed from proof trees for newly entered documents and to

store this information in a *Label Base*.

- ***Matcher***: Its function is to find labels in the *Label Base* which are similar/relevant to a submitted query and to send the retrieved information to the relevant module for further action.
- ***Message Analyser***: Its function is to sort out the incoming/out-going messages from/to the *kbh_Messages Handler* and send them to the relevant modules. It is also responsible for handling inter-module communication and user requests sent from the hypermedia. *Message Analyser* has five sub-modules which are responsible of five different task; explained in Chapter 7.
- ***System Library***: All utility predicates are available in this module.

(b) ***Knowledge Base***

This keeps domain knowledge encoded as logic programs. It can have different *kb-Modules*. The main idea behind the modular development of the knowledge base is that users can build different kb-Modules for a given application according to different possible ways of viewing the application. For example a knowledge base developed for an educational course might have two different kb-Modules, one keeping knowledge according to the instructor's view and the other according to the students' view.

(c) ***Label Base***

This also stores domain knowledge, but it has been extracted by the PKShell, using the system's semi-automatic technique for extracting conceptual information from documents (see Chapter 4) via the Labeller. The system stores this information in a special format for retrieval purposes.

One point which we want to explain is that *kbh* has two user-interfaces; one for the hypermedia manager and the other for the PKShell. This is because both systems can work independently. Our approach, of allowing two interfaces, is more flexible than using only a single one, say through hypermedia [Greenleaf *et al.* 1991], which would put some extra load on the whole system. For example, if the kbs queries the user for missing information (to satisfy a kbs query) through the hypermedia user-interface, this would need to send/get messages from the hypermedia manager which would increase the load on the system and slow down its performance. Also single user-interface does not allow to run component systems independently which makes the system integrated. On the other hand the advantage of a single user-interface, say through the hypermedia manager, is complete uniformity of presentation to the user. The two user-interface approach designed here has additional flexibility and we believe is appropriate for a research platform.

3.4 Potential Advantages of *kbh*

In this section we discuss the potential advantages of the *kbh* framework. Firstly, we mention some obvious advantages of the open architecture approach of *kbh* compared to an integrated approach. Later we discuss a particular aspect of *kbh;* domain independency.

3.4.1 Modular Based Open Architecture Versus Integrated Approach

A significant benefit of an integrated approach is that both systems use a single language so communication between them is not problematic. It has, however, many disadvantages:

- ***Domain dependency:*** Integrated systems are usually developed for a particular domain. Using them in wider domains requires major alteration and is a laborious task; especially systems developed in imperative languages.

- ***Lack of functionalities:*** Integrated systems typically provide a limited number of functionalities which sometimes do not fulfil the requirements of wider domains. HyperCard [Apple Computers 1987] and KMS [Akscyn *et al.* 1988] are a good examples of this problem. Even though both systems provide their script languages users face serious problems with implementing new ideas [Nielsen 1990b], [Yoder & Wettach 1989].

- ***Difficult to Upgrade:*** Adding new functionalities needs major alteration in the main program which requires a lot of human resources and expertise.
- ***Programming Overhead:*** Building an integrated system requires a massive amount of computer programming, time, and finances.
- ***Closed System:*** Most integrated systems are closed systems, that is the source code is not available to end-users. So end-users can not make changes and can only use those functionalities which are provided. Using a closed system for research purposes is to work with one hand tied behind one's back.

On the other hand, the problem with an open architecture approach is that the system developer needs to build a communication channel between the two systems. This requires detailed programming in the languages used in the selected systems; and in-depth understanding about the operating system and its low level programming. Another problem is that the system developer must select those systems which run under the same operating system and allow to build such communication channel. But note that all these problems are related to the system developer not to end-users. The advantages of the open architecture approach are many, including:

- ***Ease of System's Development:*** Developing a composite system using third party products saves work compared to developing an integrated system from scratch and, hence, requires less human resources, time and finances.
- ***Easy Up-Gradation, Alteration and Memory Saving:*** A modular based system is a collection of various modules that perform different tasks. These modules can be altered as needed without making any change in the others. Also the open architecture allows users to add new modules to the system. For example if the system does not provide forward chaining, a user can build a new module that can perform forward chaining and add it to the system.

  Another important benefit is that, as both kbs and hypermedia remain independent identities, the *kbh* framework allows users to run these systems separately or together. For example, if end users do not want to use the kbs, they can run hypermedia alone which certainly saves memory and, hence, affects the system's performance. In an integrated approach, however, users have to load the whole system.

- ***Exploitation of Imperative and Declarative Languages:*** The *kbh* framework provides a communication channel between two systems implemented using two important language paradigms, imperative and declarative (specifically 'C' and Prolog). Therefore the framework allows users to use utilities written in either of these two paradigms. (The system in fact supports those imperative languages which support the Windows' DLL feature). For example, suppose a user has a new idea and that can be implemented efficiently in C^{++}. In that case he/she implements the idea in C^{++} as a *Dynamic Link Library* (DLL) (explained in Section 6.3.2) and adds it to the system. On the other hand, if a user wants to implement an idea in Prolog, he/she builds a new module in Prolog and adds that to the system. As a whole the *kbh* framework provides a flexible environment for both languages.

- ***Useful Research Platform:*** As mentioned above, fielded systems offering kbs and hypermedia features are closed systems and do not encourage end-users to make changes. So users are bound to use only those facilities which are provided. On the other hand, *kbh*'s open architecture and modular structure allow users to change the system. For research purposes users want such facilities. One practical example is the development of *kbh*'s user-interface. In the early days of *kbh* development we were using LPA 1.2 Prolog. Development of the user-interface was a big issue as LPA Prolog 1.2 offered a limited style of message boxes. So we implemented this part in 'C '. But now we are using LPA Prolog 2.3 which provides useful predicates for the user-interface. During switching over we simply plugged in the newly developed *User-Interface* module and removed the old one without making any changes in other parts of the system. Another example is that LPA 1.2 Prolog and LPA 2.3 Prolog use different strategies for DLL message handling. In this case we only changed the *kbh_Message Handler* without touching other modules. These examples show the usability of the *kbh* framework as a research platform. We believe that the *kbh* framework provides an environment in which researchers can easily prototype their new ideas.

We accept that the prototype system does not provide any special language for knowledge engineering. We believe such special languages usually do not provide the full strength of basic programming languages. The prototype system supports the basic syntax of programming languages in use and allows users to exploit the full strength of these languages.

3.4.2 Domain independency

In the *kbh* framework we have selected a hypermedia manager, Microcosm [Fountain *et al.* 1990], which has a separate link base and generally strives for complete domain independency from the document base. Our prime concern is with the domain independency of the kbs shell from the kb. The basis of the kbs shell domain independency is the modular based LP meta-programming approach. Its main advantage is that it keeps control knowledge separate from object level knowledge. So, any change in the kb does not affect the kbs shell.

As we can see the kb is divided into modules in which different aspects of the document base can be modelled. Recall that in the *kbh* framework the hypermedia document base and knowledge base do not have a direct link. It totally depends on the application what aspects of the document base are modelled in these kb modules. Since these aspects can vary widely we do not expect the kbs shell to be completely domain independent. Rather we have tried to build a selection of fairly generic tools which can be available in the shell and are useful over a range of applications. Our main tool in this area is PCR, conceptual retrieval based on proof trees. (See Chapter 8 for a practical demonstration.)

3.5 Comparison with Related Work

Currently, many applications combine kbs and hypermedia features using one of the four options listed in Section 3.1. In this section we compare such applications with the *kbh* framework.

- *Knowledge Management System (KMS)* [Akscyn *et al.* 1988]: This is a commercial product of Knowledge Systems. It is a hypertext development system providing facilities to build a frame based hierarchical hypertext system. Frames have the same characteristics as anchors in a typical hypermedia system and are stored as Unix files in a directory. A frame consists of a screen-sized workspace that may contain text, graphics, or bitmaps. Users can make links between these frames. A directory of frames is called *frameset* and can be distributed across different file servers. KMS supports an interpreted action language and users can access the operating system from within the KMS system.

An additional feature is that the system provides a special frame *catalog* which has sixteen fields. Users can store information in these fields and using another special frame, *query frame*, can retrieve information with respect to these fields. This feature is similar to structured, data base search. Although there is no general inference feature in the system, through the frames hierarchy and the action language users can introduce a degree of intelligence into an application.

We can easily say that KMS can not offer comparable facilities to the *kbh* framework. The basic reason is that it has no general reasoning, only frames and action language, therefore kbs technology can not be exploited fully. As regard the action language Yoder and Wettach [Yoder & Wettach 1989] say: "Among the frustrations we encountered using KMS, the most bothersome involve its action language....".

- ***StrathTutor:*** *StrathTutor* [Kibby & Mayes 1989] uses HyperTalk, the script language of HyperCard. Knowledge about browsing and finding relevant links is stored in frames. Patterning matching, implemented as HyperTalk procedures, is used for dynamic linking between documents. Users can also create links using HyperCard tools. This system is aimed at small scale hypermedia applications and could face serious problems in a large scale application [Tochtermann & Zink 1995]. The limitation of this system is due to HyperTalk which is a subset of an object oriented language with limited functionalities [Nielsen 1990b]. Also the system does not offer any query evaluation facility. The author claims that the system has a factor of intelligence, achieved through frames and pattern matching features. But, as with KMS, the system does not have a general inference facility and therefore cannot fully exploit the strength of kbs technology as *kbh* is able to.

- ***LINNTUS PB*** [Briggs *et al.* 1993]: is developed using an integrated approach. It is a kbs application which uses hypertext features to display various relevant cards, having a range of information, under the control of an inference mechanism. The main users of this system are pharmacists who receive professional updating. Medical diagnostic knowledge is stored in the kb in heuristic rule format. A card is the basic unit of the system. The system has different types of card. Some contain simple textual information and some provide editable windows along with textual information. The contents of these windows can be changed dynamically. The system gets queries from users through special cards and then applies kb rules to infer results which are stored in various output cards. Its hypertext feature allows users to navigate through the cards, giving information and advice about drugs, symptoms and diseases.

  This system is a special purpose integrated application designed for a particular task. Navigation through the information is controlled by the inference facility, but it only handles a specific type of information. The system's use in wider domains has limited scope.

- ***HyperLex*** [Yoder & Wettach 1989] is an example of an integrated approach. It uses the KMS hypertext system for managing knowledge about intellectual property. HyperLex builds a hierarchy of frames that contain various types of information. The top level frame has links to frames such as group bulletin boards and calendars; and these are connected with further sub-frames in a hierarchical fashion. The frames are also linked to the index of legal documents, such as patents, contracts and lawsuits, which again are represented in hierarchies of frames. Users can access information by navigating through these frames in a conventional hypertext fashion.

  In another application, database of litigation documents, HyperLex uses another feature of KMS called the *catalog* frame which has a list of 16 fields. The system developer stores information in these field such as document type and document name. Users can retrieve information related to these fields using a special query frame. As for as comparison with *kbh* is concerned this project is no more than demonstration of the KMS system and our remarks above concerning KMS applies.

- Schwabe, et. al in [Schwabe *et al.* 1990] have proposed a system called Normative Knowledge in Engineering. They use a multiple approach to knowledge representation: hypertext cards having textual information about norms; AND/OR graphs to represent norms and expert points of view. The knowledge about AND/OR graphs is encoded in a Prolog program. HyperCard is used to implement the hypertext .

  The system provides an index of terms. By selecting any one, the user can get further information and can navigate among related cards in a hypertext fashion. There are two user-levels, one for experts and one for beginners. These levels being chosen initially by the user. Beginners can not access specialist points of view. A special button, *Assistant*, is available to experts that leads to the point where the expert point of view is interpreted.

  For each anchor a *Logical Interpretation* button is available which, when pressed, presents the fragment of the logic program related to this anchor. This feature is also available only for expert users. The system has a special button *select Goal* which, when pressed, shows various user goals available in the system. The user can select any one and the system will run it as a Prolog goal. Users can see the result, but it is not clearly defined how this is achieved; whether Prolog is running the selected goal directly or a kbs shell is being used. Nevertheless, this system is a good example of a hypermedia system offering kbs facilities to its users. This system only uses unidirectional communication, from hypermedia to kbs, and offers active document features. kbs facilities are not used for conceptual retrieval or for any other purposes.

  Although, this system has some common features with the *kbh* framework, it does not support bi-directional communication between the component systems. Also its kbs component can not support hypermedia system for system level tasks. Therefore it cannot offer functionalities similar to the *kbh* framework.

- ***DataLex*** [Greenleaf *et al.* 1991] uses an open architecture approach, combining expert system, hypertext system, and information retrieval into one general purpose tool. It is being applied to the Australian Privacy Act 1988. The authors stated that it has been developed for commercial use rather than as a research platform. DataLex has six separate programs: YSH, a rule based expert system shell having a quasi natural language interface; AIRS, a free text retrieval system; LES, a text animation package and automated document generator; HYPE, a hypertext engine; and PANDA, an example-based shell for modelling legal precedents. All six components are implemented in 'C' in a Unix environment.

DataLex has three engines - inference engine, hypertext engine and information retrieval engine - which process legal knowledge in different ways. Each engine requires its own form of knowledge representation. There is a common user interface via the hypertext engine. The approach taken to integration is : the system appears to end users as a single unit whereas internally all components communicate with each other and all parts of the system share knowledge and data on run-time basis.

The DataLex's hypertext engine is a typical hypertext system with a fixed document size and links coming up on the screen with mark-up. For dynamic linking the system creates mark-up scripts for each category of document such as statutes, regulation and cases.

The system uses conventional retrieval techniques for retrieval purposes. The retrieval engine relies on five levels of concordance: chapter, article, paragraph, and words. Interaction between hypertext and inference engine is that related terms are attached with a kbs goals (similar to an active document features). The result of inference engine is shown in hypertext document with all the terms been used in inference mechanism appeared as mark-up anchors. Users can navigate through these anchors.

DataLex has some common features with the *kbh* framework also it is designed for the legal domain. In Table 3.1 we compare *kbh* and DataLex. In this table we show that although both systems are developed using an open architecture approach, they have quite distinctive features.

- *Animated Hypertext:* [Bench-Capon *et al.* 1991] proposed a similar kind of system using open architecture approach. A prototype system concerning Mobility Allowance, a UK Social Security benefit is developed. It uses StackMaker

Table 3.1: A comparison between DataLex and *kbh*

DataLex	*kbh*
- Aimed at commercial application	- A research platform.
- Developed using single imperative language by a single vendor. Different components of the system work separately, however, they are integral parts of the main system.	- Developed in two very different languages, C and Prolog. The system's main component are independent applications which can work independently. Communication takes place on a run-time basis.
- kbs is developed using imperative language techniques.	- kbs is developed using logic programming techniques.
- A closed system. End-users can not make changes in the main system.	- An open system. End-users can make changes (see Section 8.4).
-Upgrading is difficult. Only system developers can do this.	- Upgrading is very simple and end-users can add functionalities.
- Hypertext dynamic linking is based on conventional retrieval techniques.	- Hypertext dynamic linking is based on conceptual retrieval kbs techniques (see Chapter 4).
- Hypertext links have no information about contents of linked documents and are simple GOTOs.	- Hypertext links have conceptual information about linked documents (see Section 5.3).
- Displays available links in typical hypermedia fashion; has many problems.	- Displays link information according to the level of relevancy with user requests.
- Displays full text of documents like conventional IRS; users have to find information reading whole text of the retrieved document.	- Provides summaries of retrieved documents before displaying contents (see Section 8.5.2).
- Hypertext is being used to overcome the kbs problems, but, kbs is not used to solve hypertext problems.	- Hypertext is used to overcome kbs problems and kbs is used to overcome hypertext problems (see Section 5.5).
- Uses a closed hypertext system developed by system developer.	- Uses an open third party hypermedia system.

[Hutchings *et al.* 1991] implemented in HyperCard as the hypertext and a kbs shell, Skilaki [Sergot & Cosmadopoulos 1990], implemented in Prolog. The original text of a Mobility Allowance leaflet is stored in HyperCard cards and the logic of the underlying legislation is represented as a logic program in the kb. The main purpose of this system is that users can gain a general understanding of the legislation from reading the hypertext and in addition apply this legislation to a particular case by querying the knowledge base. The system provides a special link to send queries from hypertext to the kbs. Users see results through the kbs shell's user interface. Animation is a powerful *active document* feature. The proposed system, however, did not use kbs facilities for any other purpose. Although the system envisaged communication from kbs to hypermedia so as to annotate the hypertext, it did not implemented this; thus messages were only sent from StackMaker to the kbs shell. Similar to the [Schwabe *et al.* 1990]'s system it only offers active document features and so does not match the *kbh* framework.

3.6 Conclusion

We started this chapter by noting that new trends of on-line information management research are broadly based on combining, in various ways, the main contributing technologies - hypermedia, kbs and IR. In chapters 2 and 3 we argued that hypermedia is emerging as the de facto, front runner technology for managing unstructured on-line information and that therefore the key research issue was to support hypermedia via the other technologies. We also noted that IR is an essentially orthogonal technology which can be added in a straightforward way to both hypermedia and kbs. Our prime research effort has been, therefore, towards the support of hypermedia via kbs technology. We discussed four possible options for combining the two technologies. One of them, an open architecture approach, is the basis of our proposed system.

The key problem with hypermedia systems, as explained in Section 2.4 and 2.5 is that they offer little or no guidance in helping users to find information useful for the activities they are currently engaged in. Most users are not interested in exploring hypertext information spaces *per se* but rather in obtaining information to solve problems or accomplish tasks [Fischer *et al.* 1989]. The purpose of adding kbs should be to address this problem. We proposed a general framework, *kbh,* for loosely coupling hypermedia and kbs. We argued that this heterogeneous framework enhances the applicability of each of its components. On the hypermedia side the presence of the kbs can address some fundamental navigational problems. Conversely, the presence of the hypermedia can address some of the fundamental problems of user-system interaction associated with current kbs's. The proposed architecture allows the power of LP techniques, and Prolog in particular, to be exploited for kbs development without undue consideration of interface questions. Using the strength of LP, to develop information management tools as part of the kbs shell, hypermedia systems can not only handle their navigational problems but can also offer many other facilities such as conceptual retrieval, reasoning about system behaviour, and providing active document features.

We mentioned some advantages of the *kbh* framework which other systems developed using an integrated approach do not provide. One disadvantage of the integrated approach is domain dependence; it is not easy to use a system developed using this approach in wider domains without making major changes in the main program. At the same time up-grading integrated systems is quite a complicated task, especially when they are written in imperative languages and their source codes are not available. The *kbh* approach, on the other hand, provides many advantages, such as an open architecture, relative domain independency and easy up-gradation (demonstrated later in our case study).

In the final section we compared *kbh* with related systems which offer hypermedia and kbs facilities. One main difference is that most other systems offer only unidirectional communication, whereas, *kbh* offers a bi-directional communication link which allows both technologies to support each other for system level tasks. Another distinctive and important feature of the *kbh* framework is that it provides a basis for conceptual retrieval and a useful research platform that allows implementation of new ideas from both hypermedia and kbs research communities. *kbh* is consistent with the goal set out by Guy Vandenberghe, in his inaugural lecture in 1985, for future legal Informatica systems. In his opinion one of the most essential aspect of those systems will be the integration of the various possible applications of computers in law, such as word processing, office automation, legal text retrieval, and legal expert system [quoted in Oskamp & van den Berg 1990].

In our opinion *kbh* attacks the crucially important combination, hypermedia , kbs and allows other features to be added within an open framework. In the following chapter, using the *kbh* framework we have designed some LP based information management tools for hypermedia systems.

4 Proof Tree Based Conceptual Retrieval

4.1 Introduction

In this chapter we propose a semi-automatic conceptual retrieval technique - **P**roof tree based **C**onceptual **R**etrieval (PCR) - which allows conceptual retrieval in hypermedia systems by making hypermedia links conceptually enriched. This provides a basis for conceptual dynamic linking in hypermedia systems (demonstrated later in our case study). PCR is a logic programming (LP) based technique. The reason for choosing the LP path to developing PCR is because it allows a declarative treatment of the kb's in the system and, more crucially, of the reasoning process based on this knowledge. This reasoning process forms the basis of PCR.

Section 4.2 briefly introduces the basic concepts, definitions and notations of LP. The theory of PCR is described in Section 4.3. In 4.3.1, we have generalised, in a schematic way, the idea of query-the-user QTU [Sergot 1982] to include three categories of knowledge: knowledge formalised in the kb (**kb_system**); knowledge in the user's head (**kb_user**); and knowledge in hypermedia documents (**kb_doc**). Interaction between the user and the other knowledge sources expands the boundary of **kb_system** without changing the totality of knowledge. In Section 4.3.2, after presenting the general idea, we introduce our specific proposal for representing conceptual knowledge in documents, proof tree labels. The essential idea is that an appropriate query, related to the document's contents, is posed to the kbs. The execution trace generated in the interactive processing of the query is then used as a label for the document. (Note there can be more than one such label.) We have adopted the term 'proof tree label' for such a trace.

The overall purpose of PCR is to find relevant documents, and the basic idea is to do this by matching proof tree labels. We have designed a proof tree label matching algorithm: by matching predicate names of the proof tree label nodes, the algorithm generates a single number (*matching weight*) as a measure of similarity between two proof tree labels. In Section 4.4, we justify our approach of matching predicate names for matching proof tree labels. Section 4.5, by taking some hypothetical cases, illustrates different aspects of the matching process. This section also explains that the weighing scheme used to generate the matching weight is to some extent arbitrary. The matching algorithm itself is defined in Section 4.6. Finally Section 4.7 defines a percentage matching weight which is used to give a clue to users about the level of relevancy between retrieved documents and their request.

4.2 Logic Programming

Among other computer programming styles, logic programming (LP) is a popular approach to building kbs. One reason for this is that it is well-suited to knowledge manipulation since logic provides a high level, human oriented formalism for computers. An important characteristic of LP is that it keeps control (proof method) separate from the knowledge base. This feature provides a declarative style of programming in which the programmer only defines what to do; not how to do. The 'how' is carried out by the implementation of the proof method. So a logic program, from a declarative point of view, is a set of logical rules (clauses) and facts which describe relationships between individuals.

There is also a procedural side to LP. The execution of a program depends on the default proof method provided by the language and generally this affects the efficiency of the program. Programmers usually exploit the procedural features of logic programs to achieve efficient programs [e.g. Wolstenholme 1991]. Logic programming languages usually provide a number of non-logical primitives which introduce non-declarative aspects in logic programs. These primitives are commonly used, for example, for input/output, database operations and pruning. Provided these non-logical features are used with care, so as not to obscure the declarative reading of the program, LP provides an environment in which programs are, in principle, easier to construct, easier to understand, easier to alter and easier to adapt for other purposes than in a procedural language.

On the other hand, writing programs in procedural languages has other strengths.

It involves more explicit flow-charting, algorithm writing, and encoding. Although this may be complicated, it provides two advantages. Firstly, programmers can not only define what the problem is but explicitly how to solve it. Secondly, they have greater opportunity to optimise the execution of the program. But, the problem with procedural languages is that they expect detailed knowledge from users concerning the problem. So a program becomes a mixture of control and declarative statements which prejudices the readability of programs. Hence editing, debugging and adapting of program becomes difficult.

In this thesis we have chosen to take the LP path to developing the kbs part of the *kbh* framework as it allows a declarative treatment of the kb in the system and, more crucially, of the reasoning processes based on this knowledge. The latter are implemented by meta-level programs, the kbs shell, PCR, and other LP based information management tools. We believe that this use of LP exploits its strength for supporting reasoning, which is an essential element of PCR, and that this argument for its use outweighs any concession to reduced efficiency. Even as regards efficiency there is a reasonable expectation that research on program transformation [e.g. Hogger 1981] and LP compilers [e.g. King & Soper 1994] will lead to adequately efficient programs. Although we have chosen the LP path for the kbs, within the context of the whole *kbh* framework, we also accept the advantages of a procedural approach for other parts of the system. Many hypermedia systems are implemented using imperative languages and, in particular, the third party product, Microcosm [Fountain *et al.* 1990], adopted for our prototype uses 'C'. The BCC we have developed is a hybrid of 'C' and Prolog. We now continue our discussion by introducing the basic concepts, definitions and notations of LP which will be used later in the chapter to define PCR.

4.2.1 First Order Predicate Logic

LP is based on first order predicate logic (FOPL). In FOPL predicates are used to define the relationship between individuals and this provides a flexible basis for LP. There are many advantages to using FOPL as a knowledge representation formalism. Firstly, it has a formal semantics which provides a precise description of the meaning of an expression in the formalism. Secondly, it has well understood properties concerning completeness, soundness and decidability. It is possible to prove that a proof theory is sound and complete in many prove methods, but in general such proof methods can only be semi-decidable. Finally, but not least, FOPL has greater expressive power compared to many other formalisms (e.g. semantic nets).

The language used for FOPL consists of predicate symbols, individual variables, individual constants, function symbols, logical connectives, quantifiers and brackets. In FOPL, an atomic formula has the form p(T1, ..., Tn) where 'p' is a predicate symbol and the Ti are terms. Each term denotes an individual, constructed from constants, variables and function symbols. A set of well formed formulas (wffs) is defined for FOPL as follows [Frost 1987]:

- *an atomic formula is a wff;*
- *if A and B are wffs then A,* $\neg A, A \wedge B, A \vee B, A \rightarrow B, A \leftrightarrow B$ *are wffs;*
- *if A(x) is a wff then* $\forall$ *x [A(x)] and* $\exists$ *x[A(x)] are wffs.*

These wffs are in one of the 'standard' notations for FOPL. It is possible to convert wffs in the standard notation into a normal form, clausal form, which is the basis of LP. A clause, in general, is a sentence of the form $Q \leftarrow P$ read 'Q if P'. Here P is a conjunction of atomic conditions and Q is a disjunction of atomic conclusions; that is each condition and conclusion is an atomic formula. So the general form of a clause is:

$$A1 \vee A2 \vee \ldots \quad \leftarrow \quad B1 \wedge B2 \wedge \ldots$$

where each Ai and Bi is an atom. All variables in a clause are universally quantified over the whole clause. Each Ai is called a conclusion and each Bi is called a condition.

A clause with one or more conclusions is said to be a positive clause; one with no conclusion is said to be a negative clause. A clause with one or more conditions is said to be a conditional clause; one with no condition is said to be an unconditional clause. Horn clauses are those which have at most one conclusion and non-Horn clauses are those which have more than one conclusion. LP is based on the Horn clause sub-set of FOPL [Kowalski 1979], [Robinson 1992].

4.2.2 Logic Program

A logic program consists of a set of Horn clauses of the form

$$A \leftarrow B1 \wedge B2 \wedge \ldots\ldots \wedge Bn \qquad (n \geq 0)$$

where A is an atom, called the head or the conclusion of the clause, and B1, B2, . . . , Bn are atoms, called conditions, which make up the body of the clause. All variables in the clause are universally quantified at the front of the clause. We use the following terminology: a program clause with no condition is called a 'fact'; and a program clause with at least one condition is called a 'rule'. (Sometimes we use the term 'rule' generically to include rules and facts.) The execution of a logic program is initiated by a query of the form $\leftarrow G$. G is called the goal and G has the form

$$\text{G1} \wedge \text{G2} \wedge \ldots\ldots \wedge \text{Gn}$$

where the atoms G1, ... , Gn are called sub-goals of the goal G . Evaluation of a program P and a query $\leftarrow G$ is an attempt to produce a constructive proof such that $P \vdash G$, i.e. the program P proves the goal G [Kowalski 1979], [Lloyd 1987].

4.2.3 SLD-Resolution and Negation by Failure (NBF)

As well as being based on the Horn clause sub-set of FOPL, LP also adopts a particular (sound and complete) proof theory. It uses a refutation procedure known as SLD-resolution which is based on the resolution inference rules. SLD-resolution stands for SL-resolution for Definite clauses. SL stands for Linear resolution with Selection function [Kowalski 1979], [Robinson 1992]. An SLD derivation is a sequence of resolution steps; the full definitions are given in [Lloyd 1987]. An SLD refutation of a goal G and program P is a finite SLD-derivation G, G1, ... , Gn which has the empty goal □ as the last goal in the derivation. If Gn = □ then the refutation has depth 'n'.

In practice LP is generally considered to be augmented by a form of negation, called *negation by failure* (NBF). In this case the Bi in the definition of a Horn clause and the Gi in the definition of a goal (in Section 4.2.1), may each be either an atom or a negated atom (i.e. they are literals rather than atoms). During execution of a program whenever a negated goal is encountered the NBF rule is used:

$$\leftarrow \text{not(G) succeeds iff} \leftarrow \text{G finitely fails}$$

Thus NBF replaces a negated goal with its corresponding un-negated version and starts a sub-computation based on the latter. The effect of this is that LP is still based on the

Horn clause sub-set of FOPL and SLD resolution. The combination of SLD with NBF is called SLDNF resolution. Now we expand our discussion to an important topic, SLD search trees, which is useful for the purpose of defining PCR. For PCR we do not allow the possibility of negation in programme clauses and therefore it is sufficent to discuss SLD only rather than SLDNF.

4.2.4 SLD Search Tree

SLD is a sound and complete proof theory, specifying the form of individual steps (resolution steps) which make up a proof. All possible ways of applying the proof theory, both to an initially given goal and to the goals derived from it, determine the search space for the given goal G and program P. An SLD search tree is a certain type of tree which represents the search space.

According to Lloyd [Lloyd 1987], an SLD search tree of a goal G and program P satisfies the following conditions:

- G is the root node of the tree.
- Each node of the tree is a definite goal.
- Let A1, . . . ,Am, . . . , Ak (k $\geq$1) be a node in the tree and suppose that Am is the selected atom. Then, for each input clause $A \leftarrow B1, \ldots, Bq$ such that Am and A are unifiable with a most general unifier (mgu) Θ, the node has a child (A1, . . . , Am-1, B1, . . . , Bq, Am+1, . . . , Ak) Θ
- Nodes which are the empty clause have no child.

Figure 4.1 shows a finite SLD search tree for goal 'a' and Prolog program[1]:

```
a:- b, c, d.
a:- e, f.
c:- g, h, i.
f:- j, k.
b.  g.  i.  e.  j.  k.
```

[1]Since it is Prolog we use Edinburgh notation and the serach rule is text-order and selection function (computation rule) left to right.

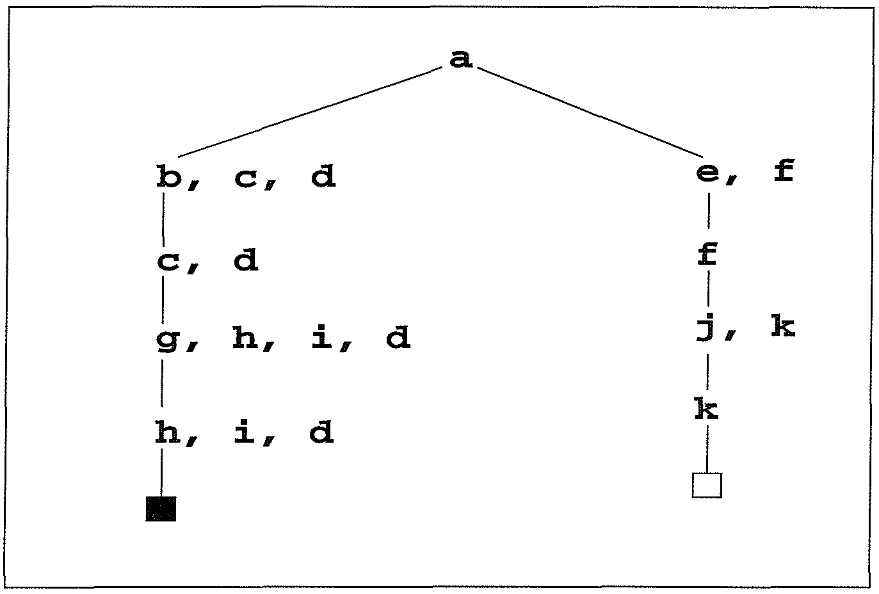


Figure 4.1: A finite SLD search tree

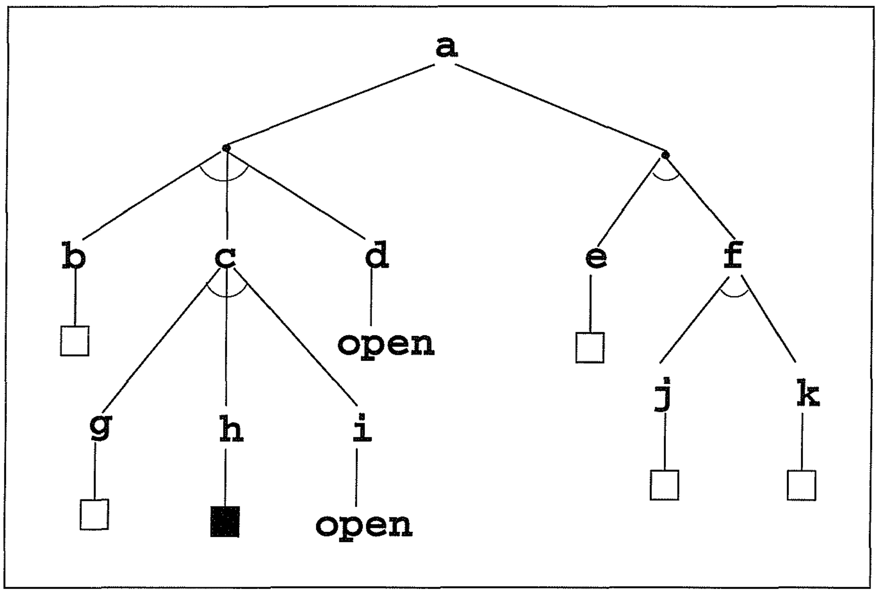


Figure 4.2: An or-and-tree

A branch corresponding to a successful derivation is called a *success branch*. A *success branch* always ends in the empty goal (□). A branch corresponding to a failed derivation is called a *failure branch*. It will be seen that a finite SLD search tree is an or-tree in which each branch ends either in success (the empty clause □) or failure (labelled with ■). Each failure branch (ignoring the label ■) ends in a non-empty goal with the property that the selected atom in the goal does not unify with the head of any program clause.

A finite failure SLD search tree is one which is finite and contains no success branch.

Figure 4.2 shows the same computation represented as an or-and-tree. In this case each or-branch of the previous tree is expanded into an and-tree. Notice that in this representation the leaves of each such branch can be catagorised into three types: success (□), failure (■) and open. The 'open' category denotes an unsearched atomic goal.

In the following we use the term 'proof tree' to denote one branch of the SLD search (or) tree represented as an and-tree. The definition for our 'proof tree labels', given in the following sections, are all proof trees in this sense.

4.3 Proof Tree Based Conceptual Retrieval (PCR)

An important requirement of conceptual retrieval is that it not only analyses conceptually the contents of documents, but also of user requests. Furthermore the adopted search strategies must be very flexible [Dick 1987], [Wildemast & de Mulder 1992]. Ideally a fully automatic system would achieve these goals - examples of some current attempts are [Mulder *et al.* 1993] and [Mulder & Combrink-Kuiters 1996] - but this approach faces serious problems. Just to give one example, concerning semantic ambiguity, in a navigational session a user exploring information about legal rights might next want to see information about *right* in the context of direction. The system is likely to retrieve information in the context of legal rights, not in the context of direction, as it knows from its previous experience that the user was exploring information about legal rights.

One way to alleviate the problems of automation is to aim for a semi-automatic method, in which machine and human work together during document storage and retrieval. The advantage of a semi-automatic method is that the human can guide the system for retrieval. We believe that this is a flexible and practical method. It puts an extra overhead on the human but the retrieval will be more effective and more meaningful. To provide conceptual retrieval in hypermedia, we have developed a new semi-automatic technique called Proof-tree based Conceptual Retrieval (PCR). PCR is an LP based kbs technique and exploits three important features of logic programming: Query-the-User, proof trees, and meta-level programming.

The main idea of PCR is to build a dictionary of general concepts from a particular domain by formalising their definitions as Horn clauses using simple available knowledge. Using these definitions, the system extracts conceptual information from documents and from users by running appropriate logical queries. During the query evaluation process the system asks users about any missing information needed to satisfy the definitions. There are two distinct, but closely related, situations when PCR comes into play: during an interactive dialogue with the kbs the user may request to see relevant documents (*dialogue situation*); when entering a new document into the system it needs to be labelled (*labelling situation*). In the dialogue situation, users provide requested information according to their current circumstances. In the document labelling situation, users provide the information from the contents of the document. At the end of this interactive session, the system generates a proof tree (execution trace) of the processed query using the supplied information. In the dialog situation, the system uses the proof tree as a source of conceptual information about the user retrieval request. In the labelling situation, the system uses the proof tree to label the document from where information is being extracted and stores this information as a 'proof tree label' in the Label Base for future retrieval. During retrieval, by matching proof tree labels, the system finds documents having similar/relevant information. The basic idea of PCR is explained in the next two sections.

4.3.1 An Extension to Query-The-User Philosophy

Logic and logic programming have been widely advocated as being suitable for representing and reasoning about verified knowledge [Kowalski 1979], [Sergot *et al.* 1986], [Bundy 1987], [Sergot 1988]. However, the standard inference mechanism of Prolog can not generally be used directly to solve problems formalised as logic programs since the kb's are generally incomplete. So many interesting queries will fail inappropriately, or a control error or some other undesirable behaviour will result. The reason for this incompleteness is that it is not feasible to store all data that is required to solve specific problems in a kb [e.g. Wolstenholme 1991]. Building interactive logic programs, which ask users for missing information, is the solution to this problem. Writing interactive logic programs declaratively was a problem in LP as the input/output predicates introduce non-logical features. Understanding interactive programs without knowing how they will behave is quite difficult. Sergot [Sergot 1982] offered a solution to handle these difficulties in a declarative manner. His proposal, Query-The-User (QTU), is now becoming an essential part of interactive meta-interpreters written for logic programs.

QTU was developed as a means of organising dialogue between system and user in a declarative fashion. In this technique, the user's knowledge is seen, conceptually, as an extension of the system's kb (**kb_system**) and both may be queried. The object level program makes no reference to the QTU procedures and so maintains its declarative reading [Sergot 1982].

In his paper Sergot shows how the user can be considered as an extra kb, full of ground assertions. Information 'told' by the users can be added to the **kb_system** with some additional tag, that does not affect the declarative meaning of **kb_system**. Adding this information offers many advantages, such as not asking the same question if it has been asked before avoiding inconsistent user answers; using 'told' information for reasoning about 'how' and 'why' type of questions. According to Sergot's proposal, considering users as an additional logical kb, a new logical definition to the (global) kb of an interactive system is:

kb = kb_system + kb_user

Sergot says of the transfer of knowledge from one part of the kb to the other that the union of **kb_system** and **kb_user** remains static and fixed. Only the boundary between the two components moves, and then only to make the machine take some of the burden from the user [Sergot 1982].

Following this idea we make a further proposal that with the user's help information stored in documents (**kb_doc**), available in the user's hands, can be transferred intelligently into the kb. Thus we can define the whole kb as

kb = kb_system + kb_user + kb_doc

Our conceptual retrieval technique is based on this idea. We consider that information is transferred into **kb_system** from the other two parts, **kb_user** and **kb_doc**, using various LP based techniques. In some sense, it can be said that transferred information has been stored within the body of hypermedia links. These links then can be used for conceptual dynamic linking.

For example, if the (incomplete) definition of some concept is available in **kb_system** then running it as a logical query could extract information from the document about the queried concept[2]. The information flow will look like this:

kb_doc $\rightarrow$ **kb_user** $\rightarrow$ **kb_system**

In our technique users act like a mediator who not only helps the system to extract conceptual information from documents but also guides the system during retrieval sessions.

4.3.2 Proof Trees as Conceptual Knowledge Representation

In the previous section we discussed the relationship of our retrieval technique to QTU. The basic purpose of PCR itself is to store the information extracted from documents within the body of links, in a suitable format, which the system can use for useful purposes - basically as conceptual links. For this purpose PCR uses proof trees.

A proof tree, in our sense, is defined in term of the derivation trace produced when a query is evaluated (see Section 4.2.4). We use proof trees as conceptual labels because they contain information about how a query is solved and which rules and facts were used during the evaluation process. Also, Sergot [Sergot 1991] mentioned, in LP a proof tree is not merely a trace of computation but a logical proof of any conclusion reached.

Insofar as the concepts are defined in **kb_system**, the proof tree of a queried concept can provide the information used to satisfy the definition of that concept. In other words it can be said that the proof tree of a queried concept is one kind of representation of the conceptual information of the concept. Although we talk here about conceptual information, it is worth noting that these proof-trees are like the explanation trace in LP or kbs. If, during query evaluation, users provide information, either according to their current interest or from a document, then it can be said that the proof tree of the queried concept has conceptual information about the user's current interest or about the contents of the document. We use such a proof tree as a conceptual label of the document from where the information is extracted. PCR is based on this

[2]As the information required to satisfy these definitions will not be available in **kb_system** so the system will ask the user who can provide it from the body of the document.

idea: that such labels are a flexible and rich representation for conceptual information. In PCR, proof trees are used as labels for linked documents.

Now we define the structure of PCR proof tree labels. It was mentioned in Section 4.2.4 that a 'proof tree' of a goal (G) and program (P) corresponds to a branch of the SLD search tree represented as an and-tree. If G succeeds, that is $P \vdash G$, the proof tree is a success branch of the SLD search tree. If G fails, that is $P \nvdash G$, the computation generates a finitely failed SLD search tree consisting (in general) of many branches. When QTU is allowed, as described above, we need to distinguish the success or failure of the leaves from **kb_system** or **kb_User + kb_doc**. For this purpose we label the leaf nodes by tagging their status as shown in Table 4.1. In PCR, a

Table 4.1: Tagging status with leaf nodes

Leaf Nodes Status	**Resolution Condition**	**Proof**
success leaf node	kb_system $\models$ Node	(Node, fact)
success leaf node	kb_system + kb_user + kb_doc $\models$ Node	(Node, told_true)
failed leaf node	kb_system $\not\models$ Node	(Node, fail)
failed leaf node	kb_system + kb_user + kb_doc $\not\models$ Node	(Node, told_fail)
open leaf node	unsearched	(Node, open)

proof tree label is one branch of the SLD search tree for the goal G and the program P. This branch (of the or-tree) is itself an and-tree and can be a success branch or a failed branch. A more detailed description of the success and failed proof tree labels is given below.

Success Proof Tree Label:

A *success proof tree label* is the and-tree of the succeeding or-branch of the SLD search tree such that:

kb_system + kb_user + kb_doc $\models$ Goal (Labelling situation)

kb_system + kb_user $\models$ Goal (Dialogue situation)

This and-tree is fully searched and the leaves are tagged with their status. An example is shown in Figure 4.3(a).

Failure Proof Tree Label:

A *failure proof tree label* is the and-tree of the deepest reasoning or-branch of the finite failure SLD search tree such that:

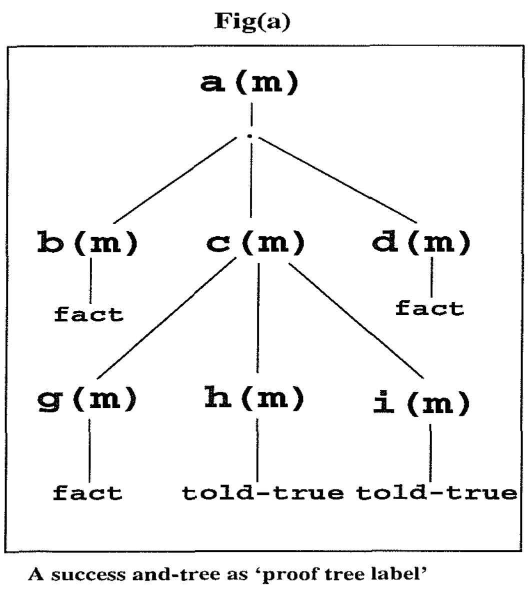
kb_system + kb_user + kb_doc $\not\models$ Goal (Labelling situation)

kb_system + kb_user $\not\models$ Goal (Dialogue situation)

The deepest reasoning or-branch of a finite failure SLD search tree is the branch which solves the greatest number of atomic goals (or if there is more than one the first of these). This and-tree is searched up to a (first) fail leaf. Any nodes to the right of this fail leaf are tagged unsearched (open). All leaves are tagged with their status, as in Table 4.1. An example is shown in Figure 4.3 (b).

Figure 4.3 shows both success and failure proof tree labels for goals a(m) and a(n) for following Prolog program:

```
a(X):- b(X), c(X), d(X).
a(X):- e(X), f(X).
c(X):- g(X), h(X), i(X).
c(X):- r(X), s(X).
b(m).  g(m).  d(m).  b(n).  r(n).
```



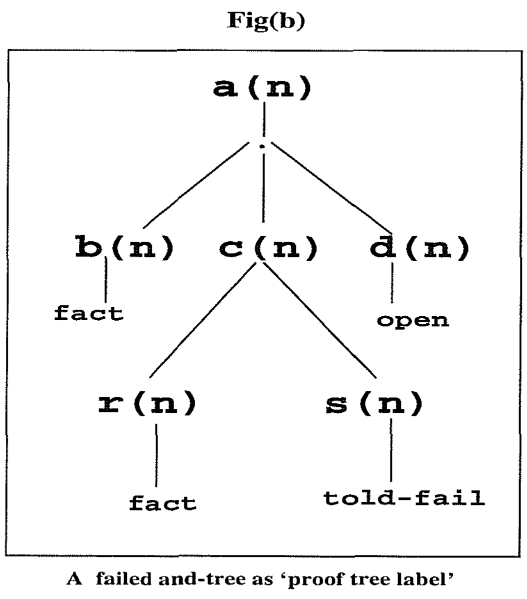


Figure 4.3: Success and failure proof tree labels

The proof tree label shown in Figure 4.3(b) is arrived at as follows. The goal a(n) generates a finite failure or-tree and of all the branches the one shown solves more atomic goals (in this case b(n) and r(n)) than for any other branch.

The definition for proof tree label given above are those used in our prototype. The implementation is given in Chapter 7. We realise that our choice of label is to some extent arbitrary and the implementation of alternative label structure, we believe is an interesting area for further research.

As far as the success label is concerned it might be argued that the choice of the first success branch is arbitrary. In practice this is not a very important point, because in typical applications the kb admits only one successful solution. For example an applicant for British citizenship will apply under one ground, say on the grounds of being a citizen of a Dependent Territory, and the case report will relate to this area of the legislation. Our technique could be extended to labels comprising more than one success branch if this was useful, but in our prototype we choose the first success branch only.

There is more room for argument concerning our choice for failure labels. The first point is why do we choose only one failure branch from the failure tree rather than the whole finite failure SLD search tree. The most important reason is that we do not want to include in the label many trivial reasons for failure. We can assume that there is at least one significant reason for failure and it is this which we hope to pick out, following Wolstenholme [Wolstenholme 1991], by selecting the failure branch with the deepest reasoning. A practical bonus of this is that success and failure labels now take the same general form of a single and-tree. There is still room for choice about how we represent this failure branch as a label.

We have considered the following generalisation of the failure label defined above. When a failure node is reached the user is asked whether they wish to explore the and-tree further. If they answer 'yes', the search is continued up till the next failure or the end of the tree. The process can be continued until the user answers 'no' or the end of the search is reached. The label generated by this generalisation can range from the label defined above for our prototype to a fully searched and-tree. We believed that these generalised labels are worth exploring further.

In this section we have described PCR proof tree labels. In the next section we explain our approach for matching these proof tree labels for retrieval.

4.4 Why Matching Proof Tree Labels by Predicate Names?

In this section we come to the question of matching two proof tree labels to determine their level of similarity. An important question will be whether to consider just predicate names or arguments as well. We have adopted the former approach. This section attempts to justify this choice and to compare with other similarity matching approaches.

Figure 4.4 shows a very simple program. It has two clauses with the same head

```
parent(X,Y):-
            son(Y,X).
parent(X,Y):-
            daughter(Y,X).
son(Y,X):-
            child(Y, X),
            boy(Y).
daughter(Y,X):-
            child(Y, X),
            girl(Y).
child(sandra, peter).
child(john, peter).
child(nicholas, peter).
girl(sandra).
boy(john).
boy(nicholas).
```

Figure 4.4: A simple program for the relation parent

`parent/2` but different bodies. Here we want to illustrate that predicate names are especially important, rather than arguments, for retrieval purposes. For example, if we submit the query `<-parent(peter,Y)` to the program shown in Figure 4.4, Prolog will give three answers:

```
Y = john;
Y =nicholas;
Y = sandra
```

From the results users can judge that 'peter' is the parent of 'john', 'nicholas' and 'sandra', but they are not sure who is son and who is daughter.

Now suppose the system informs users of the predicate names consulted during the computation. The users can easily tell the difference among these three results; that is 'john' and 'nicholas' are sons and 'sandra' is a daughter. Figure 4.5(a) shows the (partial) or-tree of this query. We can see that 'john' and 'nicholas' initially follow the same path whereas 'sandra' follows a different path, which shows that they have used different definitions of the relation `parent/2`. The information about these definitions can be checked from the predicate names of the or-tree. So, an important lesson can be derived from this discussion - that predicate names tell about the definitions used for solving a particular query. Figure 4.5 Fig(b) shows the three success and-trees of the

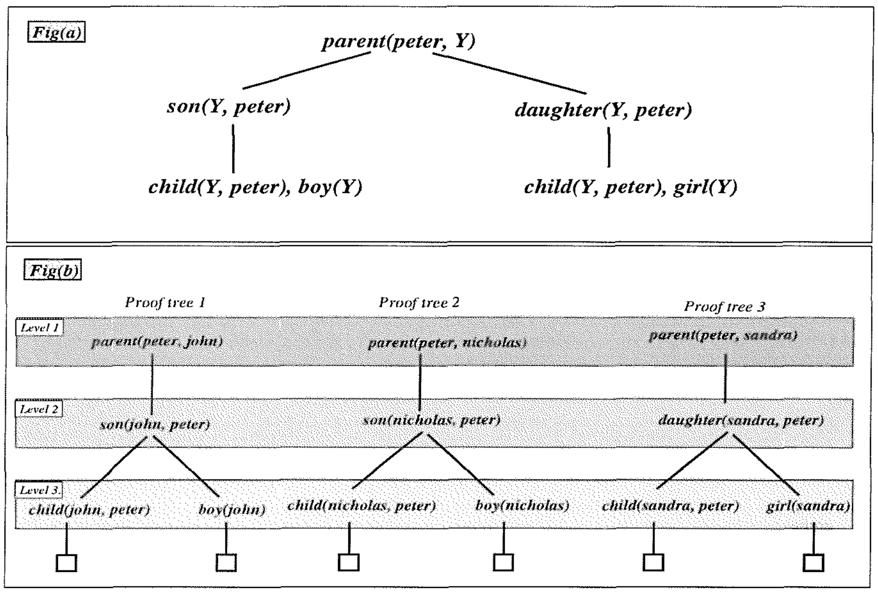


Figure 4.5: Proof trees of query parent(peter, Y)

or-tree. A feature of these proof trees is that at level 1 they all look the same as regards predicate name, in this case `parent`. Thus, each proof tree concerns information about the relationship 'parent'. But at level 2 the proof trees of 'john' and 'nicholas' look the same in the sense that they have the same predicate names and therefore concern the same relations, whereas 'sandra's proof tree looks different, as it has a different predicate name and concerns different relations. At this level, it can be seen clearly that 'proof tree 1' and 'proof tree 2' concern the same relations whereas 'proof tree 3' does not. Going deeper in the proof tree gives a more precise picture. This example shows the rationale behind matching the predicate names of proof trees of similar queries and using this to check the level of similarity about the information present in them. As PCR is designed for handling general information so we argue that it is practical in the first instance only to consider predicate names in the proof trees when finding similarity between the information stored in two proof trees. Giving so much importance to the predicates names does not mean that we deny the importance of the predicate arguments, but the value of these arguments ultimately affects the flow of the program and the flow of the program can be checked just from predicate names to a large extent.

Before explaining the matching algorithm, we want to explain the differences between PCR and other similarity finding approaches. Similarity finding approaches are commonly based on matching attributes of frames, as most proposed systems in the AI and Law literature appear to be frame based [Gentner & Landers 1989], [Kowalski 1991]. Gentner [Gentner & Landers 1989], [Gentner 1983] describe a matching strategy called *literal similarity*: all objects and their attributes match. All relations between objects are also matched. Holyoak and Thagard [Holyoak & Thagard 1989] call this scheme *structural consistency* and describe another similarity scheme called *semantic similarity* in which objects or their attributes are not exactly the same but similar, in that relations are matched. The paper [Gentner & Landers 1989] proposes another possibility called *analogy* which means that objects and relations between objects are matched, but few or no object attributes are matched. Yang et. al. [Yang *et al.* 1993] describe two other possibilities: *Implied similarity* - some objects are matched and some not, but the relations between objects are the same; *Embedded similarity* - only some extra attributes match. Other similar approaches are mentioned in [Holyoak & Thagard 1989], [Nitta *et al.* 1993].

The above approaches are quite natural as the authors have considered all possibilities which could occur in a frame based environment. The main differences between them and PCR are: PCR is based on proof tree matching, not frames matching; and PCR only considers predicate names not the arguments.

On the question of whether to consider argument values, an important issue is efficiency, in the sense of how fast and meaningful the retrieval is. Matching argument values puts unnecessary load on the system if we can retrieve the same information without matching arguments. In real world computer programs execution speed should not be ignored. If a program does not run fast it loses its applicability, especially in the case of information retrieval where users do not like slow programs. So the system developer must take account of execution speed.

Perhaps a more important point is that in daily life, for getting general information, users usually want similar rather than identical information regarding an issue. The reason is that they want to understand their situation by making comparisons with other similar situations. If they do want identical information, finding this using conventional database or simple pattern matching is probably the preferred technique. So our point is that predicate arguments are usually at too fine a grain, unless exact matches are required. We should also remember that finding information by just comparing predicate names in proof trees can retrieve information at various levels of similarity down to a very fine grain. We believe that for designing a general purpose information matching scheme just considering predicate matching is appropriate.

4.5 Weighing Scheme for Similarity Calculation Among Proof Trees

In Section 4.1, we mentioned that the proof tree labels matching algorithm generates a single number (*matching weight*) as a measure of similarity between two proof tree labels. For generating this number we design a weighing scheme which calculates the level of similarity (*matching weight*) between two proof trees. A rough description of the algorithm is given here, a precise definition being given in Section 4.6. Suppose we have two proof tree labels with the same root predicate. Their matching weight is computed by comparing nodes of the proof trees in a depth-first traversal. According to this scheme, if the predicate names of corresponding non-leaf nodes of two proof trees are the same then add 0.2 to the matching weight. If the predicate names of two corresponding leaf nodes and their truth values are the same then add 0.1; if they have different predicate names but both have truth value true then add 0.01; if they have the same predicate name but different truth values then add .001 to the matching weight. In all other cases add 0.0 to the matching weight. The particular values of these weights, in the ratio 200:100:10:1, have been chosen for convenience. However the basic idea of distinguishing between these sources of information in this order of importance seems a reasonable way to assess similarity of labels.

Taking four hypothetical cases, we now explain the practicality of our weighing scheme. We suppose that the first three hypothetical cases given below are labelled with their 'proof tree labels' and that the labels are available in the Label Base. The last case shows a 'proof tree label' for a user's request relevant to which he/she wants to see any relevant case report.

- Case 1.

  Miss. Jones was a citizen of a British Dependent Territory. She came to the UK in June1983 for an indefinite stay. She had a valid visa for her stay. She went back to her country in March 1987 and came back again to the UK in March, 1989. During her stay in the UK she was not in breach of the immigration laws. She submitted her application for British citizenship on January, 1991 and her application was turned down.

- Case 2.

  Mr. Shah was a citizen of a British Dependent Territory. In his country, he was working in a reasonable post in the Crown Service and showed outstanding performance. In addition to this he had very close connections with the UK and had some special, considerable circumstances. He submitted his application for British citizenship in January, 1984 and his application was accepted.

- Case 3.

  Mr. Green was an Overseas citizen. He came to the UK in 1985 for an indefinite stay with a valid visa. He submitted his application for British citizenship on January, 1992 and his application was accepted. Before submitting his application he was not outside the UK for more than 450 days in five years. In these five years he was also not at any time in breach of the immigration laws. He was also not outside the UK more than 90 days in the year before submitting his application.

- User Request.

  The user is a male Overseas citizen. He came to the UK in January 1988 with a valid visa. In the last five years, he did not spend more than 450 days outside the UK. He was also not outside the UK for more than 90 days in the last year. In the last five years he was not at any time in breach of the immigration laws. He wants to submit his application for British citizenship in January 1994. The intention of this search is to find cases relevant to his situation so that he can get some idea about his situation.

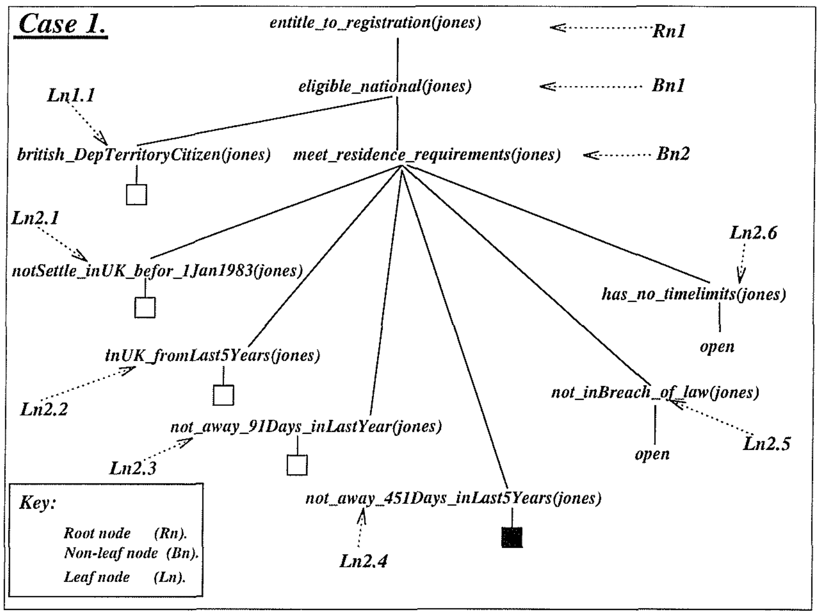

Case 1.
entitle_to_registration(jones)
Rn1
eligible_national(jones)
Bn1
Ln1.1
british_DepTerritoryCitizen(jones)
meet_residence_requirements(jones)
Bn2
Ln2.1
notSettle_inUK_befor_1Jan1983(jones)
Ln2.6
has_no_timelimits(jones)
open
inUK_fromLast5Years(jones)
Ln2.2
not_inBreach_of_law(jones)
open
Ln2.5
not_away_91Days_inLastYear(jones)
Ln2.3
not_away_451Days_inLast5Years(jones)
Ln2.4
Key:
Root node (Rn).
Non-leaf node (Bn).
Leaf node (Ln).

Figure 4.6: Proof tree label of Case 1

Assume proof trees for cases 1-3 and the user request have been generated, as shown in Figures 4.6, 4.7, 4.8, and 4.9 respectively. From these figures, it can be seen that the root nodes of all proof trees are the same. This shows that they are all concerned with the same concept. We match root nodes only to find the relevant proof trees from the Label Base and do not add any number to the matching weight.

We now return to the four example cases. For a successful exit from non-leaf node Bn1 (in Figures 4.6 to 4.9) many definitions of leaf nodes are available. For example, British Dependent Territory citizen, British Overseas citizen, so on (we call them eligible nationals). The definition of entitlement to registration (shown in Figure 2.2) says that an applicant must be an eligible national. The important thing

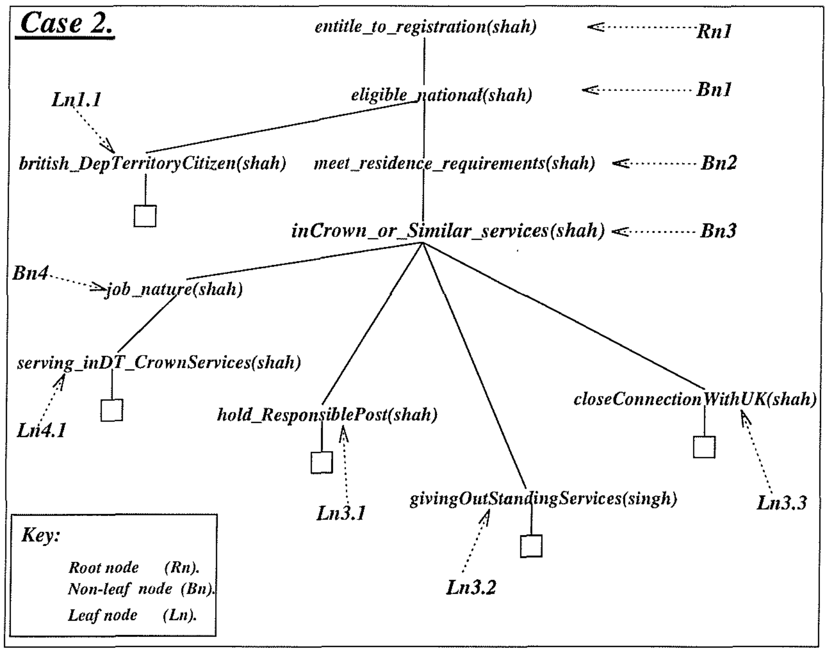

Case 2.
entitle_to_registration(shah)
Rn1
eligible_national(shah)
Bn1
Ln1.1
british_DepTerritoryCitizen(shah)
meet_residence_requirements(shah)
Bn2
inCrown_or_Similar_services(shah)
Bn3
Bn4
job_nature(shah)
serving_inDT_CrownServices(shah)
Ln4.1
hold_ResponsiblePost(shah)
Ln3.1
closeConnectionWithUK(shah)
Ln3.3
givingOutStandingServices(singh)
Ln3.2
Key:
Root node (Rn).
Non-leaf node (Bn).
Leaf node (Ln).

Figure 4.7: Proof tree label of Case 2

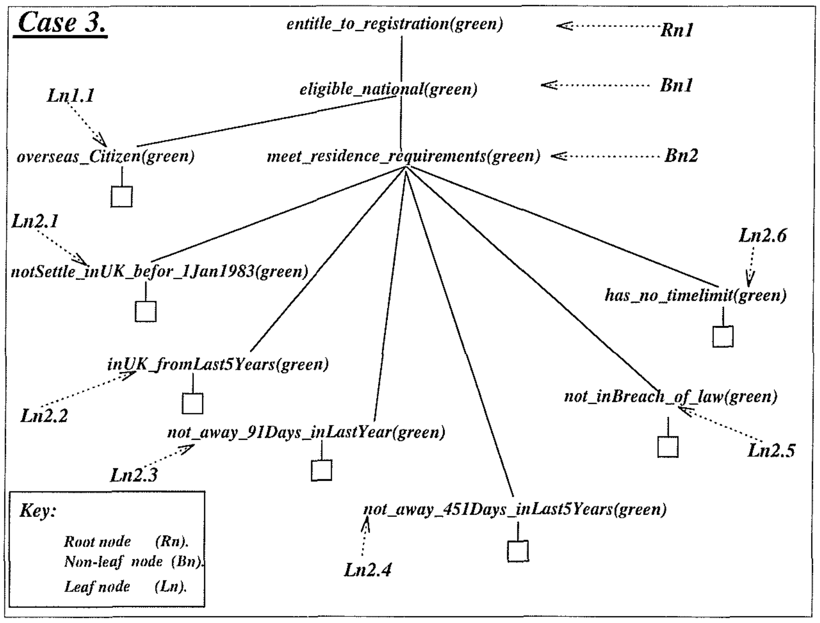

Case 3.
entitle_to_registration(green)
Rn1
eligible_national(green)
Bn1
Ln1.1
overseas_Citizen(green)
meet_residence_requirements(green)
Bn2
Ln2.1
notSettle_inUK_befor_1Jan1983(green)
Ln2.6
has_no_timelimit(green)
inUK_fromLast5Years(green)
Ln2.2
not_inBreach_of_law(green)
Ln2.5
not_away_91Days_inLastYear(green)
Ln2.3
not_away_451Days_inLast5Years(green)
Ln2.4
Key:
Root node (Rn).
Non-leaf node (Bn).
Leaf node (Ln).

Figure 4.8: Proof tree label of Case 3

here is the success of eligible national condition, not how it is satisfied. In our example the proof trees of Figures 4.6 and 4.7 successfully exit from Bn1 with a succeeding leaf `british_DepTerritoryCitizen/1`. Whereas, the proof trees of Figures 4.8 and 4.9 successfully exit from Bn1 with another succeeding leaf `overseas_Citizen/1`.

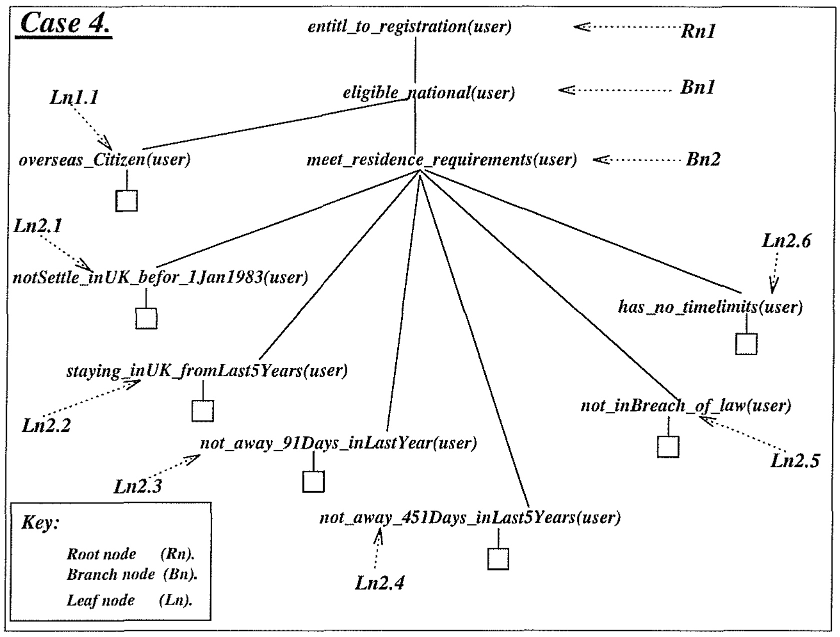

Case 4.
entitl_to_registration(user)
Rn1
eligible_national(user)
Bn1
Ln1.1
overseas_Citizen(user)
meet_residence_requirements(user)
Bn2
Ln2.1
notSettle_inUK_befor_1Jan1983(user)
Ln2.6
has_no_timelimits(user)
staying_inUK_fromLast5Years(user)
not_inBreach_of_law(user)
Ln2.2
not_away_91Days_inLastYear(user)
Ln2.5
Ln2.3
not_away_451Days_inLast5Years(user)
Ln2.4
Key:
Root node (Rn).
Branch node (Bn).
Leaf node (Ln).

Figure 4.9: Proof tree label of user's request

Now, as all proof trees successfully exit from Bn1, we say that they all are similar to each other up to this stage, but of course we can not say that they are the same. As, if only the non-leaf nodes is considered the system finds all four cases exactly the same whereas actually they are not as they satisfy different eligible national conditions. Without considering leaf nodes we can not find the exact level of similarity between two proof trees.

When matching leaf nodes related to a particular non-leaf node in two proof trees we add a different numeric value to the matching weight as leaf nodes can appear differently in two proof trees. This is further discussed below.

- Add 0.1 to the matching weight if the predicate names and the truth values of the corresponding leaf nodes in both proof trees are the same. For example, in Figure 4.8 and 4.9, the leaf node Ln1.1 has same predicate name (`overseas_Citizen/1`) and same truth value (true). Hence, both proof trees have the same kind of information.

- Add 0.01 to the matching weight if the truth values of the corresponding leaf nodes in both proof trees are same but they have different predicate names. For example, in Figure 4.6 and Figure 4.9 the leaf node Ln1.1 has same truth value, 'true', but has different predicate names; british_DepTerritoryCitizen and overseas_Citizen. This difference does not affect the success of the non-leaf node Bn.1. Also, the proof trees cover different aspects of a particular situation, and so we believe this information can be useful for users.

- Add 0.001 to the matching weight if the predicate names of the corresponding leaf nodes in both proof trees are same but they have different truth values. For example, in Figure 4.6 and Figure 4.9 the leaf node Ln2.4 has the same predicate name, not_away_451Days_inLast5Years, but has different truth values. In the first case it is 'true' and in latter case it is 'fail'. In this situation, the information in both proof trees is different, however, they concern two different aspects of a particular situation. We believe the information can also be useful to the user, but less so.

- Add 0.0 to the matching weight in all other cases. For example, in Figure 4.6 Bn2 leads to a bunch of leaf nodes. Whereas, in Figure 4.7 Bn2 leads to another non-leaf node (Bn3). This shows that Bn2 uses different definitions in the proof trees so the information is not relevant.

The basic idea of our scheme is that the system gives higher weight to non-leaf nodes because these nodes provide information about the succeeding path of the proof tree. However leaf nodes also provide important information (but less important) so we consider them too. We consider two documents similar if their 'proof tree labels' are exactly the same: that means their root nodes, all non-leaf nodes, and leaf node have same predicate names and same truth values. Recall that a proof tree is a branch of an or-tree, represents an and-tree. Two and trees can only follow the same path if they have the same factual situation. Hence it can be argued that documents which have been labelled with the same proof tree labels have the same kind of information. If the proof trees are not the same then the level of similarity depends on the truth values and predicate names of leaf nodes and the number of similar predicate names of non-leaf nodes in the two proof tree labels. With the help of this scheme the system is not only able to find documents having the same kind of information but also documents that have various degrees of relevant information. Now we give a precise statement of our proof tree matching algorithm.

```
⋆ Start
⋆ Step1- Set **Weight** = 0.0
⋆ Step2-
    • $IF'$ The root nodes of Qpt and the Lpt are not the same
    • $THEN'$
        $IF''$ a sub-tree of Lpt has the same root node as Qpt OR a sub-tree of Qpt has the same root node
            as Lpt.
        $THEN''$ Pick the sub-tree of the bigger proof tree label for matching and go to $ELSE'$
        $ELSE''$ Weight = 0.0 and goto Step3
    • $ELSE'$
        ◇ MCond1- Match nodes as described below:
            – BNode: IF the current nodes are both leaf nodes THEN goto LNode ELSE check BCond1
                ∗ BCond1: IF the current nodes are both non-leaf nodes with same predicate names
                  THEN add 0.2 to Weight and goto MCond2 ELSE check BCon2.
                ∗ BCond2: IF The current nodes are not both non-leaf nodes or have different predicate
                  names THEN add 0.0 to Weight and goto Step3 .
            – LNode: IF current nodes are both leaf nodes THEN try LCond1, LCond2 or LCond3
              and goto MCond2 :
                ∗ LCond1: IF in both proof trees the current node has truth value 'true' THEN match
                  their predicate names as below:
                    · If they are same in both proof trees then add 0.1 to Weight.
                    · If they are different in both proof trees then add 0.01 to Weight.
                ∗ LCond2: IF in both proof trees the current node has truth value 'fail' then match
                  their predicate names as below:
                    · If they are same in both proof trees then add 0.1 to Weight.
                    · If they are different in both proof trees then add 0.0 to Weight.
                ∗ Lcond3: IF in both proof trees the current node has different truth value THEN
                  match their predicates names as below:
                    · If they are same in both proof trees then add 0.001 to Weight.
                    · If they are different in both proof trees then add 0.0 to Weight.
        ◇ MCond2- IF either of current nodes is the last node THEN goto Step3 ELSE increment current
          nodes and goto MCond1.
⋆ Step3- Save Weight and Label in a list
⋆ Step4- Pick the next Lpt if there is one and goto Start ELSE goto End
⋆ End.
```

Figure 4.10: Matching algorithm

4.6 Proof Tree Matching Algorithm

On the basis of our informal comments on matching and weighing schemes for proof tree labels, described in Section 4.4 and 4.5, we have developed a matching algorithm. The algorithm is precisely stated in Figure 4.10. It uses a depth-first, left-to-right search. The search stops if the predicate names of corresponding non-leaf nodes are different in the two proof tree labels being matched or if one of the nodes is a leaf node. The algorithm generates a single number, matching weight (MW), as a measure of similarity between two proof tree labels. This measure has sufficient information to implement the ideas discussed in the preceding sections.

The algorithm starts with a proof tree label of the user's request, Qpt, and a proof tree label from the Label Base, Lpt, and computes their MW. In practice a list of MW's is computed for a given Qpt, one for each matching Lpt in the Label Base. A list of calculated MW's and corresponding document names is produced of the form

```
[(MW1, DocumentName1, ProofTree1), (MW2, DocumentName2,
ProofTree2), ......  (MWn ,DocumentNameN, ProofTreeN) ]
```

4.7 Calculation of Percentage Matching Weight

The matching weight (MW) gives the level of similarity between two proof tree labels. After calculating the MW, the system calculates a percentage matching weight (PMW) as follows. Calculate the MW of the query proof tree label, Qpt, with itself giving the maximum matching weight (MMW). The PMW between the retrieved proof tree label, Lpt, and Qpt is then

$$PWM = (MW/MMW) * 100$$

where MW is the matching weight for Qpt and Lpt.

After calculating the PMW, the system displays the names of retrieved documents and their PMWs in rank order. Displaying information about retrieved documents in this way gives some guidance to users about the degree of relevancy of the retrieved documents to the requested information and, hence, makes selection more effective.

Finally we want to mention that our scheme for assigning weights when comparing the nodes of proof tree is to some extent arbitrary. The main idea behind assigning weights is not to count the number of matching nodes but to distinguish different conditions of the proof tree nodes. The system uses this weighing scheme to calculate the MMW and PMW so as to give a guidance to users about retrieved documents. We do not claim that our weighing scheme for MW and PMW produces the best results. It is an open issue which needs more exploration. Nevertheless we do feel confident that any reasonable scheme should distinguish between different conditions of proof tree nodes as our does. In future we want to explore this issue further.

4.8 Conclusion

In this chapter we have described an LP based, semi-automatic technique - **P**roof tree based **C**onceptual **R**etrieval (PCR)- for conceptual dynamic linking in hypermedia systems. PCR has been implemented using four important features of LP: Horn clauses for the knowledge base containing the dictionary of general concepts; meta-level programming ; QTU for extracting conceptual information from user requests and documents; and proof trees for storing conceptual information about the destination anchors within the bodies of links. Dynamic linking approaches based on conventional retrieval techniques introduce serious irrelevant retrieval problems. In our approach, we do not use the actual linking structure of the document or Boolean search. Instead we analyse the contents of documents using a kbs technique and tag conceptual information about documents within the body of links. This conceptual enriching of the links offers the possibility of reducing irrelevant retrieval.

In PCR proof tree labels are used as a conceptual representation of a document's contents. Matching these labels, PCR retrieves documents which contain relevant information. We have defined a matching algorithm. Matching predicate names of proof tree nodes and using a natural, but to some extent arbitrary, weighing scheme, the algorithm generates a single number, the matching weight, as a measure of similarity between proof tree labels. The weighing scheme gives high weight to non-leaf nodes since these nodes actually provide information about the succeeding path of the proof tree. However leaf nodes also provide important information so the algorithm also consider them but with a lower weight.

PCR considers two documents to have maximum similarity if their proof trees exactly match: that means corresponding nodes, (their root nodes, all non-leaf nodes and leaf nodes) have the same predicate names and same truth values. Actually, in PCR a proof tree label is a branch of the SLD search tree represented as an and-tree. Two proof trees can only follow the same path if they have the same factual situation. Hence it can be argued that documents which have been labelled with exactly the same proof tree labels have similar kinds of information. If they have some lesser level of similarity then the level of similarity depends on the number of similar non-leaf nodes and leaf nodes of the two proof trees. An important feature of PCR is that it finds documents with varying degrees of relevant information. In the next chapter we describe how PCR is applied to hypermedia.

5 Applications of PCR to Hypermedia

5.1 Introduction

In the previous chapter we introduced and defined Proof tree based Conceptual Retrieval (PCR). We now describe how PCR is applied to hypermedia systems. There are two problems related to the use of PCR in hypermedia systems which arise immediately. The first is how to inform users which definitions are available in the kb. Without knowing this it is very difficult for users to run appropriate queries so as to link new documents. The second is how to inform users which concepts from a document are linked. Section 5.2 explains how we handle these two problems in our prototype system.

Probably the most important contribution of PCR to hypermedia is to offer conceptual dynamic linking. This is done by storing conceptual information about destination anchors within the bodies of links so that during the linking process the system will link those documents which are conceptually similar/relevant to a user's request. In Section 5.3, through an illustrative example, we demonstrate how PCR offers conceptual dynamic linking in hypermedia systems. In this section we also describe how our system provides active document features.

Providing a facility for conventional database retrieval would not normally be claimed as a big achievement, but combining such structured retrieval with PCR brings together two complementary retrieval ideas and appears to produce a considerably more effective system. Section 5.4 describes this combination, called enhanced database retrieval.

The main aim of this research is to exploit kbs techniques for solving hypermedia problems. Along with the *kbh* framework, PCR is the main outcome of this research. In Section 5.5, we expand on the advantages of PCR for solving hypermedia problems. Section 5.6 gives a concluding discussion about PCR and mentions some issues which will inform the direction of future research.

5.2 Informing Users about Available Queries and Linked Concepts

In this section first we describe how our system informs users which definitions are available in the kb. In our prototype, when a user wants to link a new document with a particular concept, the system displays an index of the definitions available in the kb. Two methods were considered for building this index. The first method was to display in an alphabetical order the heads of all clauses available in the kb and allow users to select one. The problem with this method is that it builds a long list of available clauses which could confuse users. Also there may be some clauses which do not make any sense to run as a query. The second method was to display only those clauses that have been declared as top-level queries and allow users to select one of them. The problem with this method is that it limits users to running only those queries that have been declared as top-level. Our prototype system uses a combination of these methods.

In this combined method the system initially displays an index of top level queries. The system provides an option whose function is to display the clause bodies of the selected top-level query. Users can then expand any predicate in these bodies by selecting it. Repeating this process can find any query related to their top-level selection. If users want to come back to a previous query they can do so. (Demonstrated in Section 8.3.)

Now we discuss how our system informs users about linked concepts from a document. In typical hypermedia systems when a link exists the system gives some clue of its presence. Some systems do this by changing the colour of the source anchor, others by changing the cursor shape. A most important issue in our prototype system is how users will know about the anchors that have been linked with relevant documents. One option is to adopt the conventional method (e.g. of changing the colour of linked concepts). The advantage of this method is that it gives a clear idea about the linked concepts, but there are two limitations. Firstly, users can only follow links from the particular document from where the link was created. Secondly, if the same concept appears in another document with entirely different wording users can not find the relevant information, even though, the information is available within the system. It does not seem reasonable that information is available within the system and users can not use it, but this is the behaviour found in typical hypermedia systems. The method used in our prototype system is more general and flexible.

The system does not provide any visible clue about linked concepts, but users can select any word, phrase, line, paragraph or whole document. The system then finds all concepts which have been linked with kb queries for this selection. The strategy the system uses for this purpose is discussed below.

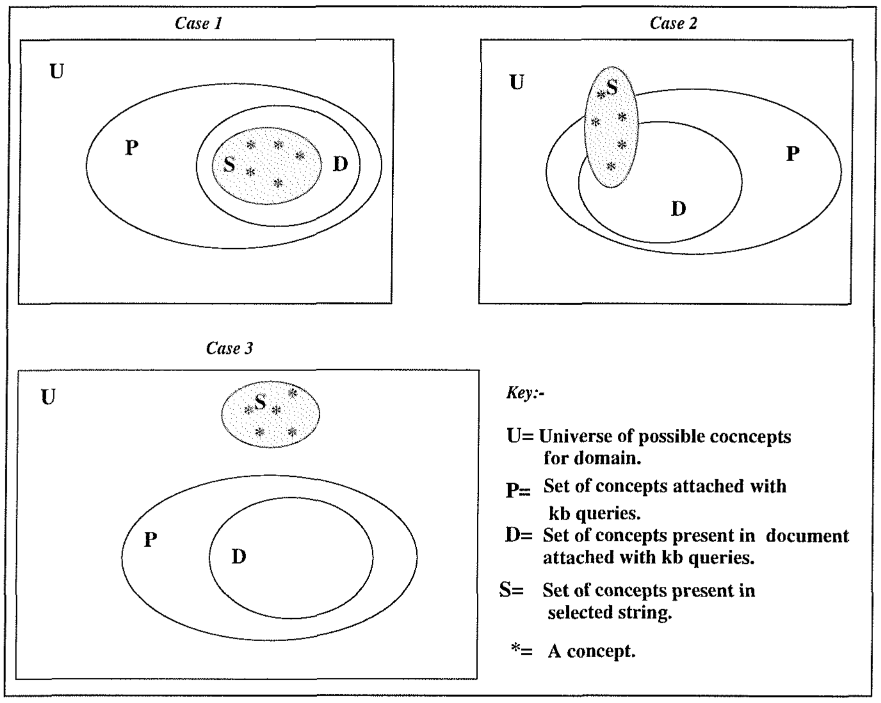

Case 1
U
P
S
D
Case 2
U
S
P
D
Case 3
U
S
P
D
Key:-
U= Universe of possible cocncepts for domain.
P= Set of concepts attached with kb queries.
D= Set of concepts present in document attached with kb queries.
S= Set of concepts present in selected string.
*= A concept.

Figure 5.1: Concept extracting schemes from selected string

Figure 5.1 shows Venn diagrams for three possible cases concerning the set of concepts S in a selected string. In these diagrams, D is the set of formalised concepts in the document containing the selected string; that is concepts attached to queries in the kb. P is the set of all formalised concepts from all other documents as well as the selected one. U is the set of all concepts, formalised or not.

In **Case 1** all concepts in the selected string appear in the document in which the selection is made and are attached to the kb queries. Then

$$S \subseteq (D \cap P \cap U)$$

If the selected string has only one such concept, S is a singleton, the system will simply send the related query to the kb for evaluation. If the selected string has more than one concept the system will display all members of the set S for selection. Users can select any one of them and system will send its related query to the kb.

In **Case 2** some concepts of S appear in the document attached to kb queries, others are attached to kb queries in different documents, and others still have not been formalised. The set S divides into three types of concepts.

1. $S \cap D$

   These concepts are dealt with as in case 1.

2. $S \cap P - S \cap D$

   These concepts are displayed. They have attached kb queries in some other documents. The user can choose to link them to the current document. This gives an important form of generic linking.

3. $S \cap U - S \cap P$

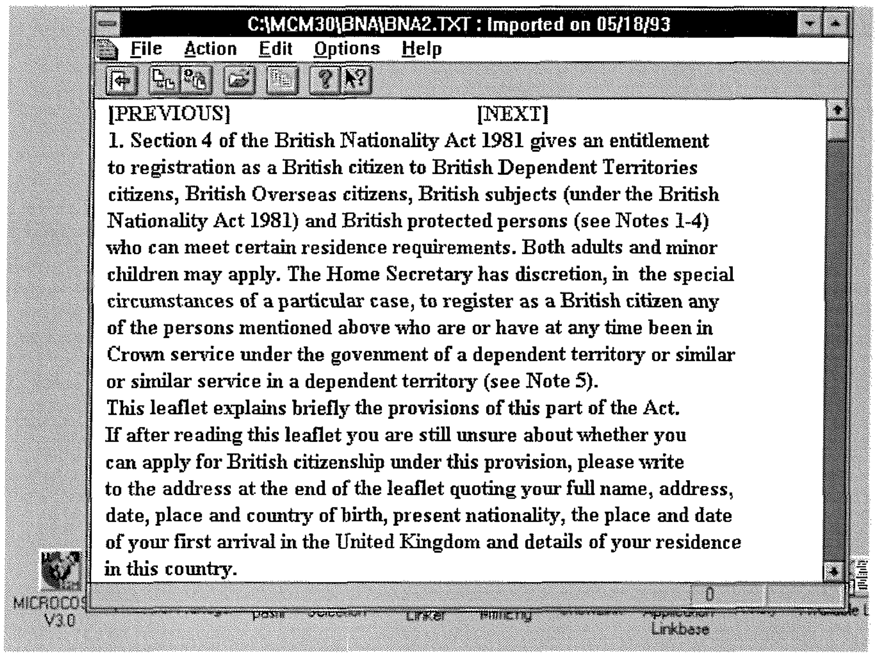
   These concepts are not formalised and therefore can not be displayed.

Case 3 is very obvious; the selected string does not have any formalised concepts (attached to a kb query) so the system will only display a message that it does not have any information.

We now go on to discuss the purpose of users sending their queries to the kbs. Essentially there are two purposes: to evaluate the query which is the active document feature; and to retrieve documents with label matching which is the conceptual dynamic linking feature.

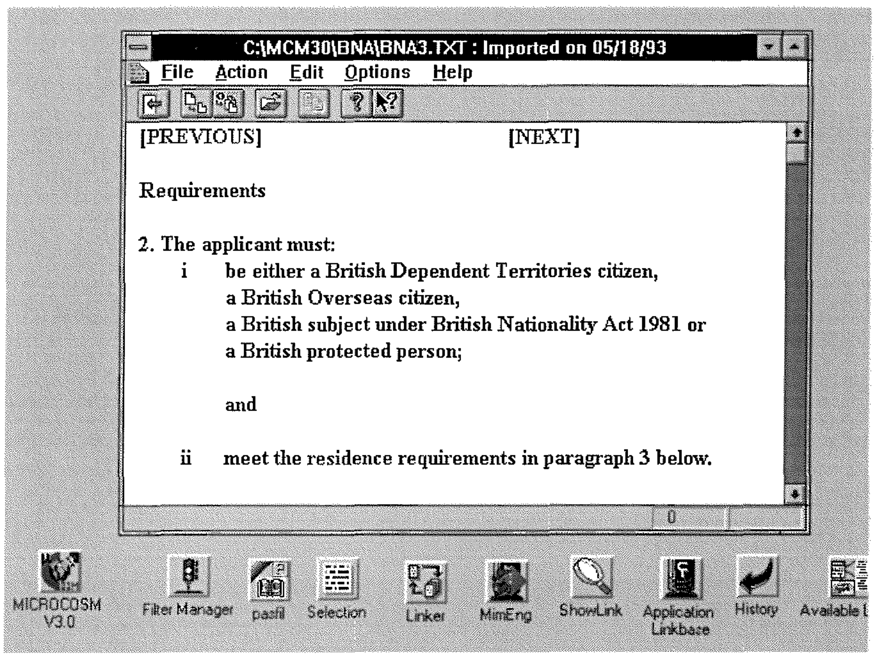
5.3 Conceptual Dynamic Linking and Active Document Features

For conceptual dynamic linking it is important to store conceptual information about destination anchors within the bodies of links so that during the linking process the system links those documents which are conceptually similar to a user's request. The main problems are, as mentioned in [Tomek & Murer 1992], how to extract conceptual information from destination anchors? How to store this information within the body of links? We now explain the PCR approach to handling these problems.


C:\MCM30\BNA\BNA2.TXT : Imported on 05/18/93
File Action Edit Options Help
MICROCOSM V3.0
Linkbase

[PREVIOUS] [NEXT]
1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories citizens, British Overseas citizens, British subjects (under the British Nationality Act 1981) and British protected persons (see Notes 1-4) who can meet certain residence requirements. Both adults and minor children may apply. The Home Secretary has discretion, in the special circumstances of a particular case, to register as a British citizen any of the persons mentioned above who are or have at any time been in Crown service under the government of a dependent territory or similar or similar service in a dependent territory (see Note 5).
This leaflet explains briefly the provisions of this part of the Act.
If after reading this leaflet you are still unsure about whether you can apply for British citizenship under this provision, please write to the address at the end of the leaflet quoting your full name, address, date, place and country of birth, present nationality, the place and date of your first arrival in the United Kingdom and details of your residence in this country.

Figure 5.2: A text showing the first paragraph of the leaflet.



[PREVIOUS] [NEXT]

Requirements

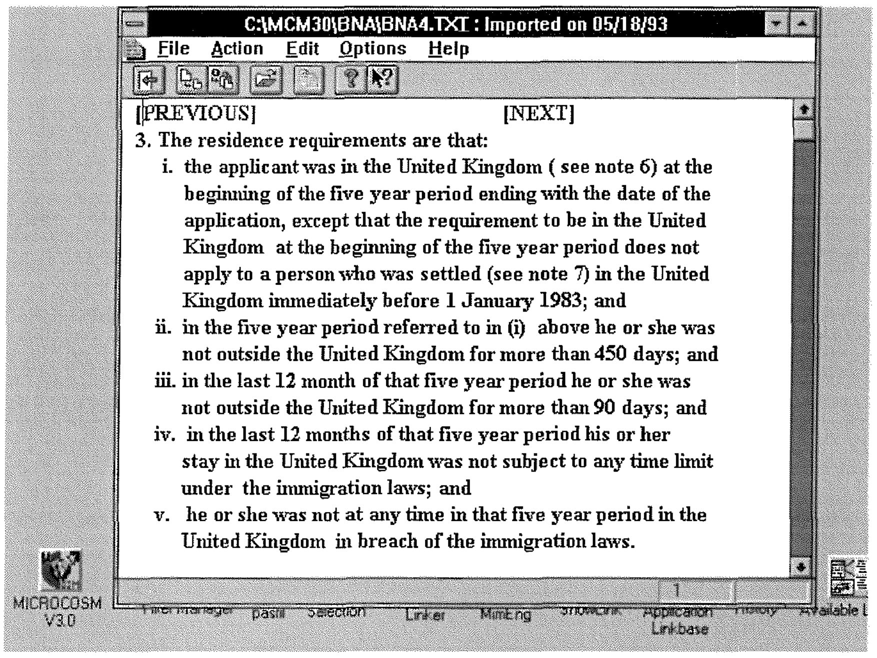
2. The applicant must:
i be either a British Dependent Territories citizen,
a British Overseas citizen,
a British subject under British Nationality Act 1981 or
a British protected person;

and

ii meet the residence requirements in paragraph 3 below.

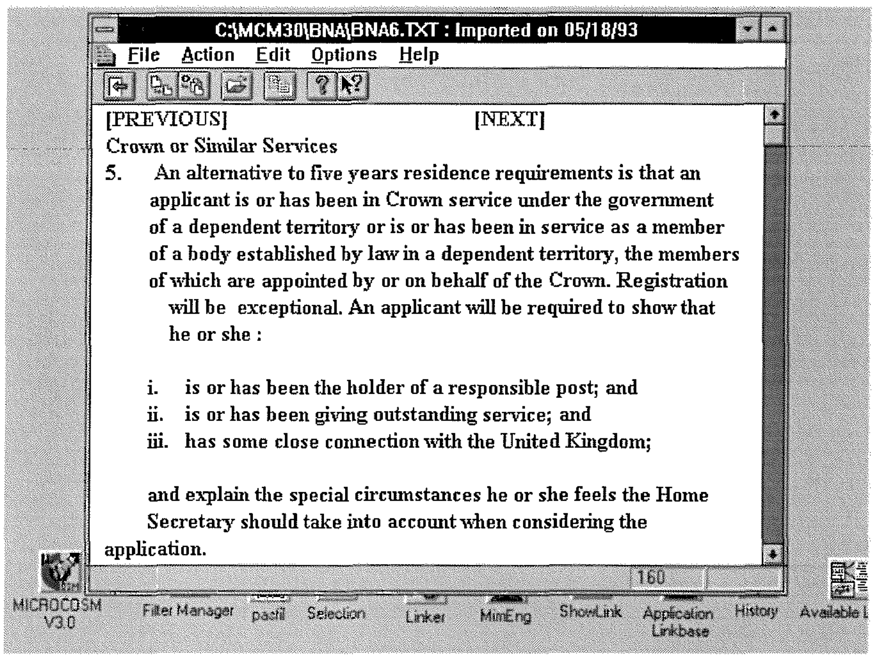
Figure 5.3: A text showing the requirements for registration.

Figures 5.2, 5.3, 5.4 and 5.5 show the text of a leaflet giving information about


C:\MCM30\BNA\BNA4.TXT : Imported on 05/18/93
File Action Edit Options Help
MICROCOSM V3.0
pasti
Selection
Linker
MimEng
Application Linkbase
Available L

[PREVIOUS] [NEXT]
3. The residence requirements are that:
i. the applicant was in the United Kingdom (see note 6) at the beginning of the five year period ending with the date of the application, except that the requirement to be in the United Kingdom at the beginning of the five year period does not apply to a person who was settled (see note 7) in the United Kingdom immediately before 1 January 1983; and
ii. in the five year period referred to in (i) above he or she was not outside the United Kingdom for more than 450 days; and
iii. in the last 12 month of that five year period he or she was not outside the United Kingdom for more than 90 days; and
iv. in the last 12 months of that five year period his or her stay in the United Kingdom was not subject to any time limit under the immigration laws; and
v. he or she was not at any time in that five year period in the United Kingdom in breach of the immigration laws.

Figure 5.4: A text showing the residence requirements.



[PREVIOUS] [NEXT]
Crown or Similar Services
5. An alternative to five years residence requirements is that an applicant is or has been in Crown service under the government of a dependent territory or is or has been in service as a member of a body established by law in a dependent territory, the members of which are appointed by or on behalf of the Crown. Registration will be exceptional. An applicant will be required to show that he or she :

i. is or has been the holder of a responsible post; and
ii. is or has been giving outstanding service; and
iii. has some close connection with the United Kingdom;

and explain the special circumstances he or she feels the Home Secretary should take into account when considering the application.

Figure 5.5: A text showing the Crown or similar services requirements

persons who are entitled for registration as British citizens [Home Office 1991c]. In Figure 5.2, there are many general concepts like *entitlement to registration, British Overseas citizen, residence requirements, Crown service, etc.* The definitions of these concepts are given in different parts of the leaflet. Taking *entitlement to registration* as an example, Figure 5.2 explains that persons who want to apply for registration must satisfy the conditions shown in Figure 5.3. Figure 5.3 shows that the applicants must satisfy the requirements 2(i) and 2(ii), as they are connected through a logical *AND* connective. The logical *OR* connective in requirement 2(i), however, categorises applicants into four disjoint groups. In Figure 5.5 the phrase *alternative to five years residence requirements* further categorises them into those who are/have been in Crown service and those who are/have not been in Crown service. People from these categories need to satisfy different conditions shown in Figure 5.4 and Figure 5.5.

This example shows that a single concept *entitlement to registration* can break down into many different definitions. Correspondingly, documents relate to this concept in many different ways. For example, one document might have information about a person who is a British Overseas citizen and satisfies the residence requirements. Another document might have information about a person who is a British Dependent Territories citizen and is a Crown servant.

From the text of Figures 5.2, 5.3, 5.4 and 5.5 we can see that the definition of the concept *entitlement to registration* can be formalised in Horn clauses as shown in Figure 5.6[1].

Let us suppose these definitions are available in the kb and therefore users can run them as logical queries. For example, suppose a user wants to link a document having information about a British Overseas citizen who satisfies the residence requirements with the concept *entitlement to registration*. The user will run the query `entitle_to_registration(Client)` in the context of the kb (a fragment of which is shown in Figure 5.6) related to this particular concept. The system will try to satisfy the first definition, about British Dependent Territories citizen, and ask the user about the client's nationality by posing the question

british_Dependent_Territories_citizen(client)?

Certainly, the answer will be 'no' as the document is not about a British Overseas person so this category of the definition will fail. The system will next try the definition about

[1]In this example our intention is not to show the best method of knowledge engineering; the idea is to explain the different aspects of PCR. Also we expect from the reader that issues of open texture should not be considered here. It is discussed in later sections.

```
entitle_to_registration(Client):-
            eligible_national(Client),
            meet_residence_requirements(Client).
entitle_to_registration(Client):-
            eligible_national(Client),
            in_Crown_or_similar_services(Client).
entitle_to_registration(Client):-
            eligible_national(Client),
            not_in_Crown_or_similar_services(Client),
            in_the_UK_for_the_Last_5_Years(Client),
            not_in_breach_of_the_Immigration_Laws(Client),
            has_special_circumstances(Client).
eligible_national(Client):-
            british_Dependent_Territories_citizen(Client).
eligible_national(Client):-
            british_Overseas_citizen(Client).
eligible_national(Client):-
            british_subject_under_BNA_1981(Client).
eligible_national(Client):-
            british_protected_person(Client).
meet_residence_requirements(Client):-
            not_settled_in_the_UK_before_1Jan1983(Client),
            in_the_UK_for_the_Last_5_Years(Client),
            not_away_91_days_in_the_last_year(Client),
            not_away_451_days_in_the_Last_5_Years(Client),
            not_in_breach_of_the_Immigration_laws(Client),
            not_subject_to_any_time_limits(Client).
in_Crown_or_similar_services(Client):-
            job_nature(Client),
            satisfy_special_conditions(Client).
job_nature(Client):-
            serving_in_DT_Crown_services(Client).
job_nature(Client):-
            appointed_by_or_on_behalf_of_Crown(Client),
            serving_a_body_established_by_law_in_DT(Client).
satisfy_special_conditions(Client):-
            holder_of_a_responsible_post(Client),
            giving_outstanding_services(Client),
            has_close_connection_with_the_UK(Client).
```

Figure 5.6: Fragment of Formalisation of concept entitlement to registration

British Overseas citizens. Certainly, the user's answer will be 'yes'. With this answer the system will get information about the document, that it is about a British Overseas citizen. The system will proceed in a similar way and extract other information available in the document. At the end of this process the extracted information is available in the kb.

The next problem is to build the body of a link between the selected concept and the document from where the information has been extracted. The main function of a link is to make association between source and destination anchors. In this example the source anchor is *entitlement to registration* and the destination anchor is a document. Our link has three kinds of information: text of the source anchor, name of the destination anchor and conceptual information about the contents of the destination anchor which is stored in a proof tree label. In our example the proof tree label has been generated by the dialogue just given above. The use of a proof tree for this purpose has two advantages: firstly, building proof trees in LP is not a difficult task; secondly, the proof tree contains information about which clauses are used to satisfy a definition, and hence it conveys conceptual information. This particular feature allows the system to split linked documents according to their conceptual contents which makes retrieval more effective and gives a solution to the problem of handling complex information.

In our example it can be seen that a proof tree of an evaluated query can give us an idea that the newly linked document contains information about a person who is a British Overseas citizen who satisfies the residence requirements. If the information in another document satisfies another definition of the same concept then its proof tree will be different. Thus tagging proof trees[2] within the bodies of links conceptually enriches them and makes them potentially more effective in sense that the chance of irrelevant retrieval may be reduced.

Another important point to mention here is that once a concept is attached to a kb query, users can run it as a logical query just to see how that particular concept applies in specific circumstances. In this case the system will evaluate the query and

[2]We want to make it clear that we are not considering other approaches which have been proposed to improve the functionalities of links, like adding date, version, heading, etc. These techniques do not change the basic nature of a link, whereas our technique does. Tagging conceptual information as proof trees within the body of links is novel, and should not be confused with other methods of adding information within the body of links.

display the results from kbs shell user interface. For example, in the above example the user can see whether he/she is entitled for registration or not. This is a powerful active document feature. It allows users to make the practical use of the available information within hypermedia documents. Through this feature, the system offers a better information understanding environment compared to hypermedia and kbs system individually.

5.4 Enhanced Database Retrieval Facility for Hypermedia

Users of typical hypermedia systems can only submit a request by selecting data from a displayed document. Sometimes users simply want to retrieve information according to particular factual information; for example, to retrieve a case report with given case title, case date, court name. Hypermedia systems (without IR facilities) do not allow this. In *kbh* we provide an option for this kind of retrieval, explained in this section.

Database retrieval techniques are commonly used to handle structured information, typically records having different fields. Database systems are effective because users can easily retrieve structured information according to one or more particular fields. For example, a user can see all cases that have given case date and findings. In this case, he/she will submit a query declaring Case Date & Findings as a key and the system will display all those cases that have matching case date and similar findings. If users want to see all those cases that have the same case date or all those cases that have the same findings, they will use the logical OR in their request and the system will retrieve both kinds of case. The use of databases techniques is very effective up to this extent, however, they have some problems such as:

- They only handle structured data.
- The retrieval is totally dependent on the structure of user requests.
- They do not retrieve other relevant information. For example, a user only knows the case number of a particular case and he/she wants to see other relevant cases. In typical database systems the user first finds the record for that particular case report by telling the system the case number and then using this retrieved information, submits other queries to try to find other relevant cases. The nature

of the future retrieved information totally depends on the structure of the new query.

PCR combined with conventional database retrieval techniques can handle these kinds of problem. The combination has advantages for both databases and hypermedia. On the database retrieval side it provides a conceptual retrieval environment which allows retrieval of relevant information semi-automatically. For example, by just giving the case number, the system can retrieve as well cases which are conceptually relevant to this particular case. Thus it allows handling of unstructured information in database systems. On the hypermedia side, it provides an enhanced database retrieval facility which certainly increases the strength of hypermedia systems because it does not ignore easily available and useful structured information - such as title, date and so on. We explain below how both techniques work together in a complementary fashion in *kbh*.

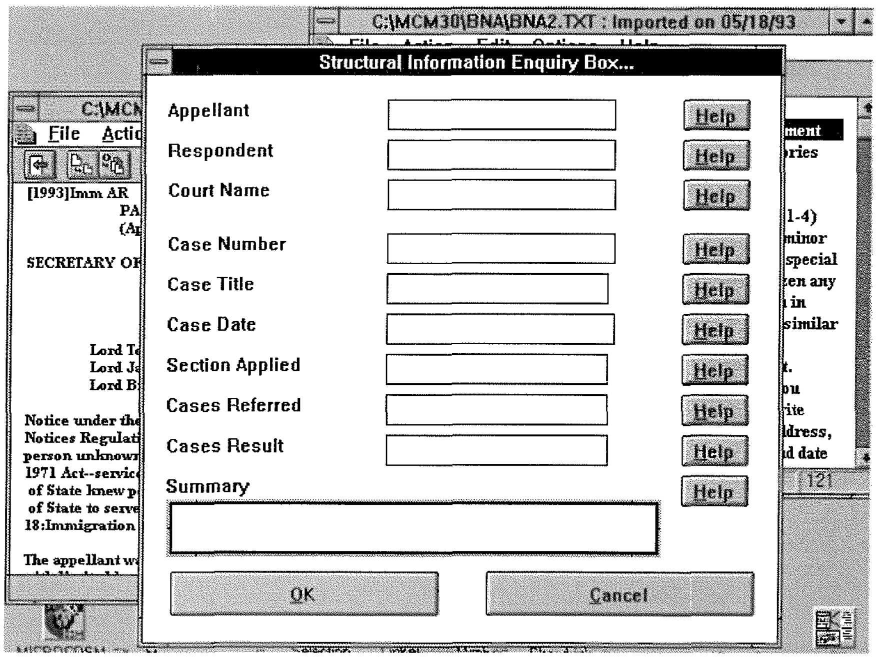


Figure 5.7: Window for getting structured information of case reports

For the retrieval of documents according to structured information, it is essential that the system acquires structured information from documents. For this purpose, during new document entry, the system displays a dialog box (shown in Figure 5.7) for requesting structured information about a newly entered document, in this example a case report having title, court name and so on. The motivation behind the development of this window were:

- To allow users to retrieve documents according to a particular factual situation. Retrieval through PCR leads users to a dialogue session. Sometimes users just want to retrieve documents according to the particular factual information, such as title, author, and so on.
- To extract factual information about a document which can be shown to users to give a general idea about the document. This information along with the proof tree label can be used to generate a summary of the documents. The system generates this summary automatically. This summary has enough information to help users during navigation. Detailed discussion about summary generation is given in Section 8.5.2.
- To provide an easy way for getting such information about documents. This form filling technique is effective and useful.

The system stores information acquired through this dialog box along with the conceptual information of the document in the Label Base in the form;

```
    labellingInfo(_, _, StructuredInfo, ConceptualInfo, _).
```

Here the argument `ConceptualInfo` contains the proof tree label and the argument `StructuredInfo` contains structured information. In this example, for a case report, the variable `StructuredInfo` will have information of the form

```
caseInfo(
    appellant(Name), respondent(Name), courtName(Court),
    caseNumber(Number), caseTitle(Title), caseDate(Date),
    sectionApp(Sections),refCases(References),
    caseResult(Result), caseSum(Summary)).
```

Once this information is stored into the Label Base, it is available for conceptual and structured retrieval.

Users can ask the system for structured retrieval through a provided option. If users do so, the system displays an editable window through which users can enter a search key. Given a search key, the system uses a simple pattern matching technique to search the Label Base for matching entries. If the system finds a matching entry, it picks the proof tree label of this entry and starts matching it with other proof tree labels in the Label Base. By matching proof tree labels it finds other relevant documents. So the system automatically retrieves documents relevant to that particular document. This shows the potential of PCR when combined with data base techniques. We believe this feature will be very useful for law professionals who, for example, sometimes wish to retrieve relevant cases with respect to particular factual information. A practical demonstration of enhanced database retrieval is given in Section 8.7.

5.5 Advantages of PCR Applied to Hypermedia

In this section we discuss the advantages of PCR. Some of the more important are mentioned below:

- PCR is based on LP and offers the full strength of LP in a hypermedia environment.
- PCR is a kbs technique and allows conceptual dynamic linking in hypermedia systems. The linking is dynamic because it is computed by the kbs. It is conceptual because the kbs encodes conceptual information concerning linked documents, contrasting with dynamic linking techniques based on conventional retrieval techniques which tend to cause irrelevant retrieval. Techniques for finding documents relevant to an anchor, such as word search, ranking them by number of links or number of visits, filtering [Tomek & Murer 1992] techniques are not very effective. For example one document may have exactly that information which a user wants to see but that document has been visited only once, so the system will not retrieve it using these methods. The basic reason is that these methods rely on statistics rather than specific conceptual content.
- PCR conceptually enriches hypermedia links since, through PCR, they acquire conceptual information about their linked documents. The use of proof trees

within the bodies of links offers an effective method to attach this conceptual information and offers a solution to the problem of how and what information should be attached to hypermedia links so as to take them beyond simple GOTO's.

- Introducing LP strength in hypermedia has many advantages. A single definition of a concept in the kb can be used for a range of purposes, for example, for information labelling, information retrieval, active document features.

- One particular concept can have many definitions according to its different aspects. Formalising all possible definitions in the kb (as Horn clauses) and using PCR for linking and retrieval purposes makes it is possible to split documents according to different points of views. This feature offers a solution to the problem mentioned in Section 2.5 about linking different case reports to a single concept[3].

- A document linked with a top-level concept can be retrieved from its lower level concepts. For example a document labelled with a concept *entitlement to registration* has also information about the concept *meets residence requirements*. In typical hypermedia systems users would need to make links between available documents and both concepts explicitly. With PCR documents linked with *entitlement to registration* can automatically be linked with *meet residence requirements*. This feature reduces linking overheads for users and it is not offered by existing systems.

- Since PCR is based on reasoning an explanation can be given as to why a particular document is being retrieved. So PCR provides a reasoning facility for hypermedia. We have not developed this idea as yet.

- Displaying the percentage matching weight between retrieved documents and submitted requests helps users to select links, and hence makes navigation easier. This offers a solution to the problem mentioned in [Tomek & Murer 1992] of how

[3]We have already discussed the limitations of the other approaches like, hierarchical organisation and layered model.

to display information about available links so as to give a clearer idea about document relevancy to the current anchor.

- The availability of a summary of a document generated from its proof tree and structured information gives significant information about the contents of the document. This can obviate the need for users to read the whole contents of the document. This offers a solution to the problem of how to inform users about the contents of documents so that they can avoid irrelevant navigation.
- Using PCR in conjunction with database retrieval techniques improves the retrieval process as a whole and provides an intelligent method in hypermedia systems to retrieve information according to structured information. On the database side it allows unstructured information to be handled thus increasing their efficiency and effectiveness.

5.6 Concluding Discussion

The PCR technique proposed in this chapter is not intended to provide a complete knowledge representation model. Its main interest is to provide a general method for conceptual retrieval in hypermedia systems. It is a step towards including conceptual knowledge of destination anchors within the bodies of links which has the potential to greatly improve access to information. Furthermore the generic structure of our proposed links means that they can be used for a range of computational purposes, such as sending queries or supporting reasoning. Thus PCR aims to take maximum advantage of the presence of the kbs in the hypermedia environment.

Our proposed PCR links can be categorised as directional since they express relationships between source and destination anchors. The link relationship is logical. Unlike typical hypermedia systems, a link is not attached to a defined area within a window, but explicitly and logically bound to a portion of text (or area), which may appear in any document having conceptually relevant information. This means that a new document, having relevant information about an anchor, immediately acquires links when brought into the system by our process.

One advantage of PCR is that it offers automatic linking of a document to its lower level concepts once it is linked to its top-level concept. This means that documents having information about lower level concepts can be retrieved through their lower level concepts thus reducing the manual linking load for users. Some other advantages of PCR which can improve navigation are the display of a level of relevancy between retrieved documents and user requests and system's generated summaries of documents. This reduces the chances of disorientation and cognitive overhead.

The use of database techniques together with PCR allows users to retrieve data items according to structured information. This facility could increase the effectiveness of hypermedia systems.

There are many issues which need to be further explored:

- ***Building a dictionary of concepts:***

  Development and maintenance of the kb as regards open texture concepts is an important issue (see Chapter 1). The manifestation of this problem is not so severe in our *kbh* framework since we are not using formalised knowledge to derive concrete results but for conceptual retrieval. Our main aim is to find information about particular concepts. For example formalising the concepts like *reasonable post, special circumstances* is quite difficult since it is not clearly defined.

  In our approach there is no need to simplify such concepts and the application developer can use them as they are. Our approach, in some sense, is similar to Kowalski and Sergot's work, the use of logical models in legal problem solving [Kowalski & Sergot 1990], leaving the open texture concepts undefined in logical models. We believe that during document linking users can tell the system that this particular document relates to this particular open texture concept; and during retrieval readers can check from the document the meaning of considered open texture concept. However, there are many other issues we have not covered in this thesis like negation, aspects of double negation and disjunction and probability which could usefully be explored.

  In the literature on the legal domain many systems have used a conceptual retrieval technique for retrieving relevant cases. These systems can be categorised into three groups; systems explicitly attaching some cases to open-texture concepts [Schild 1989]; systems which use retrieved cases for deriving results like MPC [Kowalski 1991]; and systems which retrieve cases so that users can use the information to draw conclusions (argumentation), like HYPO [Ashley 1990].

PCR does not have any similarity with first two approaches, but it does have some similarity with HYPO. The main aim of HYPO is to retrieve cases for information purposes, not for deriving conclusions. The PCR technique is similar to HYPO's approach up to this extent as it is designed for retrieving information according to user requests. The difference is that HYPO is designed for special purpose retrieval. It has a fixed format for entering/retrieving cases. In other words it only uses one particular definition for a top-level concept to represent case reports. PCR does not impose this limitation. A top-level concept may have different definitions and documents having information about different definitions can be linked and retrieved accordingly. This might help with the problem mentioned in [Susskind 1987a]: "Those materials that are regarded by legal experts as being formal sources of the law ought to be accepted as such by legal knowledge engineers without further jurisprudential question ... However, disputes have often arisen over what actual legal regime or system is in force or valid at any one time to the extent that legal experts themselves disagree". We believe PCR can retrieve information according to different points of view and this can help law practitioners to explore future lines of action.

PCR retrieval of information is directly based on the definitions of concepts. Retrieval will be as efficient and precise as the definition of the concepts are. For example the concept table may have the definitions below:

```
table:-
          (wooden; iron; mixed),
          (office; dining; study).
table:-
          (four legs; three legs),
          (wooden legs; iron legs),
          (wooden top; iron top; glass top),
          (small; medium; big),
          (office; dining; study).
```

The main difference between these definitions is that the first one is more concise, whereas the second is richer and more descriptive. One problem, however, with very rich definitions is that the system will ask many questions during the retrieval and labelling times and this could annoy users. Application developers need to make choices about the level of concept definitions which are appropriate for the problem. For this purpose they can use textbooks, experts, or common sense knowledge.

- *Exploring alternative techniques for storing similar cases.*

  Searching the Label Base in a realistic system is likely to present efficency problems. Recently we have been trying to develop another technique for storing labels. In place of keeping separate labels for each document we tried to use a common proof tree as a label for all those documents which have 100% matching proof trees. This reduces the search space and retrieval becomes faster. One problem is that each document has different structured information such as case number and so on, which can not use a common label. One solution to this problem is that we save this information separately from proof trees so that we can use a single proof tree as a common label. The system only consults structured information about documents when users want to see a summary or want to use database retrieval.

- *Making the matching algorithm faster.*

  In future we want to explore alternative strategies for storing proof trees which can improve the speed of the matching algorithm. At the same time we want to explore new search strategies to reduce the search space. At present we use a sequential method to find matching labels. One drawback is that for large entries this becomes very slow. In future we want to explore other strategies such as parallel search and partitioning of the Label Base.

- *Exploration of other control strategies*

  Currently we are only using backward chaining. In future we want to explore the feasibility of mixing backwards and forwards chaining, for example presenting the user with forms, for retrieval.

These are some of the main issues which will inform the direction of future research.

6 Development of Bi-Directional Communication Channel (BCC)

6.1 Introduction

The *kbh* framework combines two major technologies, hypermedia and kbs, but the communication channel (BCC) is also an essential part. Without the BCC there is no *kbh*. One objective of the composite architecture is to allow the use of third party products instead of building a hypermedia manager and a kbs shell. In this work we develop our own kbs shell - the reason for not using a third party kbs shell is explained in the next chapter - but nevertheless we need to select a programming language. So the selection of suitable third party products was an important decision. The most considered points in this regard were: the selected applications should match with the modular based, open architecture approach of *kbh* and should allow implementation of a bi-directional communication channel. Also, the selected applications should run under the same operating system and use the same hardware platform.

Hypermedia is the first essential component of the *kbh* framework. Although there are many hypermedia systems available, we selected Microcosm [Fountain *et al.* 1990]. The reasons for this decision were: it uses an open architecture approach which allows us to insert new filters for adding new functionalities; its architecture matches with the *kbh* architecture; its source code was accessible within our department.

The selection of Microcosm put some restrictions on the selection of hardware platform and kbs development tools. The current version of Microcosm runs on IBM-PCs under the MS Windows operating system so it was important to select the kbs development software that ran under the same environment. As mentioned earlier, our intention was to build the kbs in LP style, so we were interested in Prolog. We selected LPA Prolog-386 for Windows (PROWIN) [Steel 1993] from the few Prolog systems available running on IBM-PC's under Windows environment.

After making these decisions the next step is the development of the BCC. The main purpose of the BCC is to provide a communication channel between the two other components of *kbh*, hypermedia and kbs shell. It provides a message passing protocol through which these components can send and receive messages on a run time basis. The architecture of the BCC is discussed in Section 6.2. This section also mentions some advantages of bi-directional versus uni-directional communication.

Before implementing the BCC it was important to get a deeper understanding of some key concepts. These included a low-level understanding of the MS Windows operating system, since both applications run under it, a low-level understanding of Microcosm, and an understanding of PROWIN's foreign language features. The necessary background to these concepts is given in Sections 6.3, 6.4 and 6.5.

The development of the BCC was not like writing a stand-alone application for Windows. It was a substantially more complicated task. Complicated because the two applications concerned, Microcosm and PROWIN, were written by different vendors. Neither application provided any in-built facility through which it could communicate with the other. A substantial amount of work was carried out in the implementation of the BCC. A record of this work and the resulting implementation details are given in Section 6.6. Details of the BCC initialisation are explained in Section 6.7.

6.2 Architecture of BCC Module

The Figure 6.1 shows architectural details of the BCC. The BCC module has two sub-modules, *kbh_Filter* and *kbh_Message Handler*, and a common DLL data segment. Both, *kbh_Filter* and the *kbh_Message Handler* share this common data segment and store/retrieve data during communication.

The important tasks of *kbh_Filter* are: to receive and send messages to the Microcosm Filter Manager (MFM); to receive and send messages to the *kbh_Message Handler*; and to store and retrieve data in a common DLL data segment.

The important tasks of the *kbh_Message Handler* are: to receive and send messages to the *PKShell's Message Analyser*; to receive and send messages to *kbh_Filter*; and to store and retrieve data in a common DLL data segment.

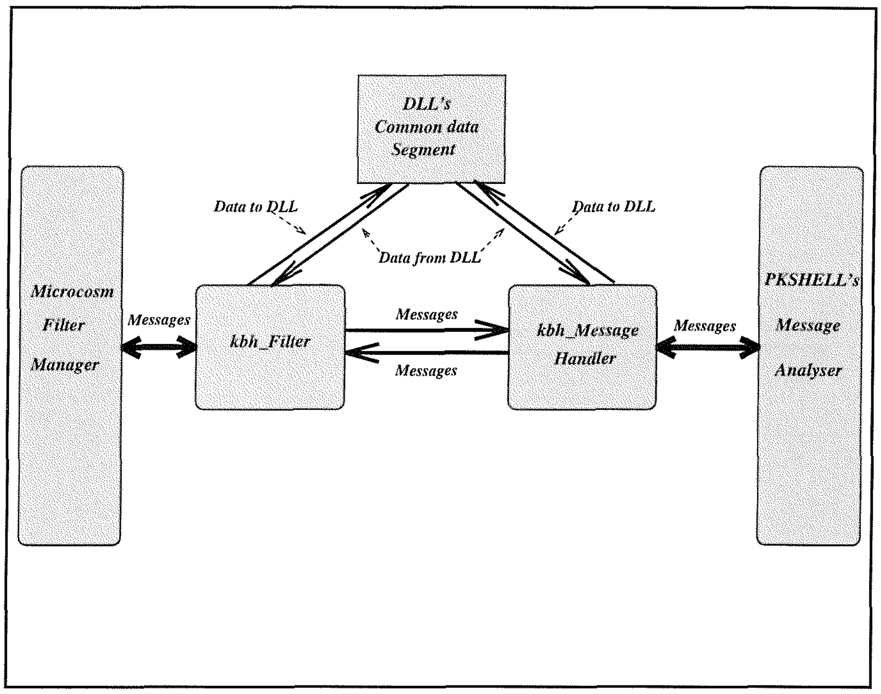


Figure 6.1: Figure showing architectural details of BCC.

It was mentioned in Chapter 3 that many systems offering kbs and hypermedia features provide uni-directional communication between the two. One advantage of a uni-directional link is that its implementation is very simple. However, an important problem is that it does not provide any feedback facility. Systems that provide uni-directional communication, use a master-slave model in which the front-end component acts as a master. Only the master can dictate tasks to the slave. The control of the system always remains in the master's hands. Only temporarily does it give control to the slave to perform a given task. The master does not receive any feedback from the slave. Many kbs applications developed using existing kbs shells use this layered model. User interaction with the system takes place through the kbs and the hypermedia just provides a windowing system that displays information whenever asked. Our point is that uni-directional communication between two technologies available on a single platform, is a very limited development and in no way the two technologies can exploit the full strengths of each other.

In contrast, bi-directional communication has no master-slave hierarchy. Both systems, hypermedia and kbs, are equal. At run time, any part of the combined system can take control of the whole system. The main advantage of bi-directional communication is that the systems can exploit the functionalities of each other for system levels task on a run time basis. In this style hypermedia gains the strength of an inference facility and kbs gain an effective user-interface.

6.3 Microsoft Windows Operating System

Microsoft Windows 3.1 is a trade mark of Microsoft Corporation. It provides a graphical user interface based on an event driven framework. This gives a uniform user interface to applications, through the layout of windows, menus and dialogue boxes. Using the event processing capability within the environment, it provides protocols for inter-process communication. The system operates on a wide range of IBM-PC compatibles which makes it, to a large extent, hardware independent. Windows 3.1 provides a multitasking environment which allows many applications to apparently run concurrently. This feature is actually provided through co-operation between executing processes themselves. Thus the multitasking environment of Windows is not the same as other multitasking operating systems such as UNIX and VMS. In the latter different processes can run concurrently using a time sharing scheme. Windows can load many processes into its work space but can execute only one process at a time. It is an arguable point whether Windows 3.1 is a true multitasking operating system or not.

6.3.1 Dynamic Data Exchange

Dynamic Data Exchange (DDE) is a protocol that allows Windows applications to exchange data on a real-time basis. This section, briefly explains DDE; for a detailed description readers should consult the original sources, see [Heath 1992], [Petzold 1992].

To exchange data between two applications, the applications must take part in a DDE conversation. The application that initiates the conversation is known as the *client*, and the application that responds to the client is known as the *server*. A given application may be involved in many conversations at any one time, and may be either client or server in each. By using the DDE protocol different applications can communicate with each other.

Three main steps occur in DDE communication:

1. **Initiating**

   The client broadcasts a message to all active windows within the system, having information about an event. Any server window that supports the event replies, and conversation begins.

2. **Conversing**

   There are four ways in which data is transmitted between client and server during a conversation.

   (a) The client can send a single item of data to the server, and also the server can transmit a single item of data back to the client.

   (b) The client can request a single item of data from the server.

   (c) The client can send commands to be executed by the server.

   (d) The client can request to be informed if a specified item of data held by the server changes. This notification should happen every time the data changes until the request is cancelled by the client.

3. **Terminating**

   The client is responsible for the termination of a conversation. It sends a message to the server, which the server must reply to, giving an indication of whether termination of conversation is possible at this time.

Underlying these three main functions, there are a host of messages that have to be sent and received in order to implement the protocol. There is also a complex flow control system, requiring positive and negative acknowledgements to be sent depending upon the success or failure of a given request.

6.3.2 Windows Dynamic Link Libraries

Usually computer programming languages provide a method by which code may be compiled into an object library, and that library can then be linked into any application that wishes to use it. This procedure has a disadvantage, that the library code is replicated within all applications which use it. This is not a problem in the standard

MS-DOS, single tasking operating system, as only one application executes at a time. Under Windows, however, there may be multiple applications running, each having its own copy of library code and occupying a significant amount of the system's memory. Windows supports a facility called *Dynamic Link Library* (DLL) which allows different applications to share a single copy of a library. In the normal linking process, references to the functions or objects that exist in other modules are resolved at *compile-time*. In the case of DLL this resolution is attempted at *run time* [Heath 1992].

A DLL only ever exists as a single instance. The code, data, and resources in a DLL module are shared among all processes which use this module. They are not directly executable, and they do not receive messages directly. They are separate files containing different functions but these functions are accessible to other modules at run time. The most important feature of a DLL is its ability to pass information between processes. Since a DLL only has one data segment, irrespective of how many processes have invoked it, information stored in that particular data segment by one process can be retrieved by another process, and vice versa. This technique offers a considerable saving in memory. It also allows the libraries of an application to be upgraded without the need to re-compile the entire application [Steel 1993], [Petzold 1992]. The DLL feature of Windows is crucial for the development of the BCC as explained in Section 6.6.

6.4 Microcosm Hypermedia System Features

Microcosm [Fountain *et al.* 1990] started as a research project at the University of Southampton and recently became a commercial product. Microcosm is based on filtering technology. The filters, arranged in a queue, perform various system level tasks. Filters can be added/removed from the system's filter queue. It provides various linking and navigational tools. It does not impose any restriction on document size, keeps link information separate from document's actual contents and supports every kind of electronic information. Microcosm uses an open architecture approach based on the following principles [Davis *et al.* 1992]:

- *No ultimate distinction between authors and users.* The system allows all users to make links and add documents.

- *Loosely coupled system with a low level of inter-dependency.* The system allows coupling with other tools that run under the host operating system.

- *Modular to allow for slot-in experimental replacements.* The system provides a useful research platform which allows the replacement and insertion of old or new modules.

- *Separation of link information from data objects.* The system does not put any mark-up upon the original information. It keeps link information in a separate place called the Link Base. Saving link information separately allows the creation of links between documents that are held in read only media such as CD-ROM.

A conceptual model of Microcosm is shown in Figure 6.2, taken from [Davis *et al.* 1992]. Microcosm consists of a number of autonomous processes which communicate with each other through a message-passing system (Filter Manager). It uses the Windows DDE facility for inter-process communication. Microcosm processes are categorised in two main groups, *Viewers* and *Filters*.

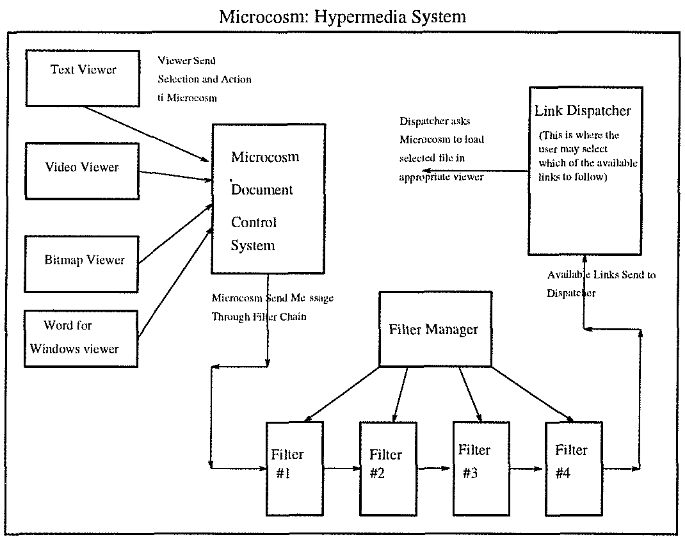

Microcosm: Hypermedia System
Text Viewer
Viewer Send Selection and Action ti Microcosm
Video Viewer
Bitmap Viewer
Word for Windows viewer
Microcosm Document Control System
Dispatcher asks Microcosm to load selected file in appropriate viewer
Link Dispatcher
(This is where the user may select which of the available links to follow)
Available Links Send to Dispatcher
Microcosm Send Message Through Filter Chain
Filter Manager
Filter #1
Filter #2
Filter #3
Filter #4

Figure 6.2: The conceptual model of Microcosm

- *Viewer:* Users interact with Microcosm through *viewers*. A viewer allows users to view a document in its native format. Microcosm provides a set of viewers which handle different documents according to their formats. The main task of these viewers is to allow users to peruse a document, to make selections and to choose actions. Typical Microcosm actions are *follow link, start link, end link*. The actions themselves are not affected by the viewers. The viewers are responsible for binding the information into a message, which is sent to a filter chain where the sent message will look for one or more processes that can satisfy the sent request.

Filters: Filters are special kinds of processes. When a filter is invoked, it opens a communication channel with Microcosm for message passing. Microcosm arranges active filters in a particular order called the filter chain. Filters are responsible for receiving messages, taking any appropriate action, and then handing the message on to the next filter in the chain. Filters may be dynamically installed, removed or reordered in the filter chain.

Filter Messages: Filter messages are in ASCII tag format. Usually they contain three kinds of information: *selection*, that is the selected information from the displayed document; *action*, that is what action is to be performed on the selection, such as *follow link, create link, end link*; and *contextual information* including the name of the document, selection off-set, date. It is possible to introduce new tags in a message, understandable to a particular filter. In this case that particular filter processes the message and the rest of the filters ignore it.

Filter messages can be generated in two ways. Firstly, by high-lighting a selection from a document and selecting an item from a pull down *Action* menu, as shown in Figure 6.3. Selection of an item from the *Action* menu will cause the message to be dispatched. Secondly, by pressing a *button*, which is actually a predefined selection of a document bound to an action. This is a special case of a more general message handling mechanism.

6.4.1 Filter Management System

One of the most important parts of Microcosm is its Filter Management System (FMS). This component is responsible for providing the functionalities of the system by communicating with individual filters. The FMS has three main tasks to perform. Firstly, to initialise all of the user's filters, described in the user's configuration file. Secondly, to maintain the filter chain while Microcosm is running. Thirdly, to allow

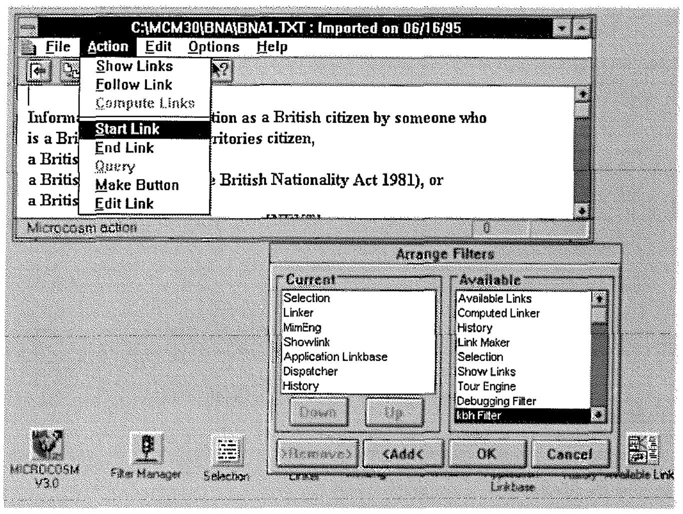


Figure 6.3: Microcosm's standard action menu and filter selection box

the filter configuration to change dynamically. A dialogue box, shown in Figure 6.3, presents the users with the lists of currently active and available filters. By selecting a particular filter, the user is able to relocate its position in the chain, or close it down altogether. Users can also add new filters into the system from the available filters list.

6.5 Prolog's Features to Communicate with Foreign Languages

Prolog (PROWIN) [Steel 1993] allows communication with other applications written in foreign languages. For this purpose it exploits the special feature of DLL. As discussed in Section 6.3.2, DLLs differ from other executable modules. The code, data, and resources in a DLL module are shared among all processes which use this module. It is important to mention here that for this purpose DLL must be developed according to PROWIN's requirement. Once a DLL is loaded into PROWIN, the DLL can start to perform background processing independently of Prolog. When this reaches a point where Prolog should react, the DLL generates a message. PROWIN provides a special hook for the DLL message handling. By writing a suitable definition for a DLL message handler, Prolog can be made to perform any operation based on a message, including calling explicit DLL functions to transfer string and other data [Steel 1993].

PROWIN provides the following builtin predicates for DLL handling. Some of their basic functions are discussed below and a detailed account is given in [Steel 1993].

`lopen(Library)`

This opens the particular DLL which is instantiated to the `Library` argument. It loads the DLL into Windows' memory, if it is not already there, and executes the *Prolog* function which registers the opened DLL to PROWIN. The *Prolog* function is the essential part of all DLL built for PROWIN. Once a particular DLL is registered all its functions are callable in any Prolog program.

`lclose(Library)`

This closes the DLL instantiated to the argument `Library` if it is already open. Since `lopen/1` loads a DLL into Windows memory so it is necessary to remove this from Windows memory so as to release the resources for other uses. As a result of `lclose/1` a special DLL function, *Epilog*, is automatically executed which frees the Windows resources. The *Epilog* function is also an essential part of any DLL built for PROWIN.

`ldict(Library)`

This predicate returns the dictionary of currently opened DLLs.

`lcall(Library, Function, Input, Output)`

This predicate is specially designed to use particular functions from an opened DLL in a Prolog program. It executes these functions from the DLL. `Input` and `Output` of this predicate can only be strings. It is important to mentioned here that this predicate is only useful for those DLLs which are specially designed for PROWIN.

`'?DLL?'(Message)`

This predicate is a DLL hook which handles messages generated from a DLL for PROWIN. The argument `Message` is the message ID sent from an open DLL.

Utilising these special features of PROWIN, we built a DLL-Message Handler which handles the messages coming from *kbh_Filter*.

6.6 Implementation of BCC

Implementing BCC was a more complicated task than developing a stand alone application for Windows. Microcosm and PROWIN are written by different vendors.

Neither application provided any in-built facility through which it could communicate with the other bi-directionally. The only possibility for communication was by trapping messages generated by the applications. This was a very low-level and complicated programming task which required many issues to be explored, such as

- How do we trap messages generated by these applications?
- How should the systems initiate communication?
- What kind of information does the BCC pass from one system to the other?
- How do we inform an application that a generated message is for it?

There were many more issues like these. After a thoughtful consideration and various programming experiments we got a clue about what to do. We built a new filter for Microcosm which traps particular messages generated by Microcosm. Using the Windows DLL facility the new filter stores the necessary information from these messages into a common DLL data segment and tells PROWIN to pick data from this particular data segment. On the way round we generate a message from PROWIN for the new filter which informs Microcosm of the required action. Working along these lines we implemented the BCC. For this purpose we built two separate modules, *kbh_Filter* and *kbh_Message Handler*. Firstly, we will discuss *kbh_Filter*.

Windows is an event driven operating system. The active application window is known as the host window. Whenever an event occurs in an active window the application broadcasts a particular message related to it. This message retains a message ID and the information about that event. All other active applications, which may be part of the host application or not, try to recognise this message. If any one of the active applications recognises the message it starts processing it and performs the required task. Microcosm is also an event driven application written under the Windows environment. It uses the same message broadcasting strategy for inter-process communication. Whenever an event occurs a message is broadcast with a particular tag. As discussed in Section 6.4 above Microcosm messages can be generated in two different ways: pressing a *button* from a document; or by selecting an item from an Action menu. Each Action menu item has a particular message tag. All messages from action menu are handled by a document control system (DCS) which sends them to the FMS and hence to individual filters for processing. For initialising the *BCC* communication from Microcosm we adopted the second method; selection of a menu item from the *Action menu.*

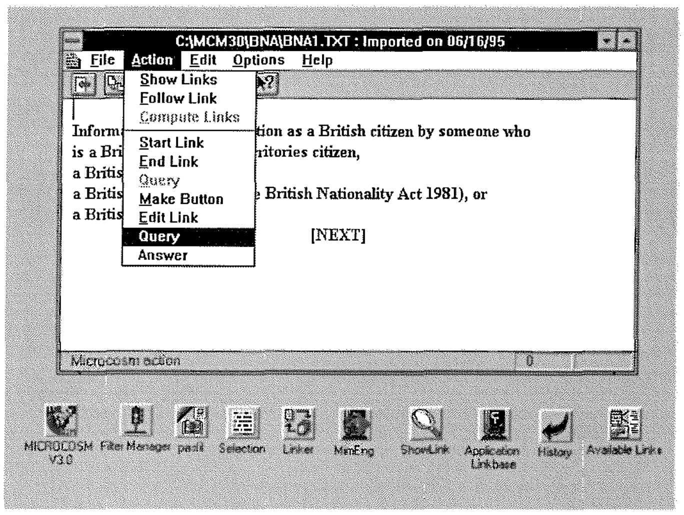


Figure 6.4: Microcosm's action menu with *kbh_Filter* additional options

For *kbh_Filter* we added two new pull down menu items, *Query* and *Answer*, in the Microcosm Action menu as shown in Figure 6.4. The first item, *Query*, is for users to send a query to kbs. The second item, *Answer*, is if kbs displays a document through the *DCS* and wants a 'reply selection' from the displayed document. The functions of these newly added Action menu items are fully explained in Chapter 7.

We designed two new messages having special tags for the newly added Action menu items. Only *kbh_Filter* can recognise these tags. The new messages have three essential parts as shown in the table below: a **selection** part which will be the area of the window selected by users (a data string); an **action** which is to be performed on the selection; and a **document name** which will be the name of the document from where the message is being initialised.

kbh_Filter Message Format	
Selection	*Text of arbitrary length.*
Action	*Query, Answer menu options.*
Document Name	*Message initiating file name.*

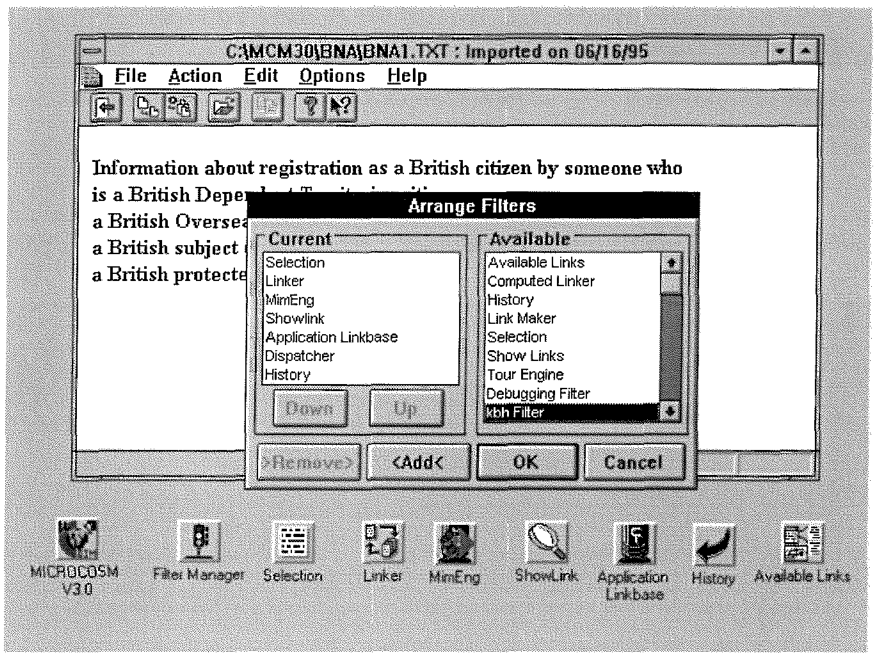
The first function of *kbh_Filter* is to identify these messages. When a message is identified it starts processing it. Firstly, it extracts useful information from the message and stores this into a common *DLL* data segment. Secondly, it sends a message with a particular tag to *kbh_Message Handler*. The *kbh_Message Handler* code is written in two different languages, 'C' and Prolog. It uses PROWIN's facilities to communicate with other languages, which in turn uses Windows' DLL features. The *kbh_Message Handler* performs three functions. Firstly, it identifies the messages coming from *kbh_Filter*. Secondly, it retrieves data stored in a common DLL data segment. Thirdly, it changes retrieved data into Prolog understandable format; and builds a new message for the PKShell's *Message Analyser*.

In the reverse direction, on request initiated by the PKShell's modules, The *Message Analyser* builds a message for *kbh_Message Handler* containing necessary information, such as viewer type and document name. In turn *kbh_Message Handler* processes this message and stores the necessary data in a common DLL data segment and sends a message to *kbh_Filter* to pick-up the data. Then, *kbh_Filter* builds a message and sends it to FMS which displays the document asked for under the control of the DCS.

6.7 Initialisation of BCC

In Microcosm, during system initialisation, the information about all available filters is notified to the FMS. The FMS keeps these filters in two queues; one for active filters, running during the current session, and the second for those which are available but are idle. It provides a selection box, shown Figure 6.5, from where users can activate an available filter by adding it into the active filter queue.

kbh_Filter is initially available in the available filters list. This was done intentionally to save the memory, for example users can run Microcosm only. It is possible to add *kbh_Filter* to Microcosm's default active filter queue, in which case the FMS automatically initialises *kbh_Filter* during start-up. During its initialisation *kbh_Filter* performs five tasks. (i) It adds new pull-down menu options to Microcosm's Action menu. (ii) It opens a communication channel with the FMS. (iii) It initialises the PROWIN. (iv) It loads the *kbh_Message Handler*. (v) It initialises the PKShell and loads various modules (detail discussion is given in Chapter 8).



Information about registration as a British citizen by someone who
is a British Depe
a British Overse
a British subject
a British protect

Current
Selection
Linker
MimEng
Showlink
Application Linkbase
Dispatcher
History

Available
Available Links
Computed Linker
History
Link Maker
Selection
Show Links
Tour Engine
Debugging Filter
kbh Filter

Figure 6.5: Microcosm's filter selection box

To send a message from Microcosm to the kbs, users need to select data from a displayed document and to initiate any one of the newly provided options from the Action menu. In response the DCS generates a message and sends it to the FMS which in turn sends this message to the filter chain. If this message has a particular tag which *kbh_Filter* recognises, the communication gets started.

The other way around, users can generate a message for Microcosm by using the kbs shell's options. In this case the *PKShell Message Analyser* sends a message to the *kbh_Message Handler* which stores the data from this message into the DLL's common data segment and generates a message for *kbh_Filter*. In turn, *kbh_Filter* retrieves the stored data from the common data segment and builds a new message for the FMS so as to perform the required task. The top level Prolog predicate of the *kbh_Message Handler* for communication between *kbh_Filter* and the *Message Analyser* is shown in Figure 6.6.

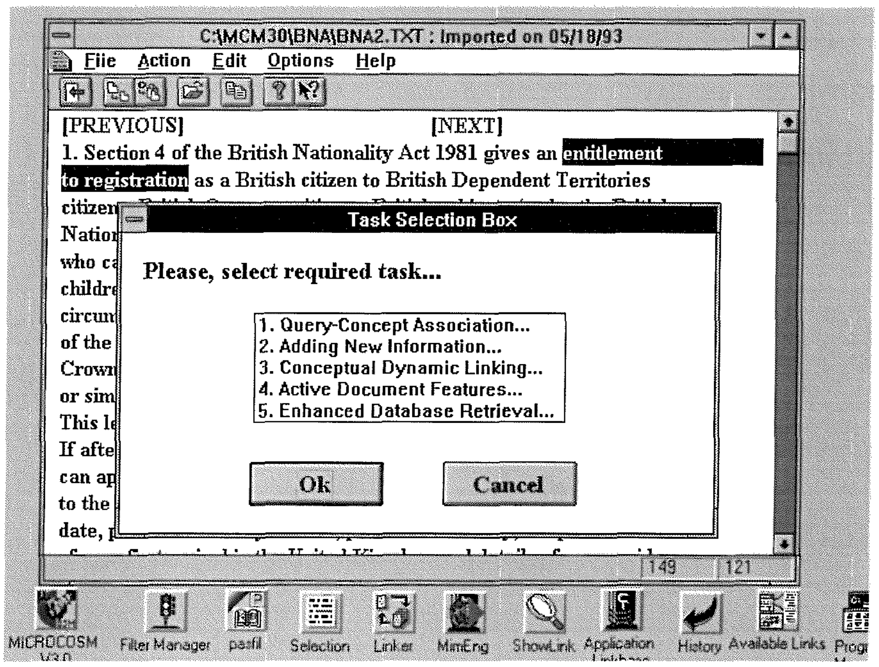
The functions of the two definitions of `'?DLL?'/1` are the same except that the first has an additional predicate `ask_required_task/1`. Five options are available for the required task: (i) Query-Concept Association. (ii) Adding New Information. (iii) Conceptual Dynamic Linking. (iv) Active Document Features. (v) Enhance Database

```
'?DLL?'(Msg_Num):-
        ask_required_task(Task),
        find_BCC_DLL(DLL_Name),
        pick_data_from(DLL_Name, DLL_Data),
        check_data_type(DLL_Data, Data_Type),
        process(DLL_Data, Data_Type, Proc_Data),
        send_to_analyser(Msg_Num,Data_Type, Proc_Data, Task).
'?DLL?'(Msg_Num):-
        find_BCC_DLL(DLL_Name),
        pick_data_from(DLL_Name, DLL_Data),
        check_data_type(DLL_Data, Data_Type),
        process(DLL_Data, Data_Type, Proc_Data),
        send_to_analyser(Msg_Num, Data_Type, Proc_Data,_).
send_Message(Prolog_Data, Viewer_Type):-
        find_BCC_DLL(DLL_Name),
        store(DLL_Name, Prolog_Data),
        send(message(pick), kbhfilter, View_Type).
```

Figure 6.6: Top-level predicate of kbs_Message Handler for communication

Retrieval (demonstrated in Chapter 8). The first top level clause, `'?DLL?'/1`, asks users about the required task through the predicate `ask_required_task/1` which shows a selection box, shown in Figure 6.7. Users can select any item. The main reason for doing this is to make the system more open for application developers. Including new options in the Action menu would involve changing the filter code which is a difficult task. This difficulty is avoided by keeping this option in the Prolog part. Application developers can then add new options in Prolog which is much easier. The predicate `find_BCC_DLL/1` finds the name of the DLL where the data is stored. Predicates `pick_data_from/2` and `check_data_type/2` pick the data and check its type[1]. After identifying the type, predicate `process/3`

[1]From hypermedia, users can send different kinds of data, such as text, bit map, etc. So the system has to check the data type before sending it to Prolog. At the moment we are only handling textual data



[PREVIOUS] [NEXT]
1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories

Please, select required task...

1. Query-Concept Association...
2. Adding New Information...
3. Conceptual Dynamic Linking...
4. Active Document Features...
5. Enhanced Database Retrieval...

Figure 6.7: Available task options for a request in *kbh*.

processes the data and converts it into Prolog understandable format. Finally, predicate `send_to_analyser/4` sends this information to the *PKShell's Message Analyser* for further action.

The second clause of `'?DLL?'/1` handles the Action menu option *Answer*. It is for sending asked information back to the kbs selecting from displayed document[2]. It performs all functions as described above except it does not ask about the task type which is obviously not required in this case. The definitions of predicates `pick_data_from/2`, `check_data_type/2` and `process/3` are a combination of DLL functions written in C and Prolog.

The top level predicate `send_Message/2` in Figure 6.6 is for sending data from the PKShell to *kbh_Filter*.

but in future we want to handle other data types also.

[2]If the kbs displays a document for such a purpose, the meta-interpreter suspends its execution and waits for an Answer message. This process is controlled by the meta-interpreter.

6.8 Conclusion

In this chapter we have discussed the software selection and the implementation of the BCC, a major part of *kbh*. As a hypermedia system for the prototype, we selected Microcosm as it is being implemented using an open architecture approach and it was available within our department. As our intention was to build a kbs using an LP approach, we selected Prolog, in particular LPA Prolog (PROWIN), for kbs development.

kbh has been implemented for IBM-PC under Windows environment. In this chapter we discussed briefly the Windows operating system, DDE, and DLL. For the implementation of the BCC we found that the Windows DLL feature provides a very useful environment for exchanging data between different applications. We found that applications running under Windows can communicate bi-directionally if system developers provide publicly accessible DLLs which other application developers can use in their applications. In this way a bi-directional communication link can be built without touching the source code of an existing application. We believe that providing this kind of facility is worthwhile for the whole software industry and has many software engineering benefits.

For the implementation of the BCC we exploited the DDE and the DLL features of Windows, the filter based architecture of Microcosm, and the communication facility of PROWIN with other languages. Using these facilities we built a new filter, *kbh_Filter*, a *kbh_Message Handler*, and introduced two new menu options to perform bi-directional communication between Microcosm and the PKShell.

The main advantage of the BCC is that it allows both systems to communicate with each other at run time basis. Also by developing this kind of communication link we have demonstrated that, if available applications satisfy some basic compatibility requirements, the strength of these applications can be combined into a single environment without the need for rewriting in a common language. It is clear that this saves time, money and human resources, and upgrading of the combined system will be much easier.

In this chapter we have shown in practice how to link LP with hypermedia. Our example demonstrates, however, that other AI technologies, such as natural language processing, neural networking, image processing, speech recognition, and so on, can be linked with hypermedia in a similar way. If today's software industry adopts a similar approach, and provides in-built facilities in applications for bi-directional communication with other applications, it will be easier to build composite systems and thus to improve the chances fulfilling the requirements of current users for handling the information crisis effectively.

7 Development of kbs Shell

7.1 Introduction

The *kbh* framework loosely couples a kbs with hypermedia (see Figure 3.1) through the BCC. We have discussed the BCC and the selected hypermedia for our prototyped system in the previous chapter. The third essential part of the *kbh* framework is the kbs. It consists of a domain independent shell and a domain dependent kb. One approach to the development of the kbs would be to start from a third party product. But, as has been stressed earlier, one of our aims is to demonstrate the strength of LP for solving hypermedia problems. Using a third party product would limit our ability to achieve this. More importantly, the development of a suitable shell is heavily influenced by the expanded context of *kbh*. This chapter describes the development of a prototype shell called **P**rolog based **K**bs **Shell** (PKShell). The shell is not a single entity in *kbh*; rather it is inference engine together with a variety of information management tools and a user interface. Our idea is that the shell is extendible to include additional useful tools. In this thesis we propose tools to support active document features, conceptual dynamic linking (PCR), and enhanced database retrieval.

The general argument for developing the shell in an LP approach is that logic provides expressive knowledge representation, a rigorous framework and the implemented tools of LP. In this style of developing kbs's not only is a clear distinction between the domain knowledge, the kb, and the kbs shell maintained, but also the shell itself is specified in a relatively declarative style. Section 7.2 describes the development of the inference engine in our prototype shell. The inference engine is designed using a meta-interpreter approach. The main predicate ('prove') of this meta-interpreter is basically a refinement of the Sterling and Loktia 'explain' predicate [Sterling & Lakhotia 1988]. The main differences are that our predicate is adapted to generate proof trees for succeeding and failing computations and offers query-the-user (QTU) facility.

In QTU facility, we have developed a 'help' facility for users when answering unclear system questions. Many approaches have been proposed in the literature for providing help during QTU sessions, such as asking simple questions [Wolstenholme 1989] or linking text files with such questions [Schild 1989]. One drawback of such approaches is that they are not flexible. Our approach to providing help during QTU sessions is more flexible and effective. Our contribution is to display the documents in which the concepts asked about have been explained. This can help users to understand these concepts. In our approach the connection between relevant information and concepts is not hard-wired, rather the system computes it dynamically. The system provides a flexible environment in which users can add new information easily. In Section 7.3, taking account of previously proposed approaches, we discuss our 'help' facility. In this section we also explain how our approach can help with handling open texture concepts.

In addition to the inference engine, the PKShell has five other modules: Message Analyser, Labeller, Matcher, User-Interface and System Library. All modules are developed using a meta-level programming approach. Section 7.4 explains the function of the four modules Labeller, Matcher, User-Interface and System Library.

The central module of PKShell is the Message Analyser. It acts like a post master. It receives messages from other modules and sends them to the relevant modules. It also provides communication links between the BCC and the PKShell. In addition to this, it is responsible for performing the tasks requested through the newly added option 'Query' in the Microcosm Action menu. From this option users can send requests to the PKShell for five different tasks: (i) Query-Concept Association. (ii) Adding New Information. (iii) Conceptual Dynamic Linking. (iv) Active Document Features. (v) Enhance Database Retrieval. For these tasks, the Message Analyser has five separate sub-modules responsible for performing these tasks. In Section 7.5 we explain the Message Analyser and its sub-modules are explained in Sections 7.5.1 to 7.5.5.

In developing the shell we adopted a modular style, mainly because this was useful for handling complexity and existing code. Also, it makes the shell open-ended allowing end-users to add new modules. Section 7.6 contains our observations on the software engineering importance of modularisation. The last section gives our conclusion.

7.2 Inference Engine

This module provides the inference facility of the PKShell. It is designed using an LP meta-interpreter technique. Meta-interpreters, in simple words, are programs which are commonly written to run other programs (object programs) under their control. One advantage of using a meta-interpreter is that it allows the object level knowledge to be left as a pure logic program, free from explicit procedural content. At the same time, meta-interpreters may provide not only alternative proof strategies, but also additional facilities that may be necessary, such as the provision of explanation, handling uncertainty, and so on [e.g. Sterling & Lakhotia 1988]. One side effect of meta-interpreters is that they slow down the execution of the program, but as Sergot says this reduction in efficiency is not too important in practice, since slower execution is acceptable in 'conversational' mode [Sergot 1982]. Also program execution can be speeded up by program transformation [Hogger 1981] and improved compilers [King & Soper 1994] as mentioned in Section 4.2. On balance the usefulness of meta-interpreters has been recognised in theory and practice and they have been used for various purposes [Sterling & Lakhotia 1988].

```
solve(true,true).
solve((GoalA, GoalB), (ProofA, ProofB)):-
             solve(GoalA, ProofA),
             solve(GoalB, ProofB).
solve(Goal, system):-
             system(Goal),
             call(Goal).
solve(Goal, Proof):-
             clause(Goal, Body),
             solve(Body, Proof).
solve(Goal, Proof):-
             askable(Goal),
             queryTheUser(Goal, Proof).
```

Figure 7.1: A meta-interpreter providing QTU facility and generating proof trees.

Usually meta-interpreters in LP are written at clause reduction level. All clause reduction meta-interpreters originated from the `demo` predicate, discussed in a seminal paper of Bowen and Kowalski [Bowen & Kowalski 1982]. Initially, we used a clause reduction level meta-interpreter, commonly given in the literature, as shown in Figure 7.1. The problem with this meta-interpreter is that it only generates proof trees of succeeding queries, whereas for our PCR technique proof trees of failing queries are also very important. We explored various proposals found in the literature [Bruffaerts & Henin 1988] and considered suggestions made on email by many researchers for building proof trees for failing goals. All were unsuitable for our purposes. Among the reasons were: some of them only explore the left most failing branch of the or-tree; and some of them do not instantiate values to the variables. The basic meta-interpreter currently in use in our prototype shell is a modified form of the 'explain' meta-interpreter given in Sterling and Lakhotia's paper [Sterling & Lakhotia 1988].

The main predicate of our meta-interpreter, `prove/4`, is shown in Figure 7.2. This meta-interpreter builds the SLD search tree for a given goal and the program definitions in the knowledge base. It has the property that it succeeds for every branch of the SLD search tree, regardless of whether or not the branch succeeds or fails. In each case it returns a proof tree in the second argument, that is an and-tree representation of the branch as described in Chapter 4, Section 4.3.2. The declarative meaning of `prove/4` can be stated as follows:

> `prove(Goal, Proof, Result, Trues)` holds if
> `Proof` is the proof tree for a branch of the SLD search tree for `Goal`, and
> `Result` is yes/no depending on whether the branch succeeds/fails, and
> `Trues` is the number of true leaf nodes in `Proof`.

Notice that if there are any success branches in the SLD search tree then

$$\text{kb} + \text{user\_info} \models \text{Goal}$$

On the other hand if there are no such branches then the SLD search tree is finitely failed and

$$\text{kb} + \text{user-info} \not\models \text{Goal}$$

```
%========================Single Goal==========================

        prove(Goal, (Goal, Proof), Result, Trues):-
                    rule(Goal, Body),
                    prove(Body, Proof, Result, Trues).
        prove(Goal, Proof, Result, Trues):-
                    askable(Goal),
                    queryTheUser(Goal, Proof, Result, Trues).
        prove(Goal, (Goal, system), yes, 1):-
                    system_call(Goal).
        prove(true, true, yes,1).
        prove(Goal, (Goal, fail), no, 0):-
                    undefined(Goal).
%=======================Conjunctive Goals=======================

        prove((GoalA, GoalB), (ProofA, ProofB), ResultB, Trues):-
                    prove(GoalA, ProofA, ResultA, ATrues),
                    prove_conj(ResultA, GoalB, ProofB, ResultB, BTrues),
                    Trues is ATrues + BTrues.
        prove_conj(yes, GoalB, ProofB, ResultB, Trues):-
                    prove(GoalB, ProofB, ResultB, Trues).
        prove_conj(no, GoalB, Proof, no, 0):-
                    tag_open(GoalB, Proof).
%=======================Disjunctive Goals=======================

        prove((GoalA; GoalB), Proof, Result, Trues):-
                    prove(GoalA, Proof, Result, Trues);
                    prove(GoalB, Proof, Result, Trues).
```

Figure 7.2: A meta-interpreter generating proof trees for succeeding and failing goals and offering QTU facility.

The fourth argument of `prove/4`, `Trues`, is used in the case of finite failure to select one of the failing branches as the proof tree label. The one chosen is that with the maximum values of `Trues`. In understanding the value of `Trues` it is important to recall, Section 4.3.2, that in a failure branch any goals following the first failure are not explored. These unexplored goals are labelled 'open'.

We now comment briefly on the structure of the `prove/4` definition. The first five clauses define the ways of processing a single goal: resolving with a kb rule; querying the user; a system call; returning `Proof` true if the goal is true; returning (Goal, fail) if the goal cannot be solved in any of the preceding ways. The next clause processes a conjunction of goals `(GoalA, GoalB)` by attempting to prove `GoalA` and `GoalB` separately. This clause calls an auxiliary predicate `prove_conj/5` which works like a case statement. If `ResultA` = yes, that is `GoalA` succeeds, then the first clause of `prove_conj/5` is used and an attempt is made to prove `GoalB`. If `ResultA` = no, that is `GoalA` fails, then the second clause of `prove_conj/5` is used, passing back the information that the goals `GoalB` are open. The final clause of `prove/4` simply processes disjunctive goals `(GoalA; GoalB)`.

We now turn to the selection of a particular branch of the SLD search tree of a goal **G** as the proof tree label. We distinguish two cases. When kb + user-info $\models$ Goal, that is `prove/4` succeeds with `Result` = yes, we select the first succeeding branch found as the label. Contrarily, when **G** finitely fails, that is `prove/4` succeeds with `Result` = no, we select the branch with the maximum numerical value of `Trues`. If there is more than one failing branch (and-tree) having the same value of Trues then some options for choosing a single branch are: ask users to select one, displaying branches one by one; select the left-most branch from such branches; select the right most branch from such branches. We chose the last option for our purposes. We reason that as the proof strategy of the meta-interpreter is from left-to-right so the user was not happy with the left branch and tried to explore the others. For the right branch, the user has already explored the other branches of the or-tree up to this point and appears to be keen to explore this particular right-most branch. This clearly shows his/her greater interest in the branch. This is not the only way to explore failure. We are considering other possibilities.

Our label selection procedure is implemented by the `explain/3` predicate defined in Figure 7.3. This figure also shows the top-level predicates of the Inference Module. The declarative meaning of `explain/3` is as follows:

> `explain(Goal, Proof, yes)` holds if
> `Proof` is the proof tree of the first succeeding branch of
> the SLD search tree for `Goal`.

```
taskFor(inferenceEngine, buildProofTree, Goal, (ProofTree, Result)):-
        explain(Goal, ProofTree, Result).
explain(Goal, ProofTree, Result):-
        allProofs(Goal, FullProofInfo),
        selectOneProof(FullProofInfo, ProofTree, Result).
allProofs(Goal, FullProofInfo):-
        findall(
               (ProofTree, Result, Trues),
               prove(Goal, ProofTree, Result, Trues),
               FullProofInfo
               ).
```

Figure 7.3: Inference Module's top level predicates.

`explain(Goal, Proof, no)` holds if
`Goal` has a finite failure SLD search tree, and
`Proof` is the proof tree of its most interesting branch
(i.e. the and-tree with maximum numeric value of `Trues`
or right most of them).

So in each case the second argument of `explain/3` returns the proof tree label for `Goal`.

7.3 Query-The-User with Help Facility and Open Texture Concepts

In this section we describe the Query-The-User(QTU) facility of the shell whose general function is to ask users for missing information required to prove a goal. In particular, we describe a help facility for users answering the system's questions. We also discuss how QTU can help with handling open texture concepts.

Wolstenholme [Wolstenholme 1989] proposed that if users do not understand a question posed by the system, then the system asks a simpler question declared at the meta-level. These simple questions should not be part of the object program. This is a good suggestion, however, some issues need clarification, for example: Who will decide which question is difficult? Should simple questions be attached with every askable question? The problem of users not understanding a question often occurs in relation to open texture concepts. So a particular issue is designing simple questions for these concepts. By their nature such concepts can not be formalised adequately. So what sort of information or simpler question should be provided?

Apart from such questions, another approach for open texture concepts is to attach simple text files [Schild 1989] or hypertext files of case reports [Hamfelt & Barklund 1990], to each open texture concept so that users are helped by reading this information. This is rather inflexible since it hard-wires relevant information to concepts. The system needs changes if new information arrives related to a concept. Also, linking hypertext files inherits all those problems which we have mentioned in Section 2.4.

To provide help during a QTU session we took a different approach from either of these mentioned above. We display documents in which the concepts being asked about have been explained. This can help users to understand the concepts. In our approach the connections between relevant information and concepts are not hard-wired, rather the system computes them dynamically. So our system provides a flexible environment in which users can add new information easily. In our prototype we simply use the open texture concepts as they are. For example, during a labelling session[1] when the system asks this question:
Is it true has_special_circumstances(client) ?
users can answer by reading the document in-hand, checking from its contents whether the client's circumstances are considered as special circumstances or not. Let us say the user's answer is 'yes'. The system now knows that this document has information about special circumstances. This information is available for future use. Let us suppose, in a QTU session, the system asks the same question and the user does not understand it clearly and wants some help. In this case the system dynamically retrieves the documents in which this particular concept was considered. Reading the retrieved documents users can get an idea about the concept which certainly should help them to answer the system's question. We believe this is a good approach to

[1]Entering a new document into the system.

handling these kinds of concept since the system provides maximum information to users and leaves it up to them to understand the meanings of open texture concepts and to take decisions.

The system provides a help option in the QTU dialogue box which retrieves any relevant information about the system's question dynamically. When a user asks for help, to answer an unclear question, the system displays a selection box asking for the type of information they would like, for example text book information, hypothetical cases, actual case reports. Getting this selection, the system searches the Label Base for proof tree labels associated with this type of document and matching the appropriate conceptual information. The document names associated with the retrieved proof tree labels are displayed. Users can see these documents (first in summary) if they want.

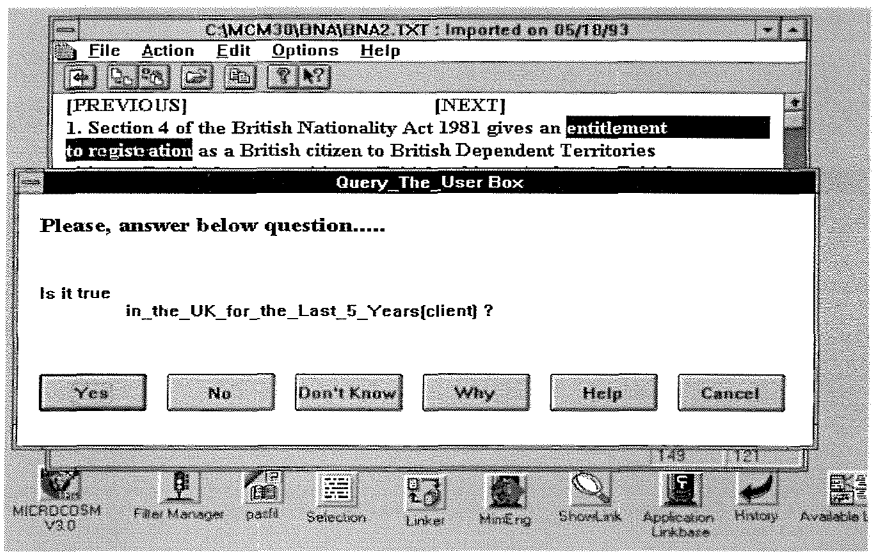
In the development of our prototype system we tried these three options. (i) The system only retrieves those documents that have information about the system's question with truth value true. (ii) The system only retrieves those documents that have information about the question with truth value false. (iii) The system retrieves all those documents that have information about the question, regardless of truth value true or false, but displays them separately. Saying which of these option is the best at this stage is not possible; evaluation in the field can decide better.

The definition of our `queryTheUser/4` predicate is shown in Figure 7.4. The predicate `askUser/2` poses the system's question to users. It displays a message box showing the question and having six response options: 'Yes', 'No', 'Do not know', 'Why', 'Help', 'Cancel'; shown in Figure 7.5. Here we explain only the 'Help' option, since this is the only novel part.

When the 'Help' option is selected the predicate `suspendExecution/2` present in the body of `checkRes/5` suspends the execution of the meta-interpreter and starts finding relevant documents by asking their type. Let us consider the user wants to see case reports. The system generates two proof trees for the current session; one for `(Question, true)` and a second for `(Question, fail)`. After generating these proof trees, it searches for matching proof tree labels of the document type 'case report' in the Label Base. It retrieves proof tree labels of all those case reports in which the system's question is considered irrespective of the court's finding. (Section 4.6 described how the system retrieves these proof tree labels.) The system then sorts the

```
queryTheUser(Goal, Proof, Result, Trues):-
        askUser(Goal, Response),
        checkRes(Response, Goal, Proof, Result, Trues).
checkRes(help, Goal, Proof, Result, Trues):-
        suspendExecution(Goal, VisitedDocuments),
        askQuestionAgain(Goal, Proof, Result, Trues),
        askToSaveReference(VisitedDocuments, SelectedReferences),
        saveReferences(SelectedReferences).
suspendExecution(Goal, VisitedDocuments):-
        showHelp(Goal, VisitedDocuments).
showHelp(Goal, VisitedDocuments):-
        askDocumentType(DocType),
        findDoc(DocType, Goal, FoundDoc),
        showDoc(FoundDoc, VisitedDocuments).
```

Figure 7.4: Definition of the query-the-user predicate.


C:\MCM30\BNA\BNA2.TXT : Imported on 05/18/93

[PREVIOUS] [NEXT]

1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories

Query_The_User Box

Please, answer below question.....

Is it true
in_the_UK_for_the_Last_5_Years[client] ?

Figure 7.5: QTU box

retrieved labels into four categories according to Table 7.1 and displays the titles of the case reports ready for selection.

According to Table 7.1, the first row corresponds to the category of case report in which the answer to the system's question and the case result agree on true . It is

Question Result	Case Result
TRUE	TRUE
TRUE	FALSE
FALSE	TRUE
FALSE	FALSE

Table 7.1: Possible system questions and case report results

expected that reading these case reports is worthwhile. In the second row, however, the case result did not succeed, but that might be due to other conditions. The court has accepted the definition of this particular question as true. Hence these cases are also useful for reading. The other two rows do not have a direct relationship to the question as the answer to the question is not true. We believe, however, that this kind of information may be important in legal practice, so as to understand what will happen if one does not satisfy the question asked.

Users can see a summary of the selected document first, before seeing the whole text of the document via hypermedia. The system generates this summary automatically using the proof tree label and the structured information of the document stored in the Label Base. The system keeps a record of visited documents. After getting information from these documents users are in a better position to answer the system's question. At the end of the QTU session, using the predicate `askToSaveReferences/2`, the system displays a list of visited document names so that the user can select those documents which they think are relevant for reference. Using the predicate `saveReferences/1`, the system saves these references and displays them along with the result of the query. The practical demonstration of this approach is given in our case study (see Chapter 8.6).

7.4 Labeller, Matcher, User-Interface and System Library Modules

In this section we describe four important modules of the PKShell: Labeller, Matcher, User-Interface and System Library. These modules are responsible for performing various tasks as explained below.

Labeller:

This module is responsible for generating suitable proof tree labels for newly entered documents and user requests for retrieving documents. Firstly, it sends the user's selected query to the Inference Engine to build its proof tree. Then it converts the proof tree into a suitable format, called a 'proof tree label'. The top-level predicates of this module are given in Figure 7.6. `postMail/4` is an inter-module communication predicate explained in Section 7.5. In `taskFor/4`, the first two arguments are constant;

```
taskFor(labeller, buildLabel, Query, Label):-
    buildLabel(Query, Label).
buildLabel(Query, Label):-
    postMail(inferenceEngine, buildProofTree, Query, (ProofTree,Result)),
    makeSuitableFormate(ProofTree, Result, Label).
```

Figure 7.6: Labeller Module's top level predicates.

one is a module identifier and the second is a task identifier.

Matcher:

Our proof tree matching algorithm is implemented, in Prolog, in this module. The module gets a query proof tree label as input and tries to find similar proof tree labels in the Label Base. Its top-level predicates are given Figure 7.7.

Firstly, the predicate `match/3` matches the query proof tree label (Qpt) with itself to get the maximum matching weight. Then, using Prolog's meta-level predicate `findall/3`, it builds a list, `FoundLabels`, of those proof tree labels (Lpts) from the Label Base which match the query proof tree label to some level. Only those proof tree labels from the Label Base are included in `FoundLabels` if they have the same root node as the Qpt, or a sub-tree of Lpt has the same root node as Qpt, or a sub-tree of Qpt has the same root node as Lpt. The other proof tree labels which do not satisfy these conditions are not included in `FoundLabels`. The 2nd argument, predicate `matchingLabels/5`, of `findall/3` does the following:

- Pick a document proof tree label from the Label Base of appropriate type.

- Find the starting point, `findStartingNode/3`, for matching. Thus the query proof tree label and the document proof tree label can have different root nodes, but a sub-tree of one proof tree label has the same root node as the other. In this case it picks the sub-tree of the bigger proof tree label for matching.
- Calculate the matching weight, `match/3`, between query proof tree label and document proof tree label. `match/3` uses the proof tree matching algorithm explained in Section 4.6, generating a number, the matching weight, as a measure of conceptual similarity.
- Calculate the percent matching weight between the query proof tree label and the document proof tree label, predicate `calPercent/3`, as explained in Section 4.7.

User-Interface:
This provides the system-user dialog facility. All other PKShell modules use it for this purpose.
System Library:
All common-purpose predicates are available in this module.

7.5 Message Analyser

The Message Analyser is the coordinating module of the PKShell and acts like a postmaster. It is responsible for various tasks. The first is to handle communication between the *kbh_Message Handler* and the PKShell modules. It processes incoming messages from the *kbh_Message Handler* and posts them to their relevant modules. Inversely, it receives messages from other modules and posts them to the *kbh_Message Handler*. Its second task is to provide inter-module communication. The Message Analyser has three top-level predicates. The first, given below, is responsible for handling inter-module communication.

```
postMail(ModuleIdentifier, TaskIdentifier, InPut, OutPut):-
          performTask(ModuleIdentifier, TaskIdentifier, InPut, OutPut).
performTask(ModuleIdentifier, TaskIdentifier, InPut, OutPut):-
          taskFor(ModuleIdentifier, TaskIdentifier, InPut, OutPut).
```

Here `taskFor/4` is the top-level predicate of the modules available in the PKShell.

```
taskFor(matcherModule, matchLabel, (DocType, QueryLabel), FoundLabels):-
        matchLabel(DocType, QueryLabel, FoundLabels).
matchLabel(DocType, QueryLabel, FoundLabels):-
        match(QueryLabel, QueryLabel, MaxWt),
        findSimilarLabels(DocType, QueryLabel, MaxWt, FoundLabels).
findSimilarLabels(DocType, QueryLabel, MaxWt, FoundLabels):-
        findall(
                (PercentWt, DocLabel),
                matchingLabels(DocType, QueryLabel, DocLabel, MaxWt,
                PercentWt),
                FoundLabels
                ).
matchingLabels(DocType, QueryLabel, DocLabel, MaxWt, PercentWt):-
        labellingInfo(_, DocType, _, DocLabel, _),
        findStartingNode(QueryLabel, DocLabel, TreeA, TreeB),
        match(TreeA, TreeB, MatchWt),
        calPercent(MaxWt, MatchWt, PercentWt).
findStartingNode(QueryLabel, DocLabel, QueryLabel, DocLabel):-
        hasSameRoot(QueryLabel, DocLabel, QueryLabel, DocLabel).
findStartingNode(QueryLabel, DocLabel, QueryLabel, DocSubTree ):-
        hasMatchingRoot(QueryLabel, DocLabel, DocSubTree).
findStartingNode(QueryLabel, DocLabel, QuerySubTree, DocLabel):-
        hasMatchingRoot(DocLabel, QueryLabel, QuerySubTree).
```

Figure 7.7: Matcher Module's top-level predicates.

The second top-level predicate, given below, only receives messages from the *kbh_Message Handler* and posts them to relevant modules.

```
analyseMail(MsgNum, DataType, InData, Task):-
                sortOutMail(MsgNum, DataType, Task, Module),
                postMail(Module, Task, InData, _).
```

The function of predicate `sortOutMail/4` is to find which module will perform the required task, based on information declared at the meta-level in the variables `MsgNum`, `DataType` and `Task`. The aim of introducing this predicate is to make the system more flexible and easily upgradeable. It allows users to build different modules for a single task, for example one module for labelling plain text files and the other for labelling SGML format files. It also allows users to add new modules for additional functionalities.

The third top-level predicate is for sending messages back to the *kbh_Message Handler*:

```
sendTokbhMsgHandler(Data, ViewerType).
```

The function of the predicate `sendTokbhMsgHandler/2` is to communicate data about the required task to the *kbh_Message Handler*, using DLL functions. The variable `ViewerType` is very important. As Microcosm has many viewers, to handle different types of document, this variable will instruct *kbh_Filter* to send information to the FMS to open the required document with the selected viewer. If this variable is uninstantiated then the system uses the default viewer, that is the text viewer.

Another important task of the Message Analyser is to perform user requested tasks. Through the newly added option *Query* in the Microcosm action menu, users can send requests for five different tasks:(i) Query-Concept Association; (ii) Adding New Information; (iii) Conceptual Dynamic Linking; (iv) Active Document Features; (v) Enhanced Database Retrieval. The Message Analyser has five sub-modules responsible for performing these tasks. These sub-modules are described in Sections 7.5.1 to 7.5.5.

7.5.1 Query-Concept Linker Sub-Module

Query-Concept Linker allows users to associate a kb query with any selected string and store this information in the *Label Base*. This information informs users about the available definitions in the kb as discussed in Section 5.2. The system stores this information for two purposes, active document features and conceptual retrieval. For active document features, the system runs the associated queries through the *Query Handler*, as explained in Section 7.5.4. For conceptual retrieval, the *Document Retriever* sub-module uses the queries (as explained in Section 7.5.3). Two main aims of this facility are, firstly, to overcome the retrieval problems related to synonyms. Associating a single query with different wordings of a concept instructs the system to consider them the same. Secondly, it lays the basis for creating active document features in hypermedia documents. To provide the active document features, users can either associate an available query from the kb with a concept (i.e. a string selection in the text - or selection in other media) or add new problem solving knowledge in the kb, associating the top-level query with the relevant concept. The data-flow diagram of this sub-module is shown in Figure 7.8 and its top-level predicates are given in Figure 7.9.

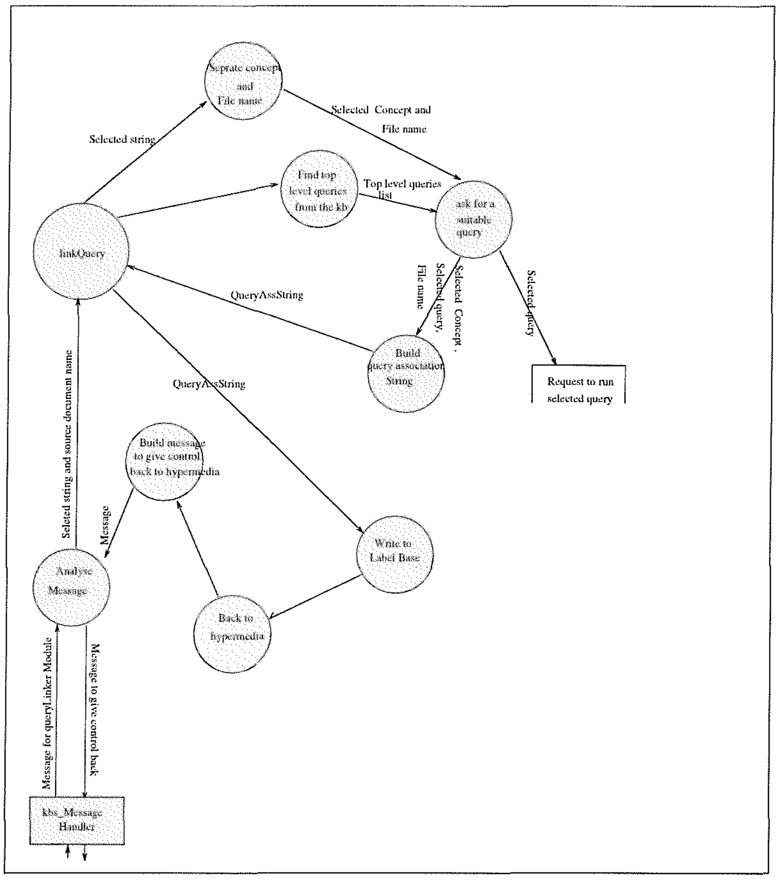

Seprate concept and File name
Selected string
Selected Concept and File name
Find top level queries from the kb
Top level queries list
ask for a suitable query
linkQuery
QueryAssString
File name
Selected query,
Selected Concept ,
Selected query
Build query association String
Request to run selected query
QueryAssString
Build message to give control back to hypermedia
Seleted string and source document name
Message
Write to Label Base
Analyse Message
Back to hypermedia
Message for queryLinker Module
Message to give control back
kbs_Message Handler

Figure 7.8: Query-Concept Linker sub-module's data-flow diagram.

The sub-module, firstly, separates the source document name and selected string from the data coming from Microcosm. Then it builds a list of top-level queries declared in the kb. It sends this list to the User-Interface for user selection. Users can run a selected query, via the Query Handler module, or using the predicates `buildAssInfo/4` and `SaveInfo/1`, store this information in the Label Base in the form

```
queryRel(SourceDoc, ConceptString, AssQuery).
```

```
taskFor(queryConceptLinker, linkQuery, InData, OutPut):-
    linkQuery(InData, OutPut).
linkQuery(InData, OutPut):-
    separate(InData, SourceDoc, SelectedStr),
    findTopLevelQueries(QueryList),
    postMail(userInterface, selectQuery, QueryList, UserSelQuery),
    buildAssInfo(UserSelQuery, SourceDoc, SelectedStr, AssociationInfo),
    SaveInfo(AssociationInfo).
```

Figure 7.9: Query-Concept Linker sub-module's top level predicates.

7.5.2 Document Adder Sub-Module

Document Adder adds new documents into the system. Firstly, it extracts the conceptual information from the document and then stores this information in the Label Base so that conceptual retrieval can be done. The data-flow diagram of this sub-module is shown in Figure 7.10 and the top-level predicates are given in Figure 7.11.

Through the User Interface, the module asks users for four items of information. The first is *document name*. The system uses this as an ID for the actual contents stored. Using the document name as an ID has three advantages: it takes less memory than the contents of the document; it does not affect the contents of the document, making the system more flexible; and information stored on read-only memory devices can be linked.

The second item is *document type*. Different documents may have different kinds of information about a particular concept. For example, information about a concept could be written differently in a text book, article, or case report. Users might want to link documents according to their types so that during a navigational session the system can retrieve them accordingly. The information about a linked document's type can be useful.

The third item is *relevant query*. The system can run this query to extract the conceptual information of the newly linked document. For relevant query selection, the module displays a list of top-level queries to choose from as discussed in Section 5.2. Users can select any one from the list shown.

The fourth item is *structured information* of the document. This is straightforward

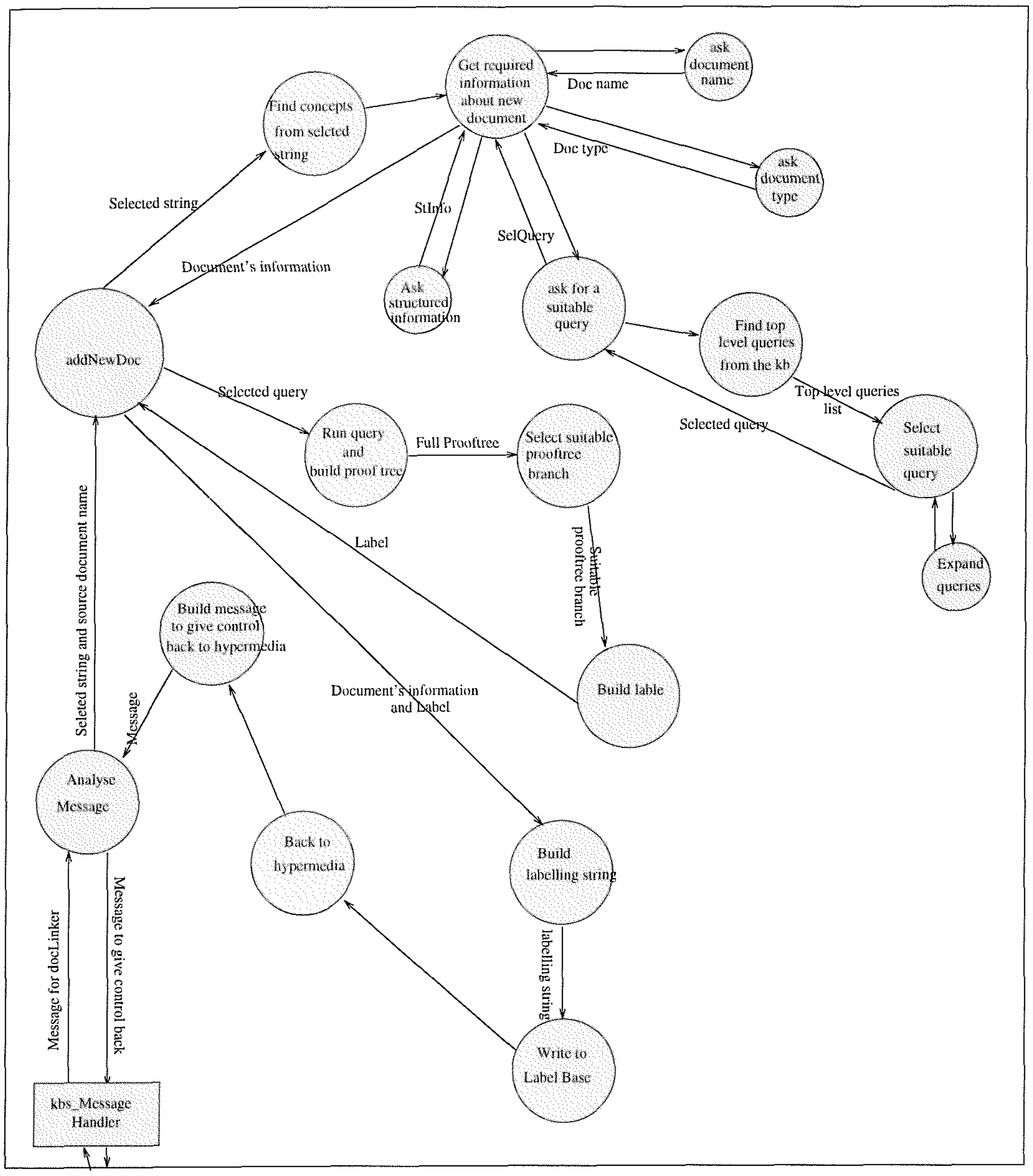


Figure 7.10: Document Adder sub-module's data-flow diagram.

data which is easily accessible from most documents of the given type and which is useful for providing a general idea about documents. For example, for case reports[2], we include case title, case number, court name, section applied; all these are generally available in case reports. During the retrieval process, such information is very useful in practice. The User Interface displays the window shown in Figure 5.7. It has different fields, especially designed to get structured information from case reports.

[2]For research articles the fields could be article title, journal name, issue number, volume number.

```
taskFor(documentAdder, addNewDoc, InData, OutPut):-
          addNewDoc(InData, OutPut).
addNewDoc(InData, OutPut):-
          separate(InData, SourceDoc, SelectedStr),
          getNewDocInfo(DocName, Query, DocType, StructuredInfo),
          postMail(labeller, buildLabel, Query, Label),
          buildLabellingStr(SelectedStr, DocName, DocType,
                            StructuredInfo, Result, Label,
                            LabellingString),
          saveInfo(LabellingString).
getNewDocInfo(DocName, Query, DocType, StructuredInfo):-
          postMail(userInterface, getDocInfo, _,
                   (DocName, Query, DocType, StructuredInfo)).
```

Figure 7.11: Document Adder sub-module's top level predicates.

The system uses this information for two purposes, structured retrieval (discussed in Section 7.5.5) and generating a summary of a documents automatically (discussed in Section 8.5.2).

After getting these items of information the module sends the selected query to the Labeller which in turn sends it to the Inference Engine which tries to satisfy it in a dialogue session with the user. Users supply this information from the body of the document according to their understanding. At the end of the session the Inference Engine generates a proof tree as explained in Section 7.2 and sends it back to the Labeller. Finally, it converts the proof tree into suitable format and the `buildLabellingStr/7` predicate builds two labelling information in the form:

```
labellingInfo(DocName, DocType, StructuredInfo, Label, Result).
queryRel(SourceDoc, SelectedStr, Query).
```

which is stored in the Label Base.

The new document is now linked with a particular concept. Once this information is stored in the Label Base, it is available for future conceptual and structured retrieval. The advantage of this way of linking is that apparently there is no physical link between the selected concept and the new document. However, the system can compute this link automatically. Another advantage of storing information in the Label Base is that if by chance a user physically removes a document, even then the system can display enough information about that document to give a general idea of its contents. In this way the system offers a graceful technique for handling dangling links. If users want to remove such information from the system they can do so by removing it from the Label Base.

7.5.3 Retrieval Sub-Module

This sub-module is responsible of handling requests for conceptual retrieval. Its first task is to get conceptual information from users about the required information. The second task is to find the documents which have conceptually similar or relevant information. The data-flow diagram of this sub-module is shown in Figure 7.12 and its top-level predicates are given in Figure 7.13.

Via the User Interface, the module asks users about search type. The system offers two search types, local and global. In local search, the system only displays concepts present in the user's selected string; those which have been associated with kb queries from source documents (from where the string is selected). In global search, the system displays concepts present in the selected string which have been associated with kb queries from any document. This means that a new document having similar wording for a linked concept will immediately acquire links when brought into the system by our process, a generic feature.

After getting the search type, the module searches the user's selected string for concepts which have already been associated with kb queries. Having found such concepts, it displays them for selection. The query associated with the selected concept is sent to the Labeller. In turn the Labeller sends this query to the Inference Engine which tries to satisfy it in a dialogue session with the users. Users supply information according to their current requirement. At the end of this session the Inference Engine generates a proof tree (as mentioned earlier) and sends it back to the Labeller which converts it into a suitable format and builds a proof tree label.

The module asks users about the document type and sends the newly generated proof tree label and document type to the Matcher Module for finding similar or relevant proof tree labels from the Label Base. The labels found are sent to the User

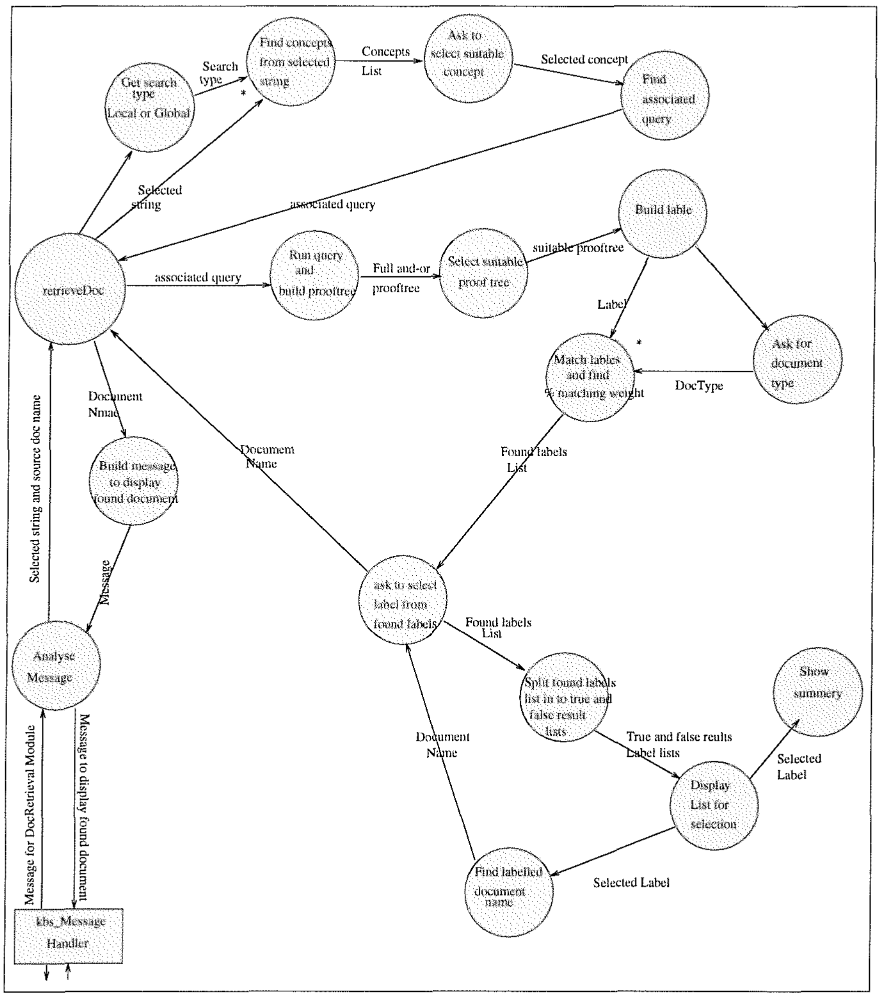

Get search type Local or Global
Search type
Find concepts from selected string
*
Concepts List
Ask to select suitable concept
Selected concept
Find associated query
Selected string
associated query
retrieveDoc
associated query
Run query and build prooftree
Full and-or prooftree
Select suitable proof tree
suitable prooftree
Build lable
Label
*
Match lables and find % matching weight
DocType
Ask for document type
Document Nmae
Build message to display found document
Document Name
Found labels List
Message
Selected string and source doc name
ask to select label from found labels
Found labels List
Analyse Message
Split found labels list in to true and false result lists
Show summery
True and false reults Label lists
Selected Label
Document Name
Display List for selection
Message for DocRetrieval Module
Message to display found document
Find labelled document name
Selected Label
kbs_Message Handler

Figure 7.12: Document Retrieval sub-module's data-flow diagram.

Interface for users' selection. Users can see the summary of the document before seeing the actual contents (as discussed earlier).

7.5.4 Query Handler Sub-Module

Query Handler is responsible of providing active document features in the *kbh* framework. Firstly, from a user selected string, it extracts the concepts which have been associated with kb queries and displays them for selection. Then it sends the selected query (concept) to the Inference Engine for evaluation and gets back, in general after a dialogue with the user, the result and proof tree. The User-Interface displays the result and offers six further options. (i) Users can ask the result to be explained displaying its proof tree. (ii) Users can ask to see the text of the rules used in the evaluation

```
taskFor(retrievalModule, retrieveDoc, InData,SelectedLabel):-
    retrieveDoc(InData, FoundLabels),
    postMail(userInterface, selectLabel, FoundLabels, SelectedLabel),
    displayDoc(SelectedLabel).
retrieveDoc(InData, FoundLabels):-
    postMail(userInterface, askSearchType, _, SearchType),
    separate(InData, SourceDoc, SelectedStr),
    findLabels(SearchType, SourceDoc, SelectedStr, FoundLabels).
findLabels(SearchType, SourceDoc, SelectedStr, FoundLabels):-
    findConcepts(SearchType, SourceDoc, SelectedStr, ConceptsList),
    postMail(userInterface, selectConcept, ConceptsList, SelectedConcept),
    find_Labels(SelectedConcept, FoundLabels).
find_Labels(SelectedConcept, FoundLabels):-
    postMail(userInterface, askDocType, DocType),
    findQuery(SelectedConcept, AssQuery),
    postMail(labeller, buildLabel,(AssQuery, QueryLabel),
    postMail(matcherModule, matchLabel, (DocType, QueryLabel), FoundLabels).
```

Figure 7.13: Document Retrieval sub-module's top-level predicates.

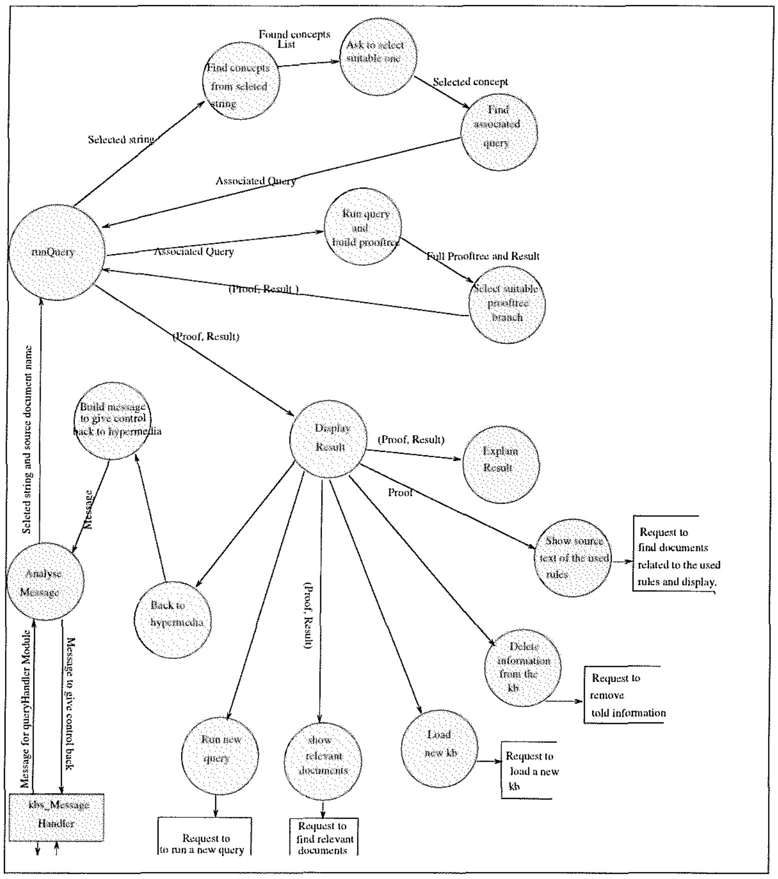
of the query; the file name associated with the rule is sent via the Message Handler to Microcosm to display the file. (iii) Users can remove the information told during the QTU dialogue. (iv) Users can remove the consulted kb and load a new one. (v) Users can run a new query. (vi) Users can ask for information relevant to the current query result; the proof tree of the current session is matched (in the Matcher Module) so as to find the relevant documents which users can see in summary and, if desired, read in full text through Microcosm. The top-level predicate of the Query Handler sub-module is given in Figure 7.14 and data-flow diagram is given in Figure 7.15.

7.5.5 Structured Retrieval Sub-Module

This sub-module handles requests for structured retrieval as explained in Section 5.4. Firstly it gets a request with respect to which users want to retrieve information. Then it searches the Label Base for entries which match the given search key. If one is found, it picks the proof tree label of that entry from its labelling information and sends it to the Matcher Module to find similar or relevant proof tree labels from the Label Base.

```
taskFor(queryHandler, runQuery, InData, OutPut):-
    separate(InData, SourceDoc, SelectedStr),
    findConcepts(_, SelectedStr, ConceptList),
    postMail(userInterface, selectConcept, ConceptList, Selection),
    find(Selection, AssQuery),
    postMail(inferenceEngine, buildProofTree, AssQuery, (Proof, Result)),
    postMail(userInterface, showResult, (Proof, Result), OutPut).
```

Figure 7.14: Query Handler sub-module's top level predicates.



Found concepts List
Find concepts from selected string
Ask to select suitable one
Selected concept
Find associated query
Selected string
Associated Query
runQuery
Run query and build prooftree
Associated Query
Full Prooftree and Result
Select suitable prooftree branch
(Proof, Result)
(Proof, Result)
Seleted string and source document name
Build message to give control back to hypermedia
Display Result
(Proof, Result)
Explain Result
Proof
Message
Show source text of the used rules
Request to find documents related to the used rules and display.
Analyse Message
Back to hypermedia
(Proof, Result)
Delete information from the kb
Request to remove told information
Message for queryHandler Module
Message to give control back
Run new query
show relevant documents
Load new kb
Request to load a new kb
kbs_Message Handler
Request to to run a new query
Request to find relevant documents

Figure 7.15: Query Handler sub-module's data-flow diagram.

These are followed by the full contents as described above.

The top-level predicates of this module are given in Figure 7.16 and the data-flow diagram is shown in Figure 7.17.

```
taskFor(strRetModule, strRetrieval, InData, OutPut):-
    postMail(userInterface, askSearchKey, _, SearchKey),
    matchSearchKey(SearchKey, MatchingLabInfo),
    find(MatchingLabInfo, Label),
    postMail(matcherModule, matchLabel, (_, Label), FoundLabels).
    postMail(userInterface, selectLabel, FoundLabels, Selectedlabel),
    display(SelectedLabel).
```

Figure 7.16: Structured Retrieval sub-module's top level predicates.

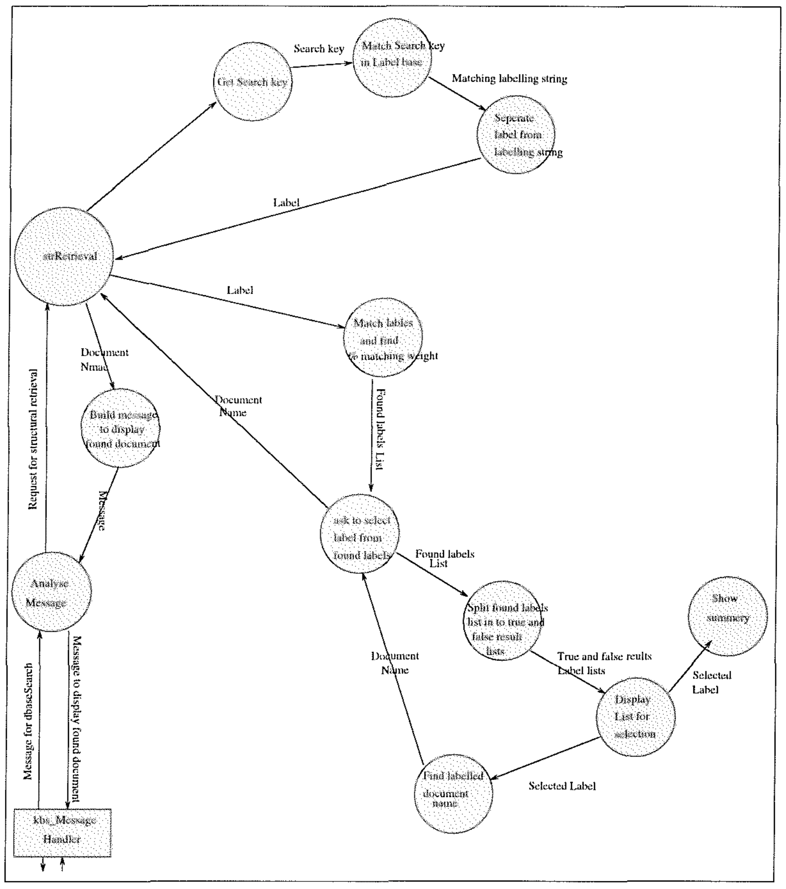

Search key
Get Search key
Match Search key in Label base
Matching labelling string
Seperate label from labelling string
Label
strRetrieval
Label
Match lables and find % matching weight
Document Nmae
Document Name
Found labels List
Request for structural retrieval
Build message to display found document
Message
ask to select label from found labels
Found labels List
Analyse Message
Split found labels list in to true and false result lists
Show summery
Document Name
True and false reults Label lists
Selected Label
Display List for selection
Message for dbaseSearch
Message to display found document
Find labelled document name
Selected Label
kbs_Message Handler

Figure 7.17: Structured Retrieval sub-module's data-flow diagram.

7.6 Software Aspects of Modularisation

In developing the PKShell we adopted a modular style, mainly because this was useful for handling complexity and existing code. Also, it makes the shell open-ended allowing end-users to add new modules. Here we want to give the answer to a particular question, which readers might have in their mind. Is it necessary to do this kind of modular programming in Prolog whose predicates are already one kind of small module? Yes, indeed Prolog's predicates are small modules, but this is not enough and this is why many Prolog system include support for modules. Our idea of modular programming is to make a logical separation between different parts of the system which perform their functions separately from each other as it provides an easy and systematic method for upgrading the system.

Suppose, an application developer does not like the Matcher Module provided and they want to use their own module. They can build their own matching module and during loading tell the system to load their module instead of the system's one. An important thing to mention about developing a new module is that the developer must check the compatibility of top-level predicates in other modules so as to conform with communication protocols otherwise control errors could occur. It will also be useful to declare information about new module top-level predicates in a system info file, so that other users can use them properly. This is one kind of online help. At the moment *kbh* does not do this but in future we are interested in providing hypertext like help about the system's different modules for upgrading or alteration.

7.7 Conclusion

In this chapter we have explained the main reason for developing the new kbs shell. Our prototype shell is not a single entity; rather it is an inference engine together with a variety of information management tools and a user interface. In developing the shell we adopted a modular style, mainly because this was useful for handling complexity and existing code. Also, it makes the shell open-ended allowing end-users to add new modules. Our prototype shell has six main modules: Inference Engine, Message Analyser, Labeller, Matcher, User-Interface and System Library. All modules are developed using an LP meta-level programming approach.

For development of the inference facility we adopted meta-interpreter techniques. We explained the advantages of LP for meta-level programming. The interpreter used in our shell is adapted from [Sterling & Lakhotia 1988]. It generates proof trees for both succeeding and failing queries and offers QTU facility. In the case of failure, it chooses that branch of the SLD search tree which has the deepest cause of failure. This was justified on the grounds that this branch is likely to be of most interest of the users. We also described a flexible approach for providing help during QTU dialogue. In our approach the system dynamically retrieves documents having information relevant to questions asked by the system. Our approach also offers a flexible method for providing help with open texture concepts.

In this chapter we described other PKShell modules which are responsible for performing various tasks. In our prototype shell the Message Analyser is the coordinating module. It acts like a postmaster and provides a means of communication between the other modules. It is also responsible of handling user requests submitted from the hypermedia manager to the shell. For this purpose it has five sub-modules which perform the requested tasks by communicating with the other main modules.

In developing our prototype shell we took modular approach. We argued that our design is important from a software engineering point of view. Since our application is written in Prolog, this point has particular force because the need for modules in Prolog, although recognised, is sometimes under-emphasised. Our experience with the PKShell has shown the usefulness of the modular approach primarily because it makes a logical separation between programs which perform different functions independently from each other. This modular style also provides a reasonably easy environment for upgrading the system.

8 Using the Prototype of *kbh* in the Legal Domain

8.1 Introduction

This chapter describes a case study carried out in the legal domain using our prototype *kbh*. The general motivation of the case study is to check the feasibility and functionalities of the newly developed information management tools and the *kbh* framework.

The case study is based on leaflets about British nationality registration provided by the Home Office [Home Office 1991a], [Home Office 1991b], [Home Office 1991c]. These leaflets contain a simplified exposition of the law and are associated with an application form. They are designed to provide information about British nationality registration in a simplified and understandable form for the general public or advice workers. In spite of this the information given in the leaflets is not rich enough for many purposes and, often, readers seek further advice from experts. Also, heavy usage of cross references, the presence of open texture concepts, overlapping of information with other leaflets, and the level of detail provided can render the leaflets less effective.

A hypermedia application based on the leaflets could overcome some of the above problems, but still users may not be able to use the the information in practice. Also a pure hypermedia application would introduce the disorientation problem, so that users might lose interest in exploring their problem, ultimately reducing the effectiveness of the system. On the other hand, a kbs advisory system, such as [Hammond 1983], for the leaflets would allow users to see the practicality of the information, but it would not provide adequate access to its details, and thus would not be very effective for the general public. The two main barriers which reduce the effectiveness of such kbs systems are: managing a satisfactory human-like user-system interaction; and that users, often, do not have a well defined question or, if they do, the question is modified or transformed during the session [Bench-Capon *et al.* 1991], [Soper & Bench-Capon 1994]. Using our *kbh* framework, we have developed a prototype *kbh* application for these leaflets which, on the one hand, offers an advisory system like environment, and, on the other hand, provides an effective environment for information exploration. We argue that the *kbh* combination offers a significant improvement over kbs or hypermedia alone.

Section 8.2.1 describes the prototype *kbh* system's specification and its initialisation. In Section 8.3, we discuss the method of linking kb queries with concepts from different documents. This section also explains our approach to handling synonyms. In Chapter 3 we made various claims about the *kbh* framework; including its openness and easy up-gradation. To support these claims, in Section 8.4, we discuss the system up-gradation in two senses: up-gradation by adding new information (in Section 8.4.1); up-gradation by adding new functionalities (in Section 8.4.2).

The prototype *kbh* system provides full liberty to users to utilise the functionalities of hypermedia, kbs, or both together. It provides a hypermedia environment for navigation through the body of information and offers two navigational modes: (i) using Microcosm navigational tools; (ii) navigation through the additional LP based kbs tools. Using the additional tools, users can dynamically create conceptual links between nodes in generic way (Section 8.5); can submit queries to the kbs providing active document features (Section 8.6); and can retrieve documents according to structured information offering enhance database retrieval (Section 8.7). The conclusion is given in Section 8.8

8.2 Case Study British Nationality Leaflets

8.2.1 System Specification

Referring to the *kbh* architecture, Figure 3.1, the intended informal relation in our prototype between the document base and the kb is as follows. The document base contains the text from three leaflets: *'Leaflet BN 12: Information about registration as a British citizen by someone who is a British Dependent Territories citizen, a British Overseas citizen, a British subject, or a British protected person'* [Home Office 1991c]; *'Guide B:*

Registration as a British citizen' [Home Office 1991b]; and *'Fees Leaflet March 1991'* [Home Office 1991a]. As much of the information is repeated in [Home Office 1991c] and [Home Office 1991b] we split up the text of these leaflets into small portions and store them in different hypermedia documents. These documents are then linked through the hypermedia manager's linking facility which creates an environment in which information from different leaflets is available in one body.

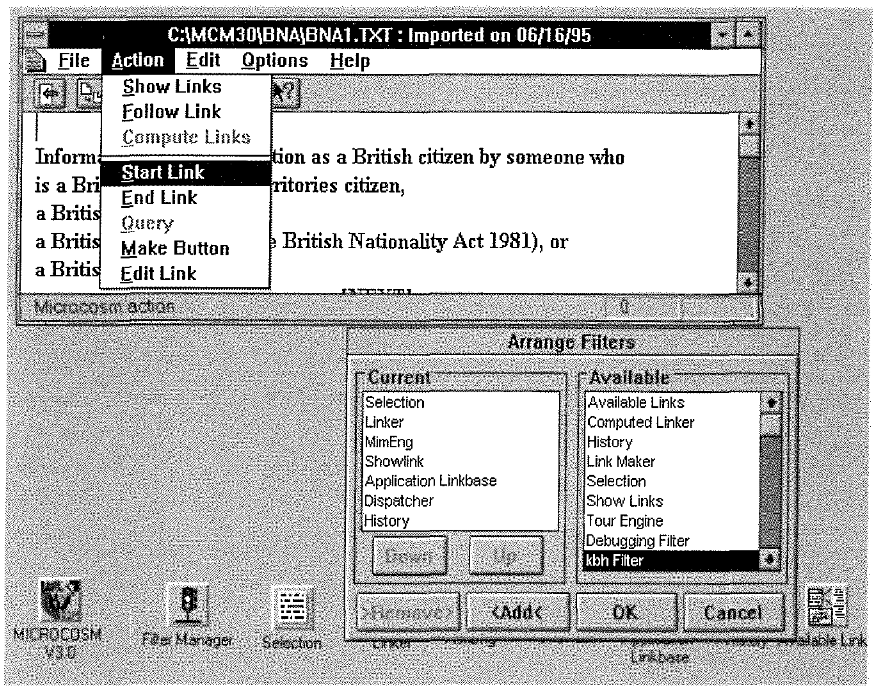
The kb contains definitions of important concepts from the leaflets. These definitions are taken from the text of the leaflets and formalised in Horn clause logic. In the first two leaflets entitlement to registration is the top-level concept and it has many lower level concepts. Using PCR some hypothetical cases, and some actual cases (stored as text files), have been linked with some important concepts.

The prototype containing this domain dependent information is used in the following sections. Hypermedia navigation of the document base solves some of the problems with the leaflets, such as problems of repetition of information and of cross references. But our main interest is that interaction with the conceptual definitions in the kb allows kbs features, such as dynamic linking, active document features and enhance database retrieval to solve others.

8.2.2 System Initialisation

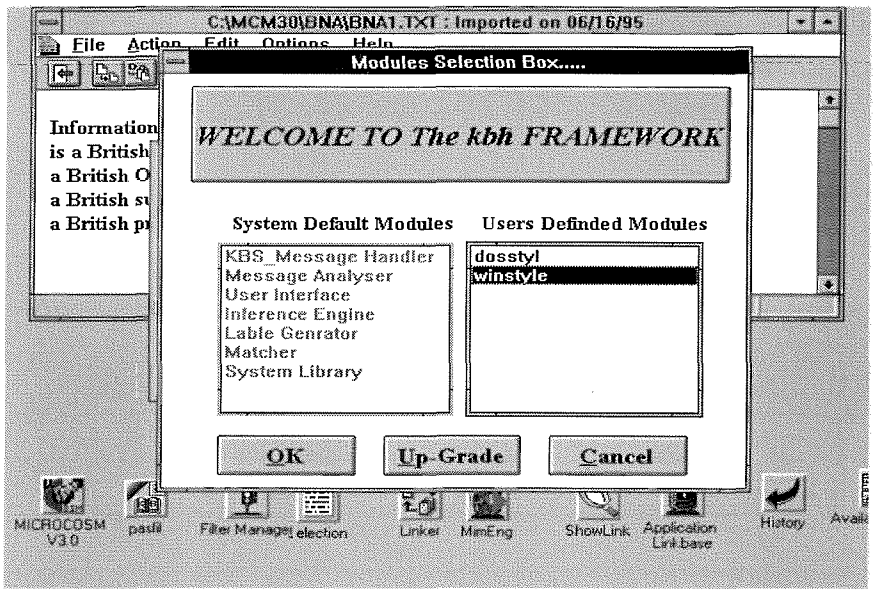
Users need to start by loading Microcosm and selecting the document base for the current session from Microcosm. Microcosm's standard Action menu options and the *kbh_Filter* in the 'Available' filters list is shown in Figure 8.1. (The reason for keeping *kbh_Filter* in the available filter list is that if users want to use the hypermedia only they can do so.)

To initialise the *kbh* framework, users need to initialise the *kbh_Filter* by adding it to the 'Current' filter list. During the initialisation process *kbh_Filter* opens the bi-directional communication channel, adding two new menu options to Microcosm's Action menu. It also loads the kbs, asking users for the PKShell modules and the kb for the current session as shown in Figure 8.2 and Figure 8.3. The system also loads the Label Base relevant to the selected kb. (The function of the 'Up-Grade' button in the message box shown in Figure 8.2 is explained in Section 8.4.2.) At the end of initialisation the system is ready for use. Figure 8.4 shows the newly added Action



Inform... tion as a British citizen by someone who
is a Bri... ritories citizen,
a Britis...
a Britis... British Nationality Act 1981), or
a Britis...

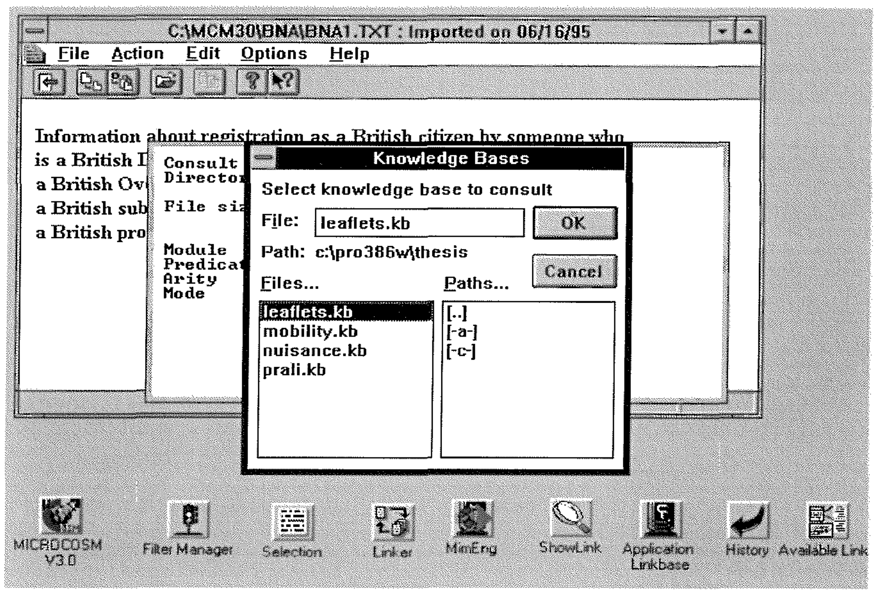
Figure 8.1: Microcosm's standard Action menu options and *kbh_Filter* in the 'Available' filter list.



Information
is a British
a British O
a British s
a British p

Figure 8.2: *kbh* module selection box.

menu options, 'Query' and 'Answer'. Through the 'Query' option users can send their queries to the kbs for the five different tasks shown in Figure 8.5. The following sections discuss each of these tasks in detail.

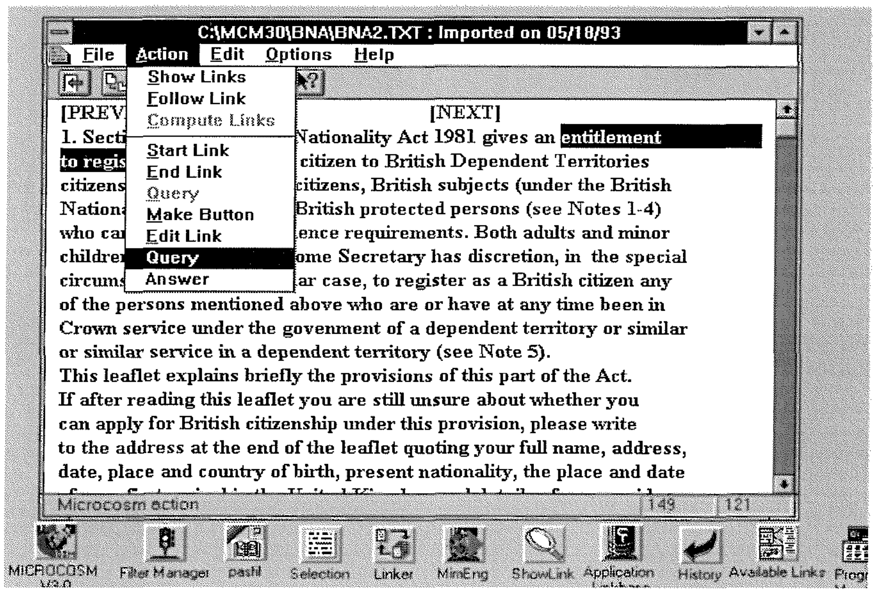


Information about registration as a British citizen by someone who
is a British D
a British Ov
a British sub
a British pro

Select knowledge base to consult
File: leaflets.kb
Path: c:\pro386w\thesis

leaflets.kb
mobility.kb
nuisance.kb
prali.kb

Figure 8.3: *kbh* knowledge base selection box.

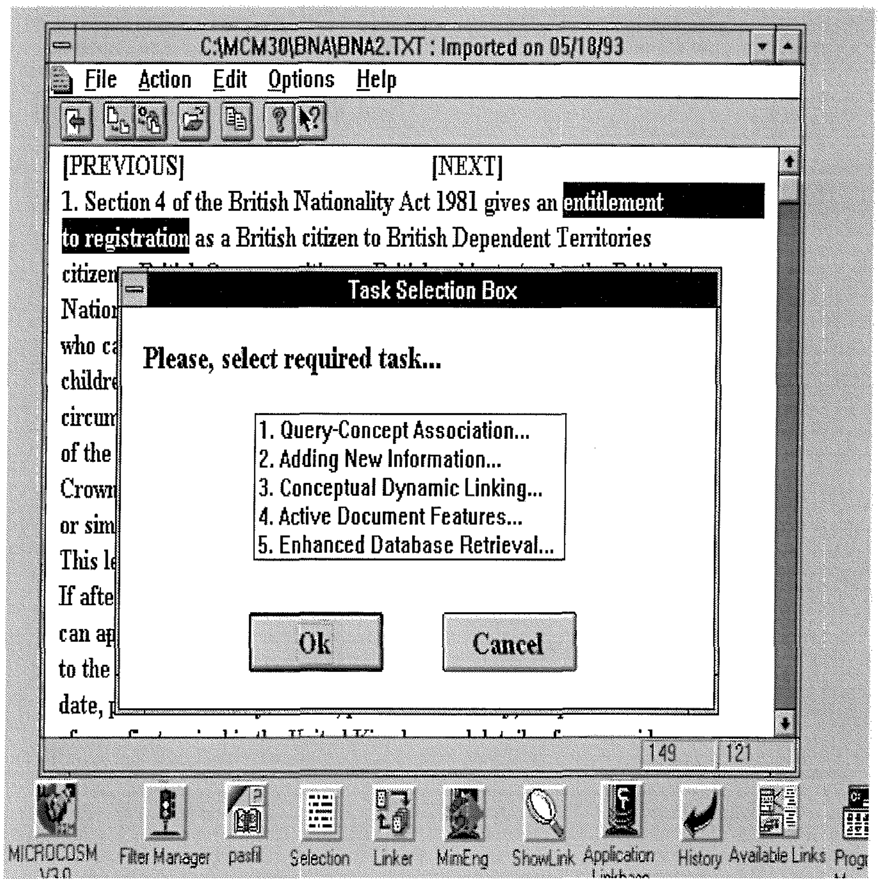


[PREV [NEXT]
1. Secti Nationality Act 1981 gives an entitlement
to regis citizen to British Dependent Territories
citizens citizens, British subjects (under the British
Nation British protected persons (see Notes 1-4)
who ca ence requirements. Both adults and minor
childre ome Secretary has discretion, in the special
circums ar case, to register as a British citizen any
of the persons mentioned above who are or have at any time been in
Crown service under the govenment of a dependent territory or similar
or similar service in a dependent territory (see Note 5).
This leaflet explains briefly the provisions of this part of the Act.
If after reading this leaflet you are still unsure about whether you
can apply for British citizenship under this provision, please write
to the address at the end of the leaflet quoting your full name, address,
date, place and country of birth, present nationality, the place and date

Figure 8.4: Newly added options in Microcosm Action menu

8.3 Linking kb Queries with Selected Concepts

This section demonstrates the procedure for linking kb queries with a selected string from a displayed document. A (suitable) string of characters is called a *concept* and after attaching it to a kb query it is called a *linked concept*. The linking procedure is called *Query-Concept Association*. The details of this procedure are explained below.


C:\MCM30\BNA\BNA2.TXT : Imported on 05/18/93

[PREVIOUS] [NEXT]

1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories

Task Selection Box

Please, select required task...

1. Query-Concept Association...
2. Adding New Information...
3. Conceptual Dynamic Linking...
4. Active Document Features...
5. Enhanced Database Retrieval...

Figure 8.5: Task selection box

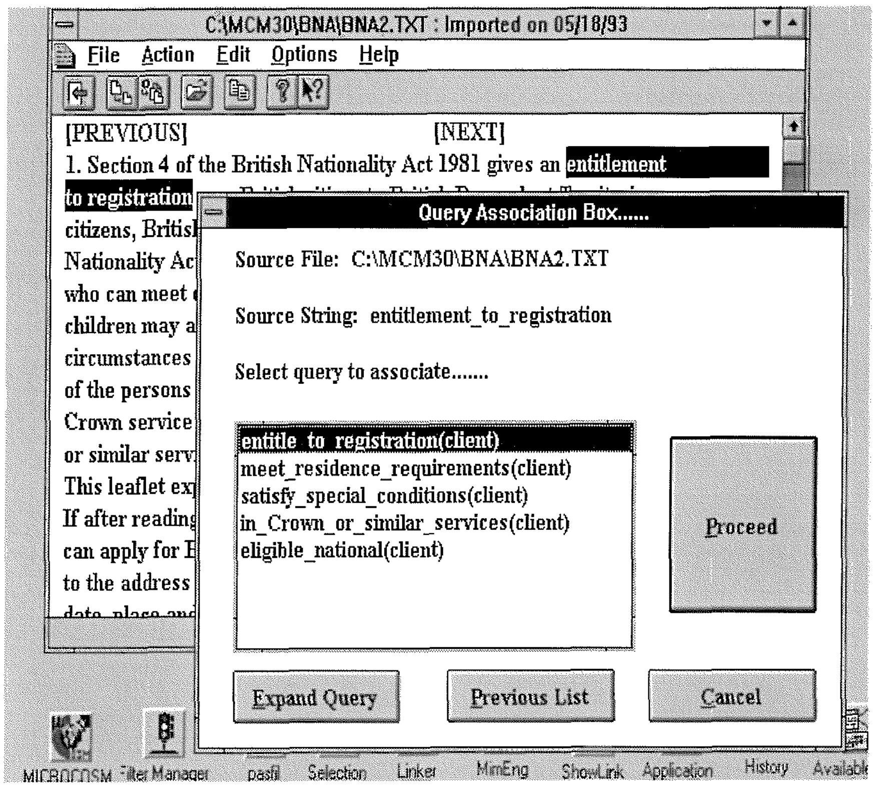
To start *Query-Concept Association*, the user highlights a string of characters in the displayed document (source document) and selects the 'Query' option from the Microcosm Action menu. In response the system displays the *Task Selection Box* (shown in Figure 8.5) and the user selects *Query-Concept Association* option. Clicking 'OK' to the *Task Selection Box* informs the system about the user's intention. In future we will call this process *Task Initialisation*.

In response to *Task Initialisation*, the system displays the *Query Association Box*, shown in Figure 8.6. The details of this message box are as follows.

Source File: Name of the document from where the string is selected.

Source String: String selected from the document. (The system replaces the extra characters - such as extra spaces, carriage return, line feed - with an under score.)

List Box: Displays a list of top-level queries from which users can select one.



[PREVIOUS] [NEXT]
1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration
citizens, Britis
Nationality Ac
who can meet
children may a
circumstances
of the persons
Crown service
or similar serv
This leaflet ex
If after readin
can apply for
to the address

Source File: C:\MCM30\BNA\BNA2.TXT

Source String: entitlement_to_registration

Select query to associate.......

entitle_to_registration(client)
meet_residence_requirements(client)
satisfy_special_conditions(client)
in_Crown_or_similar_services(client)
eligible_national(client)

Figure 8.6: Query association box

Expand Query Button: Expands a selected query (discussed in Section 5.2).

Previous List Button: Comes back to the previous query list.

Proceed Button: Informs the system to proceed.

Cancel Button: Cancels *Query-Concept Association.*

The system stores the information supplied through the *Query Association Box* in the Label Base (explained in Section 7.5.1) and successfully completes the *Query-Concept Association*. Now a query is linked with the selected string. The attached query can be initiated for two purposes: conceptual dynamic linking (see Section 8.5) and active document features (see Section 8.6).

The system allows users to link a single query with different wording of a concept. For example, a user might link the query `entitle_to_registration(Client)` with a concept appearing in the same or a different document with wording *"entitlement to registration", "eligible for registration", "British citizenship registration conditions".* In this case the system will consider all three linked concepts as the same. This offers a flexible method for handling synonyms or different wording of a concept.

To demonstrate the functionalities of the prototype we suppose that in the current session, using the explained *Query-Concept Association* procedure, the user linked two concepts with their relevant kb queries from document '$\backslash BNA\backslash BNA2.TXT$' as shown in Table 8.1.

Selected Concept	**Document Name**	**Attached kb Query**
entitlement to registration	$\backslash BNA\backslash BNA2.TXT$	entitle_to_registration(client)
residence requirements	$\backslash BNA\backslash BNA2.TXT$	meet_residence_requirements(client)

Table 8.1: Query-concept links established in the current session

8.4 System Up-Gradation

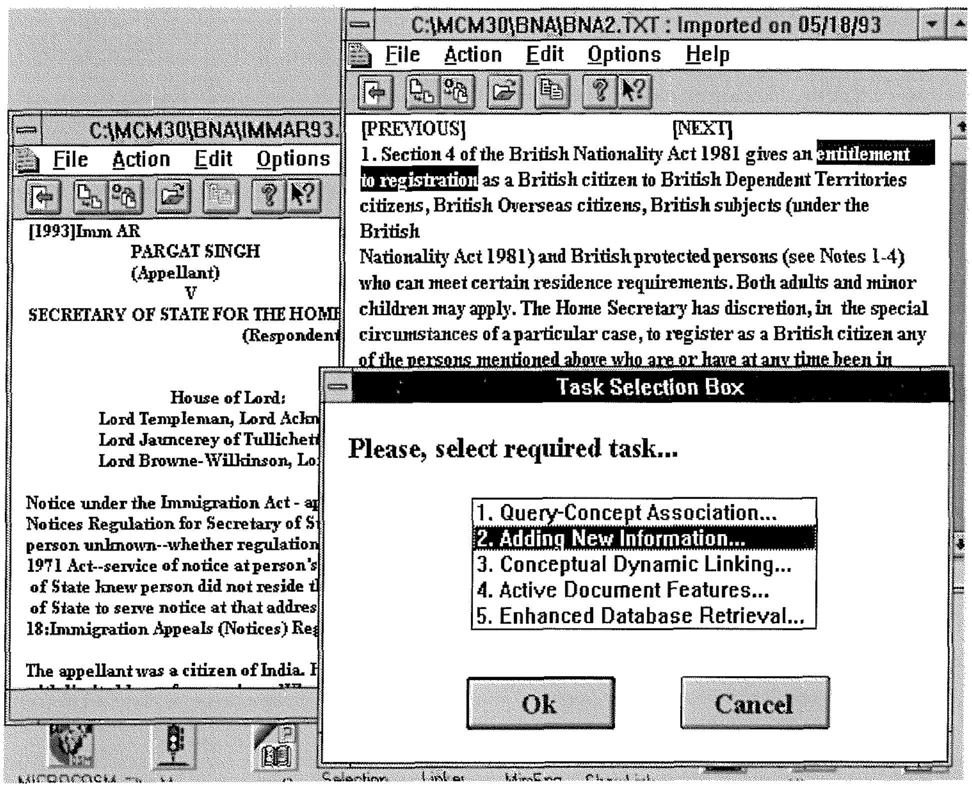
The varying needs of users and rapid growth of legal information make it important for an effective legal information management system to provide a flexible environment for system up-gradation. There are two forms of system up-gradation: up-grading the system by adding new information; and up-grading the system by adding new functionalities to meet further user demands. Most existing systems offer some form of the first option, but the second option is not usually provided. Our prototype provides a reasonably easy method for both types of up-grading.

8.4.1 Adding New Information

Suppose a user wants to link a new case report (destination document) with the concept *entitlement to registration* appearing in the document, '$\backslash BNA\backslash BNA2.TXT$', of the leaflets.

The new case report - [1993] Imm AR - is about an appellant (Pargat Singh) who was a citizen of India. He had been admitted to the United Kingdom as a visitor in February 1983 for a period of three months and went to ground. The Secretary of State decided to initiate deportation proceedings. The appellant sought to challenge the validity of the deportation order. Application for judicial review was dismissed.

It is important to note that the case selected for this exercise is not about a client who applied for registration. Instead it is about a client who was 'in breach of law'


C:\MCM30\BNA\IMMAR93.

[1993]Imm AR
PARGAT SINGH
(Appellant)
V
SECRETARY OF STATE FOR THE HOME
(Respondent

House of Lord:
Lord Templeman, Lord Ackn
Lord Jauncerey of Tullichett
Lord Browne-Wilkinson, Lo

Notice under the Immigration Act - ap
Notices Regulation for Secretary of St
person unknown--whether regulation
1971 Act--service of notice at person's
of State knew person did not reside th
of State to serve notice at that addres
18:Immigration Appeals (Notices) Reg

The appellant was a citizen of India. H

C:\MCM30\BNA\BNA2.TXT : Imported on 05/18/93

[PREVIOUS] [NEXT]
1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories citizens, British Overseas citizens, British subjects (under the British
Nationality Act 1981) and British protected persons (see Notes 1-4) who can meet certain residence requirements. Both adults and minor children may apply. The Home Secretary has discretion, in the special circumstances of a particular case, to register as a British citizen any of the persons mentioned above who are or have at any time been in

Task Selection Box

Please, select required task...

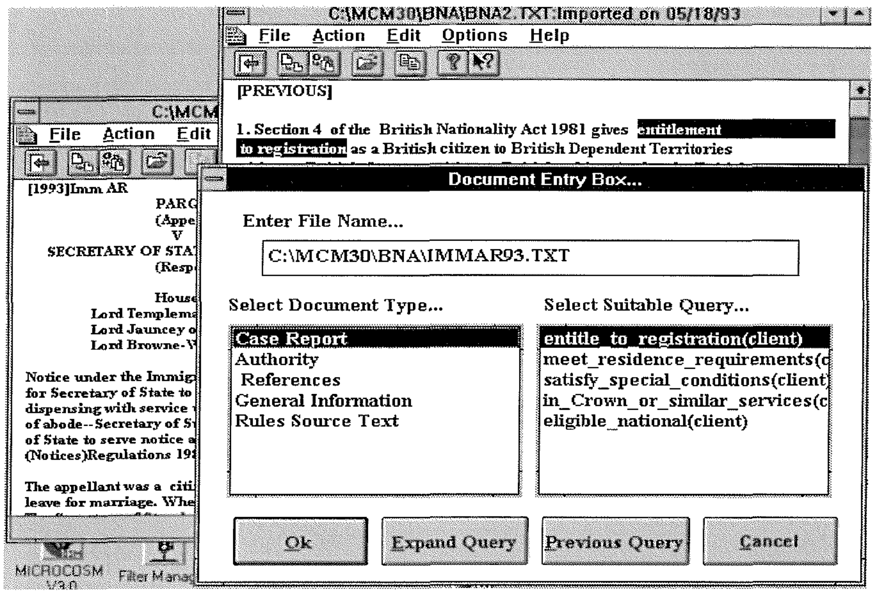
1. Query-Concept Association...
2. Adding New Information...
3. Conceptual Dynamic Linking...
4. Active Document Features...
5. Enhanced Database Retrieval...

Figure 8.7: A selected concept, a part of new case report and selected task.

and that caused his deportation. Just to refresh the reader's memory, in Section 5.3 we discussed that an important requirement for entitlement to registration is that the client should not be in breach of law. The concept 'in breach of law' is an open texture concept which could cause difficulties for a user to understand clearly. This case can help users to understand the meaning of the concept 'in breach of law'. One important aim of this exercise is to show that one case report could cover different aspects of an issue, so it needs to be labelled it accordingly. The system allows this. For example, this particular case can also be labelled through the concept 'deportation' and then the system will generate a label using the definition of the concept 'deportation'.

During *Task Initialisation*, the user highlights the concept from the document, '$\backslash BNA \backslash BNA2.TXT$', and selects the *Adding New Information* option from the *Task Selection Box*. Figure 8.7 shows (part of) the new case report, the selected concept from the document and the selection from the *Task Selection Box*.

In response to *Task Initialisation* the system displays the *Document Entry Box*, shown in Figure 8.8, asking the user for three types of information: the name of the file whose actual contents is the new document, its type and a suitable kb query. The user enters the file name and selects 'Case Report' as the document type and `entitle_to_registration(client)` as a suitable query from the query list box. The function of *Expand Query* and *Previous Query* buttons are the same as in *Query* Association Box (Figure 8.6).



1. Section 4 of the British Nationality Act 1981 gives entitlement to registration as a British citizen to British Dependent Territories

Enter File Name...

C:\MCM30\BNA\IMMAR93.TXT

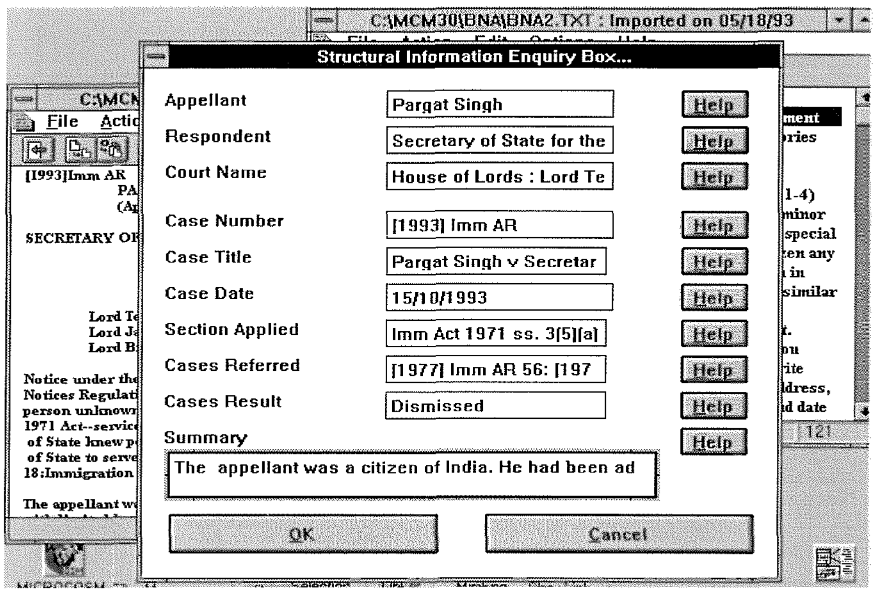
Select Document Type...

Case Report
Authority
References
General Information
Rules Source Text

Select Suitable Query...

entitle_to_registration(client)
meet_residence_requirements(c
satisfy_special_conditions(client
in_Crown_or_similar_services(c
eligible_national(client)

Figure 8.8: New document linking box


C:\MCM30\BNA\BNA2.TXT : Imported on 05/18/93
Structural Information Enquiry Box...
File Actio
Help
OK
Cancel

Appellant	Pargat Singh
Respondent	Secretary of State for the
Court Name	House of Lords : Lord Te
Case Number	[1993] Imm AR
Case Title	Pargat Singh v Secretar
Case Date	15/10/1993
Section Applied	Imm Act 1971 ss. 3[5][a]
Cases Referred	[1977] Imm AR 56: [197
Cases Result	Dismissed
Summary	The appellant was a citizen of India. He had been ad

Figure 8.9: Window for getting structured information of a case report

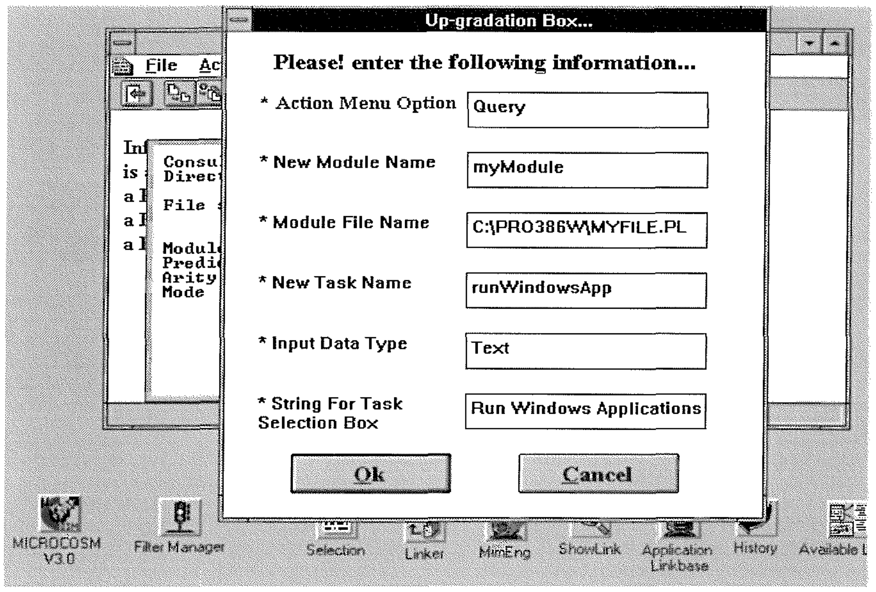
The system next asks for structured information about the case report through the message box shown in Figure 8.9. The fields in this box have been chosen as appropriate for the type 'Case Report'. The user (optionally) fills-in the information from their reading of the new case report. After getting this information, the system runs the selected query in a dialogue session with the user asking for any missing information required to satisfy the query. The user supplies the required information from the contents of the new case report according to his/her understanding. At the end of this dialogue session the system generates a proof tree label for the new case report and stores the labelling information in the Label Base as discussed in Section 7.5.2. The new case report is now entered into the system and is available for conceptual dynamic linking (see Section 8.5) and for structured retrieval (see Section 8.7) .

The process we have just described, giving a label to a new document, effectively links the new document into the existing hypermedia in a conceptually based way. During this process the system extracts the conceptual and structured information from the new case report with the help of the user and keeps it for future uses. This puts extra load on the user but in Section 8.5, 8.5.2, and 8.6 we will see that this small extra load makes the whole retrieval process very effective.

8.4.2 Adding New Functionalities

Although a full definition of an open system has not yet been given, it is commonly accepted that an open system should allow end-users to add new functionalities [Tannenbaum 1992]. Very few existing systems allow this. Our prototype system is relatively unrestricted and provides a reasonably easy up-grading environment for end-users. Users can build new modules and make them available to the system as new functionalities. To do so, during the system initialisation process, users select the 'Up-Grade' button from the *Module Selection Box* (shown in Figure 8.2). In response, the system displays the message box, shown in Figure 8.10. The function of the fields in this box are explained below.

Action Menu Option: Microcosm Action menu option which will generate a message for *kbh_Filter*. It must be one of the newly added options, 'Query' or 'Answer', as *kbh_Filter* does not recognise messages generated by other options.


Up-gradation Box...

Please! enter the following information...

* Action Menu Option	Query
* New Module Name	myModule
* Module File Name	C:\PRO386W\MYFILE.PL
* New Task Name	runWindowsApp
* Input Data Type	Text
* String For Task Selection Box	Run Windows Applications

Figure 8.10: Message box for adding new modules information

Module Name: Name of the Module responsible of performing new task.

Module File Name: File name of the new module's Prolog code having top-level predicate

`taskFor(ModuleIdentifier, TaskIdentifier, InPut, OutPut)`

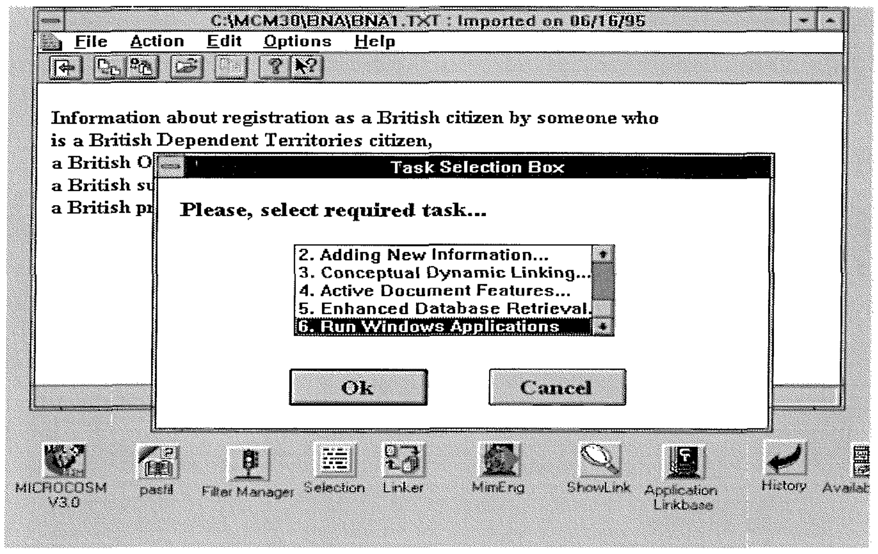
New Task Name: Task name declared in the new module with variable `TaskIdentifier`.

Input Data Type: Data type which the new module will receive from Microcosm.

String For Task Selection Box: String to be appeared in the *Task Selection Box*.

After getting this information the system adds the new option into the *Task Selection Box*, loads the new module and keeps the necessary information in the system's initialisation file.

We now give a practical demonstration of upgrading in the current session. Suppose the user wants to add a new facility to run other Windows applications from the Microcosm Action menu. During system initialisation, the user selects the 'Up-Grade' button from the *Module Selection Box* (shown in Figure 8.2), and fills in the information about his/her new module as shown in Figure 8.10. In response, the system adds the new task option, 'Run Windows Applications' to the *Task Selection Box*



Information about registration as a British citizen by someone who
is a British Dependent Territories citizen,
a British O
a British su
a British pr

Please, select required task...

2. Adding New Information...
3. Conceptual Dynamic Linking...
4. Active Document Features...
5. Enhanced Database Retrieval.
6. Run Windows Applications

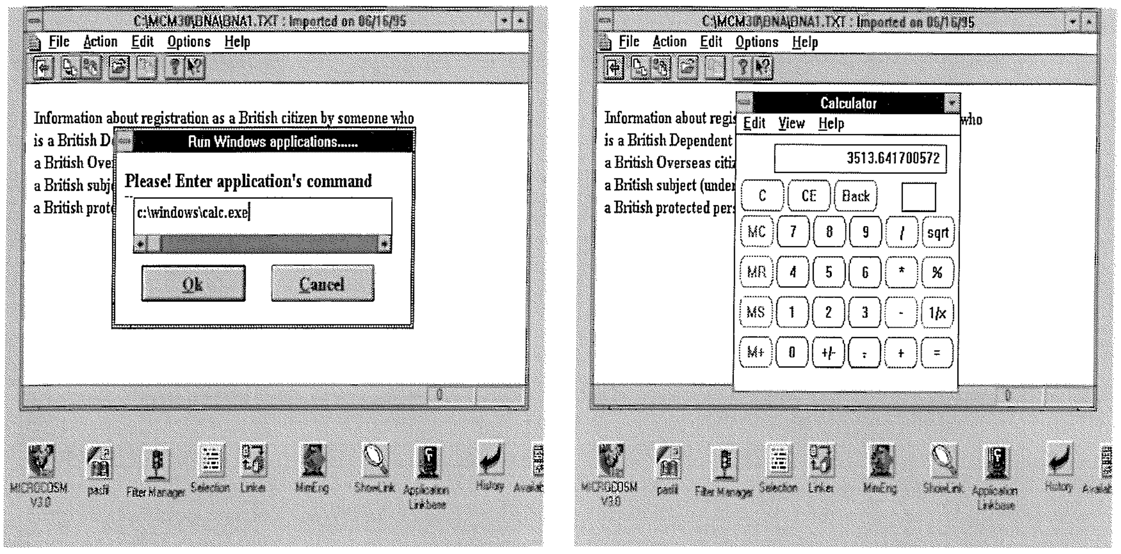
Figure 8.11: Newly added option in Task Selection Box

(as shown in Figure 8.11) and loads the new module. The system is now ready to run Windows applications from the Microcosm Action menu. The Prolog code of the new module `myModule` is given below:

```
taskFor(myModule, runWindowsApp, InPut, OutPut):-
                  runWindowsApp(InPut, OutPut).
runWindowsApp(InPut,OutPut):-
                  askCommandLine(AppCommandLine),
                  run(AppCommandLine).
run(AppCommandLine):-
                  exec(AppCommandLine,'',_).
```

In this example, the new module does not have any input or output. The predicate `askCommandLine/1` asks the user the command line of the application to be run and PROWIN's predicate `exec/3`, executes that command line.

For example, suppose in *Task Initialisation* the user selects the newly added option 'Run Windows Applications'. The system asks for the command line of the application to run. The user might enter the command line of the Windows calculator program. In turn the system would run the calculator. Figure 8.12 shows this demonstration. This practical example not only supports our claim of easy up-gradation and system openness, but also shows the strength of the *kbh* framework for adding additional



Information about registration as a British citizen by someone who
is a British D
a British Ove
a British subj
a British prot

Please! Enter application's command

c:\windows\calc.exe

Information about regis
is a British Dependent
a British Overseas citi
a British subject (unde
a British protected per

3513.641700572

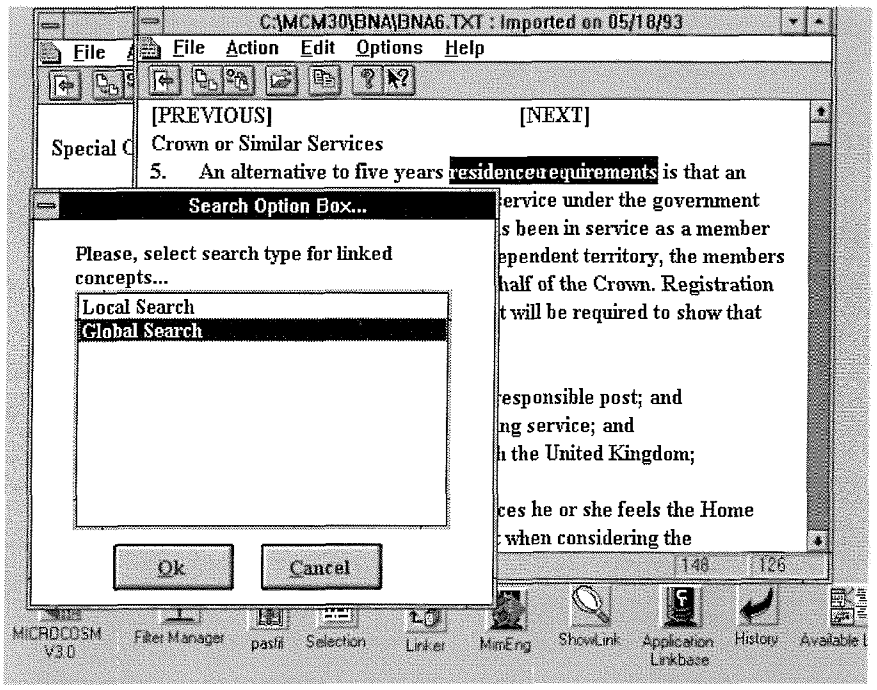
Figure 8.12: Message box asking application command line and Window application 'calculator' executed from Microcosm

functionalities.

8.5 A Generic Approach to Conceptual Dynamic Linking

In this section we demonstrate conceptual dynamic linking in the prototype. Suppose the user is reading the document ('$\backslash BNA\backslash BNA6.TXT$') of the leaflet shown in Figure 8.13 and wants to retrieve documents related to the concept *residence requirements*.

During *Task Initialisation*, the user selects the concept *residence requirements* and selects the option *Conceptual Dynamic Linking* from the *Task Selection Box*. In response, the system asks the user about the search type for the concepts, present in the selected string, which have been linked with kb queries. The system offers two options, shown in Figure 8.13, *Local Search* and *Global Search*. In *Local Search*, the system displays only those concepts, present in the selected string, which have been linked with kb queries from the source document. In *Global Search*, the system displays all those concepts, present in the selected string, which have been linked with kb queries from any document. Suppose for this session, the user selects the *Global Search* option. In turn, the system finds such concepts, as discussed in Section 5.2, and displays them for user selection. Figure 8.14 shows that one concept is present in the selected string which has


C:\MCM30\BNA\BNA6.TXT : Imported on 05/18/93

[PREVIOUS] [NEXT]

Crown or Similar Services

5. An alternative to five years residenceu equirements is that an

Search Option Box...

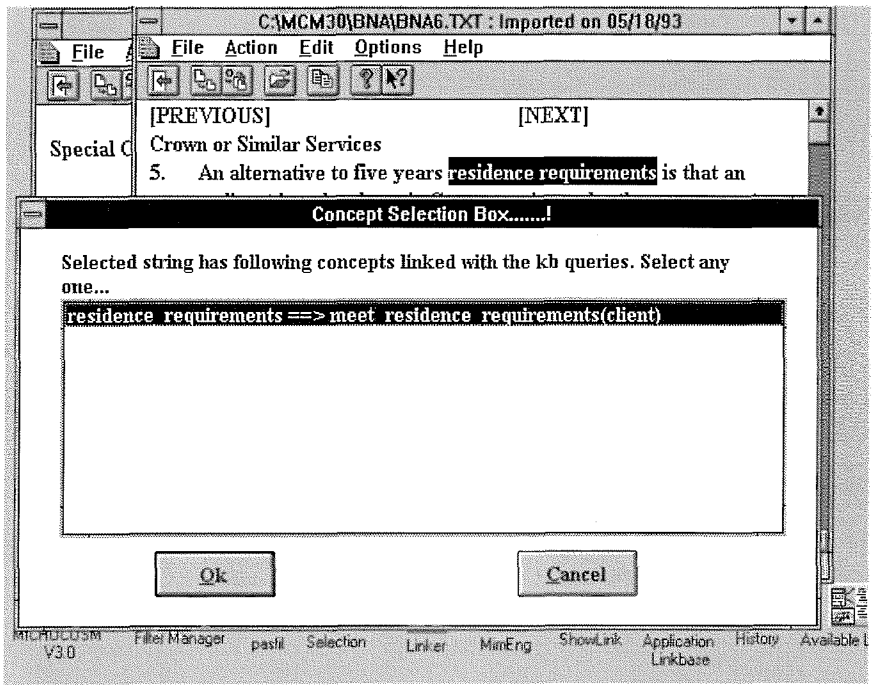
Please, select search type for linked concepts...

Local Search

Global Search

Figure 8.13: Local or Global search option.

already been linked with a kb query. Readers should note that the user has not linked the concept *residence requirements*, from the source document ('$\backslash BNA\backslash BNA6.TXT$'),


C:\MCM30\BNA\BNA6.TXT : Imported on 05/18/93

[PREVIOUS] [NEXT]

Crown or Similar Services

5. An alternative to five years residence requirements is that an

Concept Selection Box.......!

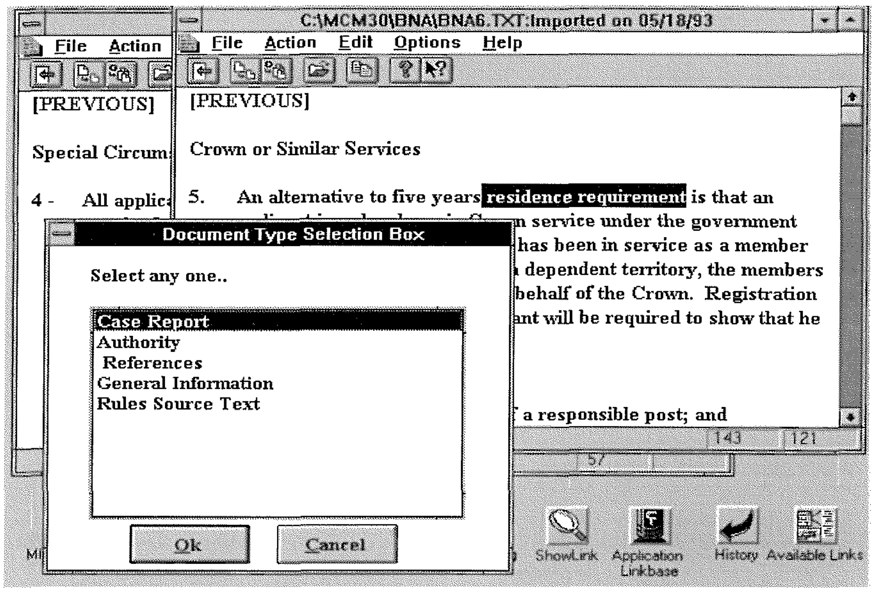
Selected string has following concepts linked with the kb queries. Select any one...

residence requirements ==> meet residence requirements(client)

Figure 8.14: Retrieved concepts from the selected string.

with any kb query. But the system informs the user that the selected string has one concept which has been linked with a kb query. This is the concept which the user linked with the kb query in Section 8.3 from the document '$\backslash BNA\backslash BNA2.TXT$' (see Tabel 8.1).

Continuing the user can select the displayed concept. The system, in turn, runs the associated query with the selected concept in a dialogue session asking the user for missing information required to satisfy the query. The user answers according to the information in the document. At the end of this dialogue session the system generates a proof tree label for the query and asks the user about the type of documents required (see Figure 8.15). The user selects 'Case Report'. The system then searches the Label Base for relevant proof tree labels of labelled case reports, as discussed in Section 7.5.3, and displays retrieved case report titles along with their level of similarity to the required information in rank order (see Figure 8.16). The user might select new case report [1993] Imm AR and see the summary, shown in Figure 8.17, and later read the actual contents.



[PREVIOUS]

Special Circum

4 - All applic

[PREVIOUS]

Crown or Similar Services

5. An alternative to five years residence requirement is that an
n service under the government
has been in service as a member
dependent territory, the members
behalf of the Crown. Registration
ant will be required to show that he

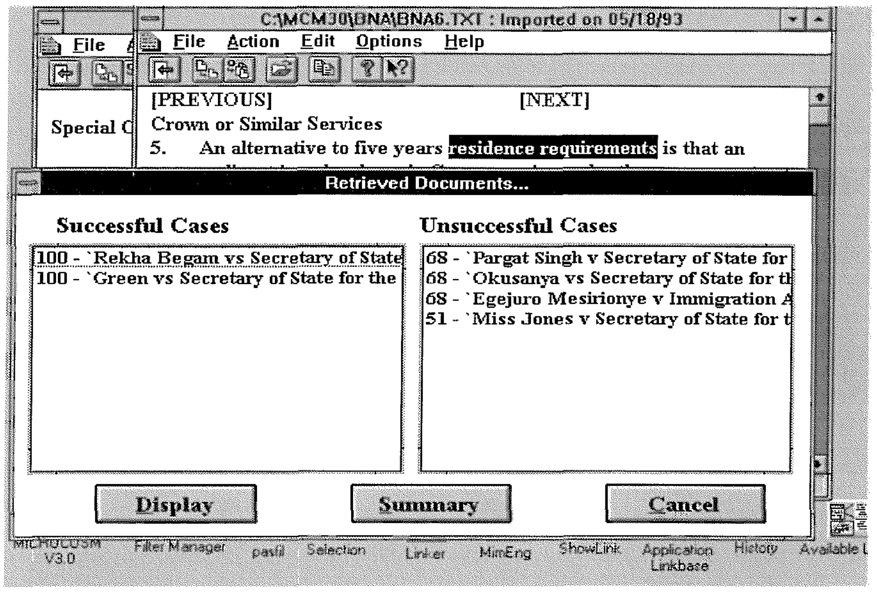
a responsible post; and

Select any one..

Case Report
Authority
References
General Information
Rules Source Text

Figure 8.15: Message box asking document type.

An important point to note is that the case report [1993] Imm AR was linked with the concept *entitlement to registration* from the document '$\backslash BAN\backslash BAN2.TXT$', but the system retrieved it with the concept *residence requirements* from the document '$\backslash BNA\backslash BNA6.TXT$'. It can be seen from the summary of the [1993] Imm AR,


C:\MCM30\BNA\BNA6.TXT : Imported on 05/18/93

[PREVIOUS] [NEXT]
Crown or Similar Services
5. An alternative to five years residence requirements is that an

Retrieved Documents...

Successful Cases	Unsuccessful Cases
100 - `Rekha Begam vs Secretary of State	68 - `Pargat Singh v Secretary of State for
100 - `Green vs Secretary of State for the	68 - `Okusanya vs Secretary of State for th
	68 - `Egejuro Mesirionye v Immigration A
	51 - `Miss Jones v Secretary of State for t

Figure 8.16: Message box displaying retrieved document names with level of similarity in rank order.

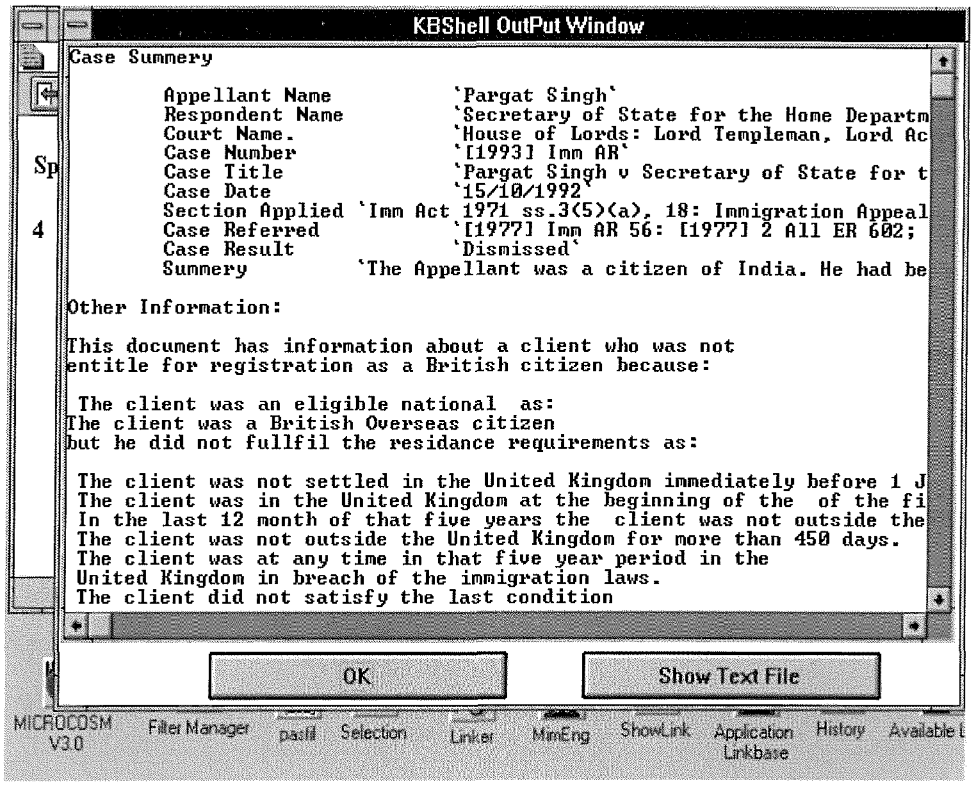
Figure 8.17, that it has information about residence requirements so the system has automatically retrieved relevant documents for the user.

So this practical example has shown that our generic approach for conceptual dynamic linking in the *kbh* framework is generic in the sense that it propagates links automatically between documents. Our approach has many advantages discussed in the next sections.

8.5.1 Reducing Linking Overhead for Users

One advantage of our approach is that it reduces the linking overhead for users. Concepts attached to lower level queries will automatically acquire links with documents which have been entered into the system through their top-level queries. For example `meet_residence_requirements(client)` is a lower level query for `entitle_to_registration(client)`. All documents entered into the system through this top-level query will automatically be linked to concepts which have been linked to the query `meet_residence_requirement(client)`. This certainly reduces the linking overhead for the users.

Another advantage is that if a particular wording of a concept, linked with a kb query from one document, appears in any other document, it will automatically be



```
Case Summery

        Appellant Name          `Pargat Singh`
        Respondent Name         `Secretary of State for the Home Departm
        Court Name.             `House of Lords: Lord Templeman, Lord Ac
        Case Number             `[1993] Imm AR`
        Case Title              `Pargat Singh v Secretary of State for t
        Case Date               `15/10/1992`
        Section Applied `Imm Act 1971 ss.3(5)(a), 18: Immigration Appeal
        Case Referred           `[1977] Imm AR 56: [1977] 2 All ER 602;
        Case Result             `Dismissed`
        Summery         `The Appellant was a citizen of India. He had be

Other Information:

This document has information about a client who was not
entitle for registration as a British citizen because:

 The client was an eligible national  as:
The client was a British Overseas citizen
but he did not fullfil the residance requirements as:

 The client was not settled in the United Kingdom immediately before 1 J
 The client was in the United Kingdom at the beginning of the  of the fi
 In the last 12 month of that five years the  client was not outside the
 The client was not outside the United Kingdom for more than 450 days.
 The client was at any time in that five year period in the
 United Kingdom in breach of the immigration laws.
 The client did not satisfy the last condition
```

Figure 8.17: Showing summary of [1993] Imm AR.

linked to that kb query and thus allow conceptual retrieval. This certainly reduces the extra load of linking kb queries with similar concepts appearing in different documents. For example, if the concept *residence requirements* appears in five documents, the user just links this concept from one document to a kb query, not from five documents.

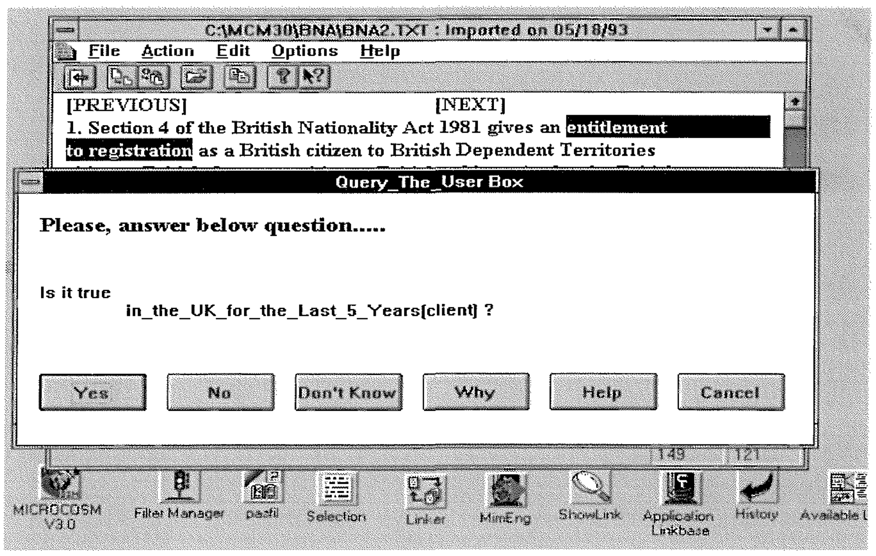
8.5.2 Displaying Link Information and Automatic Summary Generation

Figure 8.15 shows the way the system displays information about retrieved documents. It shows a percentage as a measure of similarity between the information stored in the retrieved document and the user requested information. This helps users to select a document for navigation. Figure 8.15 shows that the system found six documents related to user requested information; two of them had 100% similar information, three 68%, and one 51%. From Figure 8.17 it can be seen that the appellant of this case report, [1993] Imm AR, does not satisfy the residence requirements. However, the document discusses residence requirements. So our measure of similarity guides users to information present within retrieved documents, making navigation easier.

One advantage of extracting conceptual and structured information from a linked document is that the system automatically generates a summary of the retrieved document. The system produces this summary using structured and conceptual information about the documents. From Figure 8.17, it can be seen that this summary gives a conceptual picture of the retrieved document and that it has rich enough information about the document to make selection easier and also, possibly, to save users from reading the full text.

8.6 Active Document Features

In this section we demonstrate active document features of the *kbh* framework. Suppose the user is reading a document of the leaflet, shown in Figure 8.18, and wants to check whether he/she is entitled to registration. During *Task Initialisation* he/she selects the concept *entitlement to registration* and option *Active Document Features* from the *Task Selection Box*. In turn, the system runs the query attached to the selected concept leading to a dialogue session. The system asks the user for any missing information needed to satisfy the query; Figure 8.18 shows the QTU message box. If the user asks the system why the question is being asked, the system explains by displaying first the head of the rule in use and then (if requested) the body of the rule as shown in Figure 8.19.


C:\MCM30\BNA\BNA2.TXT : Imported on 05/18/93

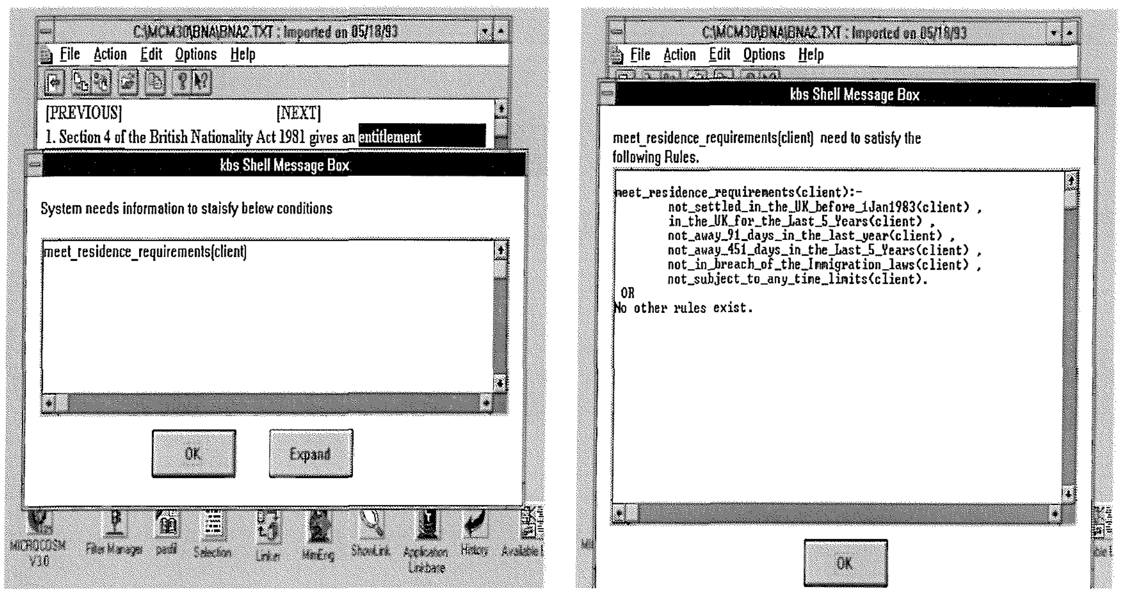
[PREVIOUS] [NEXT]
1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories

Query_The_User Box

Please, answer below question.....

Is it true
in_the_UK_for_the_Last_5_Years[client] ?

Figure 8.18: QTU box



1. Section 4 of the British Nationality Act 1981 gives an entitlement

kbs Shell Message Box

System needs information to staisfy below conditions

meet_residence_requirements(client)

kbs Shell Message Box

meet_residence_requirements(client) need to satisfy the following Rules.

```
meet_residence_requirements(client):-
        not_settled_in_the_UK_before_1Jan1983(client) ,
        in_the_UK_for_the_Last_5_Years(client) ,
        not_away_91_days_in_the_last_year(client) ,
        not_away_451_days_in_the_Last_5_Years(client) ,
        not_in_breach_of_the_Immigration_laws(client) ,
        not_subject_to_any_time_limits(client).
 OR
No other rules exist.
```

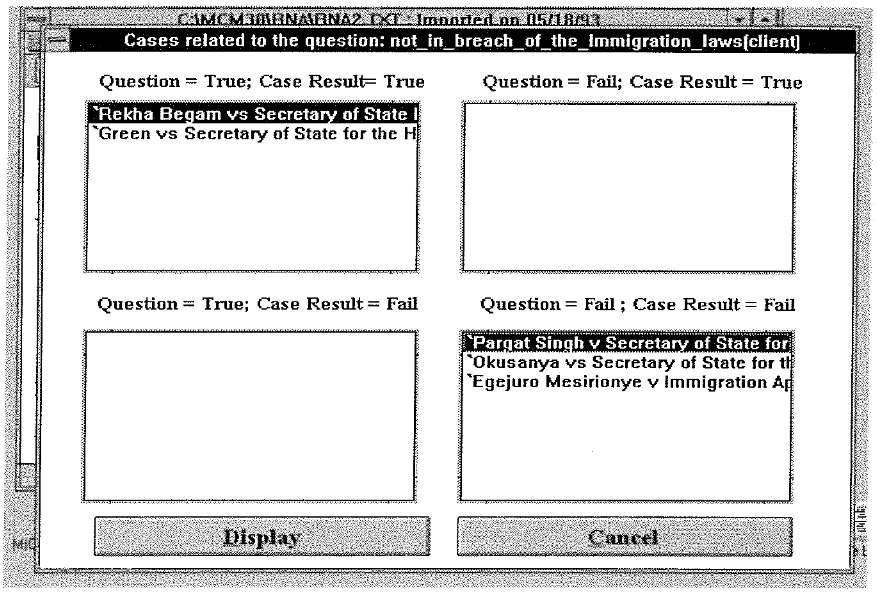
Figure 8.19: Message box explaining 'Why'

If the user is not sure about the answer to a question he/she selects the button **Don't Know**. The system considers this answer conditionally true and proceeds ahead. (In this case the system will generate a conditional answer. This feature [Wolstenholme 1989] is not documented for the shell described in Chapter 7, but it was implemented.) Suppose the system asks the user another question,

```
Is it true?
not_in_breach_of_the_Immigration_Law(client)
```

and the user can not understand the question clearly then he/she can ask the system to give help by selecting **Help** from the QTU box. The system asks the user about the document type wanted (as shown in Figure 8.15). If the user selects 'Case Report', the system retrieves case reports related to this particular question from the Label Base and displays the titles of the retrieved case reports as shown in Figure 8.20 (as discussed in Section 7.3). In this box the system does not show the measure of relevancy, but splits up the retrieved cases into four categories which helps users to decide which case reports to read. If the user selects two cases shown in Figure 8.20 the system displays the summaries of the selected cases one by one. While reading the summaries the user may ask the system to display the actual contents of the case reports. After reading these he/she may get a clearer idea about the original question and answer

'Yes' to it, say. At this point the system displays the *Reference Selection Box* shown in Figure 8.21 allowing the user to select from the visited documents those which he/she considers were relevant. The system keeps these as references and proceeds. At the


Cases related to the question: not_in_breach_of_the_Immigration_laws(client)

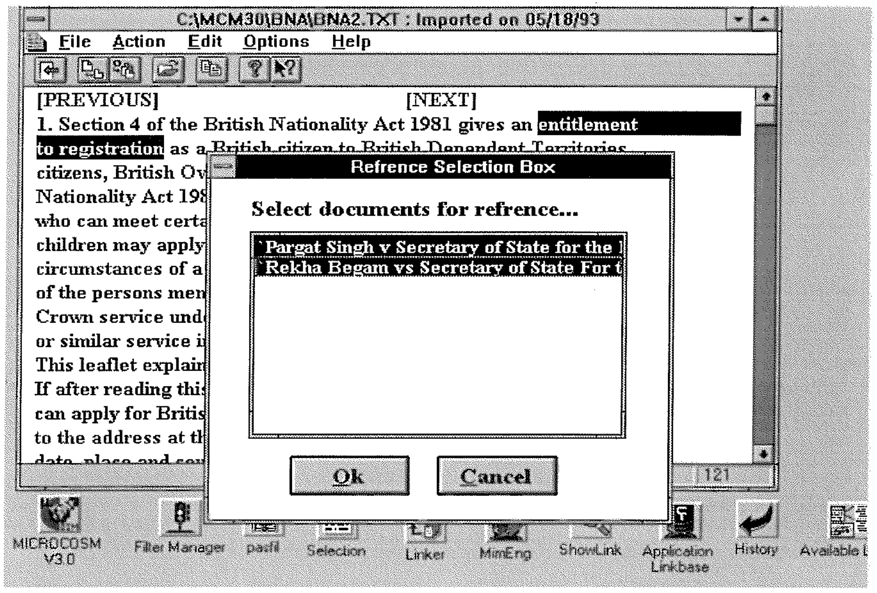
Question = True; Case Result= True
`Rekha Begam vs Secretary of State
`Green vs Secretary of State for the H

Question = Fail; Case Result = True

Question = True; Case Result = Fail

Question = Fail ; Case Result = Fail
`Pargat Singh v Secretary of State for
`Okusanya vs Secretary of State for t
`Egejuro Mesirionye v Immigration A

Figure 8.20: QTU help cases



[PREVIOUS] [NEXT]
1. Section 4 of the British Nationality Act 1981 gives an entitlement to registration as a British citizen to British Dependent Territories citizens, British Ov
Nationality Act 19
who can meet cert
children may apply
circumstances of a
of the persons me
Crown service und
or similar service i
This leaflet explai
If after reading thi
can apply for Britis
to the address at t

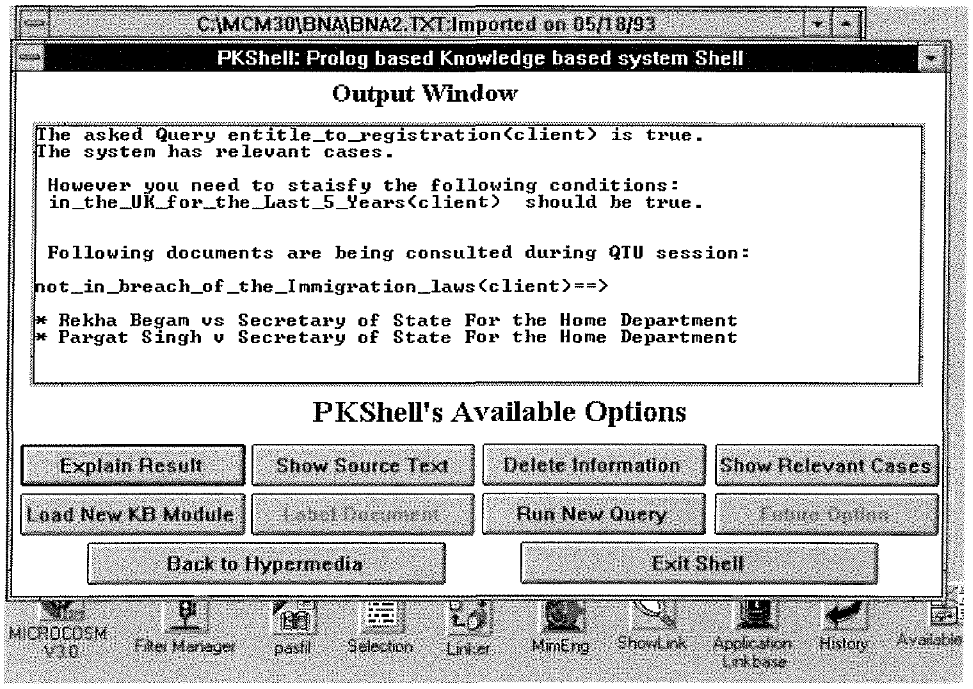
Refrence Selection Box

Select documents for refrence...

`Pargat Singh v Secretary of State for the
`Rekha Begam vs Secretary of State For t

Figure 8.21: Ask to select referred doc

end of this dialogue session the system displays the result of the query in the output window shown in Figure 8.22.


C:\MCM30\BNA\BNA2.TXT:Imported on 05/18/93
PKShell: Prolog based Knowledge based system Shell
Output Window

```
The asked Query entitle_to_registration(client) is true.
The system has relevant cases.

 However you need to staisfy the following conditions:
 in_the_UK_for_the_Last_5_Years(client)  should be true.

 Following documents are being consulted during QTU session:

not_in_breach_of_the_Immigration_laws(client)==>

* Rekha Begam vs Secretary of State For the Home Department
* Pargat Singh v Secretary of State For the Home Department
```

Figure 8.22: Result options output window box

Three things are worth noting in the output window shown in Figure 8.22.

1. The message 'The system has relevant cases.' is displayed under special circumstances. Before displaying the result, the system tries to match the current query proof tree with the proof tree labels in the Label Base. If it finds exactly the same proof tree label with the same result, it displays the shown message. If on the contrary it finds the same proof tree label with opposite result, it displays a message to the users that it is a hard case, since an old case did not succeed under similar circumstances. In this way the system provides a basis for a case-based reasoning environment.

2. The system generates conditional answers. It informs users that the answer is only applicable if the user satisfies the conditions being considered conditionally true.

3. The system displays the names of the documents referenced during QTU session. This might be very useful for law professionals of the domain, for preparing court trials, as they can refer to these cases in court.

The result output message box has many options to proceed, as discussed in next section.

8.6.1 Options Related To Displaying Query Results

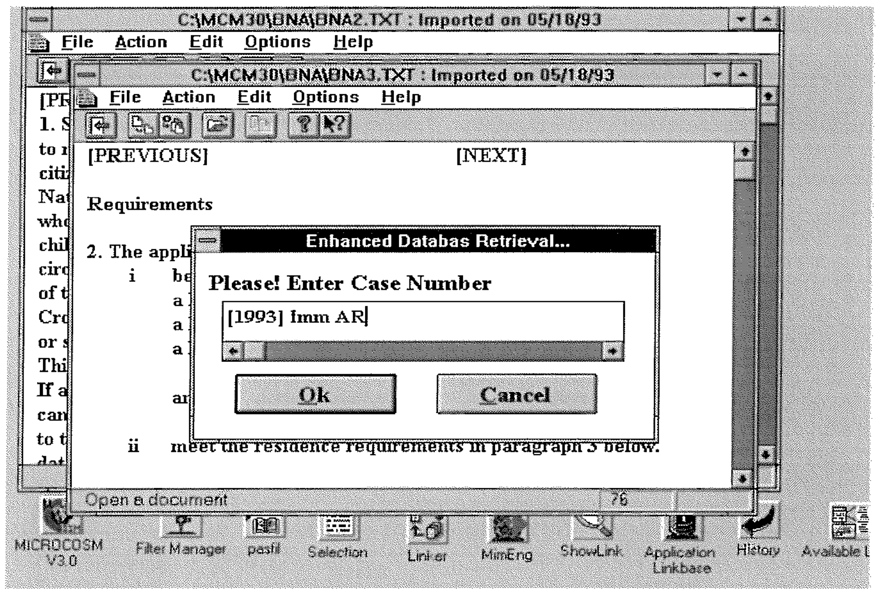
In this section we explain the functions of the available options to proceed after getting the result of a query.

- *Explain Result:* This option displays the proof tree of the processed query.
- *Show Source Text:* The system displays the source document of the applied rule. These are the documents which are labelled selecting its type as 'Rules Source Text' in the *Document entry Box* (see Figure 8.15).
- *Delete Information:* This allows users to delete information told during the QTU session.
- *Show Relevant Cases:* The system retrieves relevant cases from the Label Base by matching the query proof tree with stored proof tree labels as discussed in Section 7.5.3. Users can see the summaries first and actual contents later.
- **Load New Kb Module:** This allows users to load new kb modules, so that the system may have different modules having knowledge according to different point of views.
- *Run New Query:* This allows users to run a new kb query.
- *Back to Hypermedia:* This gives control back to the hypermedia system and users can continue their navigation from where they have submitted the query for the current session.
- *Future Option:* As the system provides a dynamic research environment, we provided this for implementing new ideas.

These options help users to understand the results generated by the system. We believe that the active document features along with these facilities provide an effective advisory system environment.

8.7 Structured Retrieval

In this section we discuss the enhanced data base retrieval facility of the *kbh* framework. Suppose the user has entered new case report [1993] Imm AR and he/she wants to see other cases relevant to this particular case. During *task initialisation* he/she selects the option *Enhanced Database Retrieval* and in response the system displays the message box asking for the search key as shown in Figure 8.23. The user can enter the case number '[1993] Imm AR' and in turn the system will search the Label Base for any entry which has that case number. As the user has entered this case recently the system picks the proof tree label of this case report and finds other similar proof tree labels from the Label Base as discussed in Section 7.5.5. At the end of this process, the system shows the titles of retrieved case reports along with a measure of similarity in rank order (see Figure 8.24).

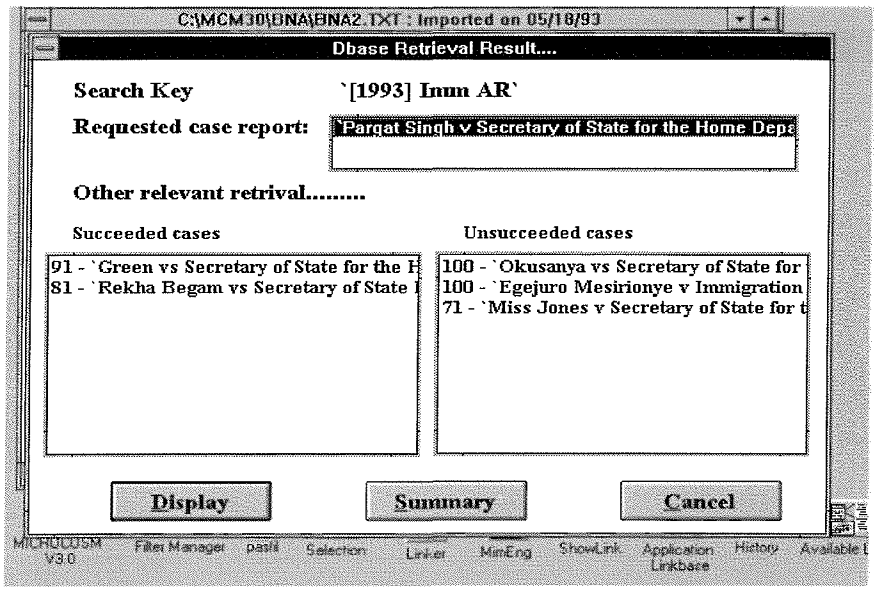

C:\MCM30\BNA\BNA2.TXT : Imported on 05/18/93
File Action Edit Options Help
C:\MCM30\BNA\BNA3.TXT : Imported on 05/18/93
File Action Edit Options Help
[PREVIOUS]
[NEXT]
Requirements
2. The appli
Enhanced Databas Retrieval...
Ok
Cancel
Open a document
MICROCOSM V3.0
Filter Manager
pastil
Selection
Linker
MimEng
ShowLink
Application Linkbase
History
Available L

Please! Enter Case Number

[1993] Imm AR

Figure 8.23: Dialogue box asking search key.

From Figure 8.24 it can be seen that the system not only retrieved the requested case, but also other cases which have similar kinds of information. This practical example indicates the effectiveness of our enhanced data base retrieval technique.


C:\MCM30\BNA\BNA2.TXT : Imported on 05/18/93

Dbase Retrieval Result....

Search Key `[1993] Imm AR`

Requested case report: `Pargat Singh v Secretary of State for the Home Depa

Other relevant retrival.........

Succeeded cases

91 - `Green vs Secretary of State for the H
81 - `Rekha Begam vs Secretary of State f

Unsucceeded cases

100 - `Okusanya vs Secretary of State for
100 - `Egejuro Mesirionye v Immigration
71 - `Miss Jones v Secretary of State for t

Figure 8.24: Dialogue box displaying found case reports

8.8 Conclusion

In this chapter we have discussed a prototype application developed for British nationality advice leaflets. The application, on the one hand, offers an integrated environment for information exploration which solves some of the problems related to these leaflets such as, cross references and repetition of information. On the other hand, it provides an advisory system like environment which allows users to see the practicality of available information. A particular feature is its 'Help' facility during QTU dialogue which retrieves relevant information dynamically to help users to understand open texture concepts. We believe this kind of application is not only useful for the general public but also for advice workers who can use it as a litigation tool.

In this prototype application we demonstrated the openness of the system which makes it useful in a practical world. Users can up-grade the system in either of two ways, adding new information and adding new functionalities. These features allows end-users to upgrade the system as new information arrives and add new functionalities to meet their further requirements.

One particular feature of the prototype is that it allows users to retrieve information dynamically on conceptual grounds. Since the system retrieves this information conceptually the chances of irrelevant retrieval are reduced. At the same time its generic retrieval feature reduces the manual linking overhead from users. The system displays the retrieved information with a percentage which is a measure of relevancy between the information asked and the information retrieved. The prototype also generates a summary of the retrieved documents automatically. Both these facilities help users to select documents for navigation. The system provides useful features for displaying results of evaluated queries. These features help users to understand the generated results. Additionally, the system's enhanced database retrieval facility makes the retrieval feature more effective. We believe such features help users to understand the behaviour of the system as well as increasing the applicability and usability of the prototype.

On the basis of experience from the case study, we would like to argue that the *kbh* framework along with its information management tools has wider scope for other domains. In future we want to conduct fuller case studies in different domains such as medicine, education, industry and law.

9 Conclusion and Future Work

9.1 Conclusion

What this thesis set out to do was to address some of the particularly acute problems of information management in the context of burgeoning quantities of on-line information. It has concerned new proposals for the management of on-line information by combining the strengths of two information management technologies, hypermedia and kbs. We chose to select these two from the three basic technologies, information retrieval systems (IRS), hypermedia and kbs. We argued that hypermedia, despite its limitations, is a powerful technology for handling various types of unstructured on-line information; also that it seems to be clear from its popularity that the technology is likely to be a front-runner in the future information management systems. As regards kbs we argued that it is an essentially complementary technology, having strength in handling structured information and offering reasoning facilities which can directly support hypermedia technology. Whereas, IRS, on the other hand, we argued was an essentially orthogonal technology which can be added in a straightforward way to both hypermedia and kbs.

In Chapter 2, reviewing the hypermedia technology, we mentioned that the main strength of hypermedia systems lies in *links* but, at the same time, these *links* constitute the main problem of hypermedia systems giving rise to disorientation and cognitive overhead. In this chapter, discussing some current proposals such as dynamic linking through string search and conceptual association through layered models, we illustrated that such proposals are not suitable for handling complex information (e.g. legal information). We argued that lack of conceptual information about the contents of linked documents is the heart of the problem. We concluded that hypermedia systems need support from other technologies which allow implementation of alternative techniques to solve their problems.

In this context our proposal is focussed on developing kbs support for hypermedia systems. In Chapter 3, we proposed a general purpose framework, knowledge based hypermedia (*kbh*), in which a logic programming (LP) based kbs and a hypermedia system are loosely coupled into a composite system. In the *kbh* framework our main contributions are the development of kbs shell which contains relatively domain independent information management tools and implements a bi-directional communication channel (BCC) between component systems.

The *kbh* framework has two distinctive features, a modular based open architecture and a bi-directional communication between component systems. Comparing with other existing approaches for combining kbs and hypermedia technologies, we argued that our framework can offer two practical advantages which the others do not. First, the bi-directional communication can enhance the applicability of each of its components. The combined technologies can exploit the functionalities of each other on a run-time basis. Second, the modular based open architecture approach offers a reasonably easy method to upgrade the framework and a relatively domain independent environment - the hypermedia manager and the kbs shell remain domain independent, whereas the hypermedia document base and kbs knowledge base (kb) are developed according to the application. It also allows other features to be added within the framework.

In Chapter 4, we proposed our Proof tree based Conceptual Retrieval (PCR) technique. Among others, it is one of the main outcomes of this research and a key information management tool in the *kbh* framework. It provides a semi-automatic method of extracting conceptual knowledge from documents and user's requests and allows conceptual dynamic linking and enhanced database retrieval in hypermedia systems. The idea of PCR is implemented using four important features of LP: Horn clauses for building the dictionary of general concepts; LP meta-level programming; QTU for extracting conceptual information from user requests and documents; and proof trees for storing conceptual information about the destination nodes within the bodies of links. Matching these proof tree labels, the system retrieves documents which contain relevant information. For matching these labels we designed a proof tree matching algorithm. By matching predicate names of proof tree nodes and using a natural, but somewhat arbitrary, weighing scheme the matching algorithm generates a number as a measure of similarity between proof tree labels.

We argued that finding information by just comparing predicate names in proof trees can retrieve information at various levels of similarity down to a very fine grain. So designing a general purpose information matching scheme just considering the predicate matching is appropriate. Explaining our weighing scheme we pointed out the resulting percentage matching weight gives guidance to users about the degree of similarity of retrieve documents. We do not claim that our weighing scheme produces the best results. This is an open issue which needs more exploration. Nevertheless we do feel confident that any reasonable scheme should distinguish between different conditions of proof tree branches as ours does.

Probably the most important contribution of PCR to hypermedia is to offer conceptual dynamic linking. This is done by storing conceptual information about destination nodes within the bodies of links so that during the linking process the system links those documents which are conceptually similar/relevant to a user's request. There were two problems related to the use of PCR in hypermedia systems which arise immediately; how to inform users which definitions are available in the kb? and how to follow a link? In Chapter 5, we discussed a very general and flexible approach for handling these problems. Our approach offers three practical advantages. First, it handles problems related to different wording of concepts - attaching different wording of concepts with a single query. Second, it uses generic conceptual dynamic linking - same wording of a linked concept appearing in various documents automatically acquire links, the system computing these links dynamically. Third, it provides a basis for active documents features - a linked concept can initiate a relevant logical query just to see the practicality of that particular concept.

An enhanced database retrieval facility for structural retrieval in hypermedia system is another outcome of this research. We proposed that combining conventional structured retrieval techniques with PCR brings together two complementary retrieval ideas and therefore has the potential to produce considerably more effective results. On the database retrieval side it provides a conceptual retrieval environment which allows retrieval of relevant information semi-automatically. On the hypermedia side, it provides an enhanced database retrieval facility which certainly increases the strength of pure hypermedia systems because it does not ignore easily available and useful structured information - such as title, date and so on.

In Chapter 6, we discussed the implementation of a most crucial part of the *kbh* framework; the bi-directional communication channel (BCC). Its implementation made it possible to achieve the main objective of the composite architecture to allow the use of third party products instead of building our own hypermedia and kbs. The BCC builds a communication channel between the two other components, hypermedia and kbs shell implemented in two different language paradigms, 'C' and Prolog. It provides a message passing protocol through which these components can send and receive messages on a run-time basis. For its implementation we exploited the DDE and the DLL features of Windows, the filter based architecture of Microcosm (the hypermedia selected for the prototype), and the communication facilities of PROWIN (the Prolog selected for the prototype) with other languages. Using these features we built a new filter, *kbh_Filter*, a *kbh_Message Handler*, and introduced two new menu options to perform bi-directional communication between the hypermedia manager and the kbs shell.

Discussing our experiences with the implementation of the BCC we mentioned that the Windows DLL feature provided a very useful environment for exchanging data between different applications. We found that applications running under Windows can communicate bi-directionally if system developers provide publicly accessible DLLs which other application developers can use in their applications. In this way a bi-directional communication link can be built without touching the source code of an existing application. We argued that providing this kind of facility is worthwhile for the whole software industry and has many software engineering benefits. We also argued that if available applications satisfy some basic compatibility requirements, the strength of these applications can be combined into a single environment without the need for rewriting in a common language. It is clear that this saves time, money and human resources, and upgrading of the combined system will be much easier.

In chapter 7, we discussed the implementation of the kbs shell which is another major development in the *kbh* framework. Its development is heavily influenced by the expanded context of *kbh*. It is developed using a modular based logic programming approach. We chose the LP path as in this style of developing kbs's not only is a clear distinction between the domain knowledge, the kb, and the kbs shell maintained, but also the shell itself is specified in declarative style. Our experiences with the shell have indicated the usefulness of the modular approach primarily because it makes a logical separation between programs which perform different functions independently from each other. This modular style makes *kbh* an easily upgradeable framework.

Our prototype shell is not a single entity; rather it is an inference engine together with a variety of information management tools and a user interface. It has six main modules: Inference Engine, Message Analyser, Labeller, Matcher, User-Interface and System Library. All modules are developed using an LP meta-level programming approach. The Message Analyser is the coordinating module. It acts like a postmaster and provides a means of communication between the other modules. It is also responsible of handling user's requests submitted from the hypermedia manager to the kbs shell. The Inference Module is developed using meta-interpreter techniques. The interpreter used in our shell is adapted from [Sterling & Lakhotia 1988]. It generates proof trees for both succeeding and failing queries and provides QTU facility. We also proposed a flexible approach for providing help during QTU dialogue by dynamically retrieving documents having information relevant to questions asked by the system. This offers a flexible way to provide help with open texture concepts.

Finally, we discussed a case study carried out in the legal domain; advice leaflets for British nationality. The case study illustrates the following important features of the *kbh* framework and PCR.

- The prototype provides an easy way of attaching kb queries to selected concepts. Attaching a single query with different wording of a concept appearing in one document or different documents allow handling of problems related to synonyms. Also queries can be attached to concepts to explore their use in practice.

- The prototype provides two easy ways of system up-gradation; adding new information when it arrives, adding new functionalities without making any alteration in the main framework. The second option is not usually provided by the existing information management systems. This particular feature makes the *kbh* framework a suitable platform for researchers of different domains to prototype their new ideas easily. Another advantage is that the framework is a hybrid of an imperative language 'C' and a declarative language 'Prolog'. This allows researchers to exploit the functionalities of both languages. We mentioned, however, that the framework only supports those imperative languages which can use the Windows DLL facility.

- The generic approach to conceptual dynamic linking offers a practical advantage of reducing manual linking overhead. A particular wording of a concept, linked from one document but appearing in different documents, automatically acquires links. A newly entered document having this particular wording will automatically acquire the links. Also, a concept linked with a lower level kb query automatically acquires links with the documents which have been entered into the system through its top-level query.

- Showing a percentage as a measure of similarity between the information stored in a retrieved document and a user's requested information helps users to select a document for navigation.

- Automatically generated summaries of retrieved documents gives a conceptual picture of the retrieved document and has rich enough information to make selection easier. It also, possibly, saves users from reading the full text.

- Active document features allow users to explore the available information in practice. The 'Help' facility during a QTU dialogue uses a novel approach for helping users by retrieving relevant information dynamically. Our approach is more flexible than attaching text files or building simple questions related to open texture concepts. The prototype offers many options related to displaying query results including: explaining results by displaying proof trees, showing source texts of the rules used and retrieving case reports relevant to the current result. These options build an effective environment for users to understand system generated results.

- The enhanced database retrieval facility offers two main advantages. Firstly, it adds a structural retrieval environment to pure hypermedia systems. Secondly, it not only retrieves requested documents but also others which have relevant information.

In general we believe that the features discussed above show that *kbh* offers an effective environment for information management and makes a significant improvement over kbs or hypermedia alone. On the basis of experiences learnt from the two mini-case studies, we can say that the *kbh* framework along with its information management tools has wider scope for application in other domains like medicine, education and industry.

9.2 Future Research

The main contribution of this thesis has been to propose and implement the PCR framework for conceptual retrieval and point out its potential benefits. What we have not done is to carry out empirical studies to evaluate its sensitivity and effectiveness. The first task for future research will be to explore the properties of PCR.

The use of the full proof tree labels needs to be compared with the use of simpler labels which could be used instead. Various simpler labels can be envisaged, such as the set of predicates in the proof tree (or just the set of QTU predicates) or hierarchically organised sets of predicates. Because these labels are simpler than our proof tree, they can be expected to have efficiency advantages. A topic for future research is to explore the conceptual information carried by these simpler labels in relation to any efficiency gains. Associated with this, the stability of the proposed matching algorithm, and possible variants, should be investigated thoroughly.

The means of searching the Label Base for matching proof tree labels (or some of their variants) also warrants further study. The following points would be interesting to investigate.

- In future we want to explore alternative strategies for storing proof trees which can improve the speed of the matching algorithm. At the same time we want to explore new search strategies to reduce the search space (e.g. indexing or partitioning). At present we use a sequential method to find matching labels. In future we want to explore other strategies such as parallel search and partitioning of the Label Base.

- Currently we are only using backward chaining. In future we want to explore the feasibility of combined forward and backward chaining for retrieval. In this regard we want to use form filling techniques for user input and then according to the supplied information the system will find relevant information.

- A problem with storing labels for each case report and matching each proof tree is that it makes the system slow. It seems obvious that if the Label Base has a large number of labels then the system will become very slow. We have tried to develop another technique for storing labels. In place of keeping separate labels for each document we tried to use a common proof tree label for all those documents which have 100% matching proof trees. This reduces the search space and retrieval becomes faster. One problem is that each document has different structured information such as case number and so on. A common label can not store such information. Saving this information separately from proof tree labels is one possible solution. The system only consults structured information about documents during summary generation and enhanced database retrieval. We want to explore other methods to make retrieval faster since efficiency is important for practical systems.

Another task which this thesis has left to future research is to evaluate the usefulness of our proposed framework in practice in realistic domains. Such research will need to explore further the usefulness of the *kbh* information management tools, especially PCR, in a realistic application and evaluate our system in relation to other approaches such as case-based reasoning systems. A key aspect will be to evaluate under practical conditions the claim that our system can deliver more precise retrieval than conventional methods. We are particularly interested to conduct fuller case studies in the following domains:

- Medicine - designing an intelligent tutoring system about shoulder joint for third world medical students.
- Law - designing a help system for law practitioners in the area of Islamic Inheritance Law.

The two suggestions above are specific applications of interest to us, but in general the application of the *kbh* framework and PCR to other areas of interest must be a high priority.

Finally we would like to investigate the claim in this thesis that combining PCR conceptual retrieval with more conventional techniques, including statistical retrieval and database retrieval, results in a significantly enhanced information management framework. This claim also needs to be tested in a realistic application.

Bibliography

AGOSTI, M., R. COLOTTI, & G. GRADENIGO 1991. "A Two-level Hypertext Retrival Model for Legal Data", *in Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrival,* pp. 315–325.

AGOSTI, M. & P. G. MARCHETTI 1992. "User Navigation in the IRS Conceptual Structure through a Semantic Association Function", *The Computer Journal,* 35 (3): 194–199.

AIL 1987. *Proceedings of The First International Conference on Artificial Intelligence and Law.*

AIL 1989. *Proceedings of the Second International Conference on Artificial Intelligence and Law.*

AIL 1991. *Proceedings of The Third International Conference on Artificial Intelligence and Law.*

AIL 1993. *Proceedings of the 4th International Conference on Artificial Intelligence and Law.*

AIL 1995. *Proceedings of the 5th International Conference on Artificial Intelligence and Law.*

AKSCYN, R., D. MCCRACKEN, & E. YODER 1988. "KMS: A distributed hypermedia system for managing knowledge in organization", *Communications of ACM,* 31 (7): 820–835.

ALLEN, L. E. & C. S. SAXON 1987. "Some Problem in Designing Expert Systems to Aid Legal Reasoning", *in Proceedings of The First International Conference on Artificial Intelligence and Law,* pp. 94–104.

Apple Computers 1987. *HyperCard User's Guide.* Apple Computer, Inc.

ARENTS, H. C. 1995. "Knowledge-Based Indexing and Retrieval of Hypermedia Information". *in* Rada, R. & Tochtermann, K., (eds.), *Expertmedia - Expert Systems and Hypermedia,* chapter 7, pp. 137–169. World Scientific, London.

ARENTS, H. C. & W. F. L. BOGAERTS 1993. "Concept-based retrieval of hypermedia information: From term indexing to semantic hyperindexing", *Information processing & Management,* 29 (3): 373–386.

ASHLEY, K. D. 1990. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals.* MIT Press, Cambridge.

ASHLEY, K. D. & V. ALEVEN 1991. "Toward an Intelligent Tutoring System for Teaching Law Students to Argue with Cases", *in Conference Proceedings of the Third International Conference on Artificial Intelligence and Law,* pp. 42–52.

BARRAGAN, J. & L. BARRAGAN 1991. "Knowledge Acquisition and Knowledge Base Refinement Problems in Developing the KBS Legal Expert System", *in Conference Proceedings of the 3th International Conference on Artificial Intelligence and Law,* pp. 196–200.

BENCH-CAPON, T. & M. SERGOT 1988. "Towards a Rule-Based Representation of Open Texture in Law". *in* Walter, C., (ed.), *Computer Power and Legal Language,* chapter 6, pp. 39–60. Greenwood/Quorum Press, New York.

BENCH-CAPON, T., P. J. SOPER, & F. COENEN 1991. "Animation of Advice leaflets Using Hypertext and Knowledge Based system techniques", *in Proceedings of the Forth International Conference on Law and Computing,* Roterdam.

BING, J. 1987. "Designing Text Retrieval Systems For Coceptual Searching", *in Proceedings of the First International Conference on Artificial Intelligence and Law,* pp. 43–51.

BING, J. 1988. "The Text Retrival System as a Conversation Factor". *in* Watter, C., (ed.), *Computing Power and Legal Language,* chapter 11, pp. 119–134. Green/Quorum Press.

BLAIR, D. C. & M. E. MARON 1985. "An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System", *Communications of the ACM*, 28 (3): 289–299.

BOWEN, KENNETH A. & ROBERT A. KOWALSKI 1982. "Amalgmating Language and Meta Language in Logic Programming". *in* Clark, K. & Tarnlund, S.-A., (eds.), *Logic Programming*, pp. 153–172. Academic Press.

BRIGGS, J. H., C. TOMPSETT, & N. OATES 1993. "Using Rules to Guide Learning Through Hypertext", *Computer Education*, 20 (1): 105–110.

BRUFFAERTS, A. & E. HENIN 1988. "Proof Trees for Negation as Failure: Yet another Prolog Meta-Interpreter", *in* Kowalski, R. A. & Bowen, K. A., (eds.), *Proceedings of the 5th International Logic Programming Conference and Symposium*.

BRUZA, PD. & T. P. VAN DER WEIDE 1992. "Stratify Hypermedia Structure for Information Disclosure", *The Computer Journal*, 35 (3): 208–220.

BRUZA, P. D. 1993. *Stratified information disclosure: A synthesis between hypermedia and information retrieval (Ph.D. dissertation)*. Thesis Publishers, Amsterdam.

BUCKLEY, R. A. 1981. *The Law of Nuisance*. Butterworths, London.

BUNDY, A. 1987. "How to Improve the Reiability of Expert Systems". *in* Moralce, D. S., (ed.), *Research and Development in Expert Systems*, IV, pp. 3–17. Cambridge University Press.

BUSH, V. 1945. "As We May Think", *Atlantic Monthly*, 176: 101–108.

CASSON, A. & D. STONE 1992. "An Expertext System for Building Standards", *in Proceedings of Workshop on Computers and Building Standards*, Montreal.

CONKLIN, J. 1987. "Hypertext:An Introduction and Survey", *Computer*, pp. 17–41.

CROFT, W. B. 1993. "Knowledge-Based and Statistical Approaches to Text Retrieval", *IEEE EXPERT*, 8 (2): 8–12.

CROFT, W. B. & R. H. THOMPSON 1987. "I^3R: A new Approach to the Design of Document Retrieval Systems", *Journal of the Amarican Society for Information Science*, 38 (6): 389–404.

DAM, ANDRIES VAN 1988. "Hypertext'87 keynote address", *Communications of the ACM*, pp. 887–895.

DANIELS, J. J. & E. L. RISSLAND 1995. "A Case-Based Approach to Intelligent Information Retrieval", *in Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 238–245, Seattle, WA.

DAVIS, H. C., W. HALL, I. HEATH, G. HILL, & R. WILKINS 1992. "MICROCOSM: An Open Hypermedia Environment for Information Management", Technical Report CSTR 92-15, Department of Electronics and Computer Science, University of Southampton, UK.

DAYAL, S., M. HARMER, P. JOHNSON, & D. MEAD 1993. "Beyound Knowledge Representation: Commercial Use for Legal Knowledge Bases", *in Conference Proceedings of the 4th International Conference on Artificial Intelligence and Law*, pp. 167–174.

DIAPER, D. & R. RADA 1991. "Expertext: Hyperizing Expert Systems and Expertizing Hypertex". *in* Brown, H., (ed.), *Hypermedia/hypertext and object-oriented databases*, Applied information technology; 8, chapter 8, pp. 125–163. Chapman & Hall, London.

DICK, J. P. 1987. "Conceptual Retrieval and Case Law", *in Proceedings The First International Conference on Artificial Intelligence and Law*, pp. 106–115.

ELHANI, O. 1992. "Intelligent Course Architecture", *in Conference Proceedings of EXPERSYS-92*, pp. 21–27.

FALOUTSOS, C., R. LEE, C. PLAISANT, & B. SHNEIDERMAN 1989. "Incorporating String Search in a Hypertext System: User Interface and Signature File Designe Issues", Technical Report CS-TR-2266, Department of Computer Science, University of Maryland.

FALOUTSOS, C. & D. W. OARD 1995. "A Survey of Information Retrieval and Filtering Methods", Technical Report CS-TR-3514, Department of Computer Science, University of Maryland.

FAUTH, J. 1995. "Poles in Your Face: The Promises and Pitfalls of Hyperfiction", *Mississippi Review*, 4 (1).

FISCHER, G., R. MCCALL, & A. MORCH 1989. "JANUS: Integrated Hypertext with a knowledge -based Design Environment", *in Proceedings of the Hypertext'89*, pp. 105–117.

FOUNTAIN, A. M., W. HALL, I. HEATH, & H. C. DAVIS 1990. "MICROCOSM: An open model for Hypermedia with Dynamic Linking", *in Hypertext: Concepts, Systems and Application (Proceedings of ECHT' 90)*. Cambridge University Press.

FRAKES, W. B. & R. BAEZA-YATES 1992. *Information Retrieval - Data Structures & Algorithms*. Prentice Hall.

FROST, R. A. 1987. *Introduction to knowledge base systems*. Collins Professional and Technical, London.

GARG, P. 1988. "Abstraction mechanisms in hypertext", *Communications of the ACM*, 31 (7): 863–870.

GELBART, D. & J. C. SMITH 1991. "Beyond Boolean Search: FLEXICON, A Legal Text-Based Intelligent System", *in Conference Proceedings of the Third International Conference on Artificial Intelligence and Law*, pp. 225–234.

GELBART, DAPHNE & J. C. SMITH 1993. "FLEXICON: An Evaluation of the Statistical Ranking Model Adopted to Intelligent Legal Text Management", *in Conference Proceedings of the 4th International Conference on Artificial Intelligence and Law*, pp. 142–151.

GENTNER, D. 1983. "Structure Mapping: A Theoretical Framework for Anology", *Cognitive Science*, 7: 155–170.

GENTNER, D. & R. LANDERS 1989. "The Mechanisms of Anological Learning". *in* Vonsniadou, S. & Ortony, A., (eds.), *Similarity and Analogical Reasoning*, pp. 199–241. Cambridge University Press.

GLOOR, P. A. 1991. "CYBERMAP: Yet Another Way of Navigating in Hyperspace", *in Hypertext'91 Proceedings*, pp. 107–119.

GREENLEAF, G., A. MOWBRAY, & A. TYREE 1991. "The DataLex Legal Workstation - intergating tools for lawyers", *in Proceedings of The Third International Conference on Artificia Intelligence and Law*, pp. 215–224, St. Catherine's College, Oxford, England. Society for Computer and Law, ACM.

HAFNER, C. 1987. "Conceptual Organization of Case Law Knowledge Bases", *in Proceedings of the First International Conference on Artificial Intelligence and Law*, p. 35, Boston.

HALASZ, F. G. 1988. "Reflection on NoteCards: Seven Issuses for the Next Genration of Hypermedia Systems", *Communication of the ACM*, 31 (7): 836–852.

HALASZ, F. G. 1991. "Seven Issues: Revisited", Hypertext'91: Closing Keynote Address.

HALL, WENDY 1994. "Ending the Tyranny of the Button", *IEEE Multimedia*, Spring: 60–68.

HAMFELT, A. & J. BARKLUND 1990. "An Intelligent Interface to Legal Data Bases Combining Logic Programming and Hypertext", *in Proceedings of International Conference on Database and Expert Systems Applications*, pp. 56–61, Vienna, Austria.

HAMMOND, P. 1983. "Representation of DHSS Regulations as a Logic Programme", *in Proceedings of the 3rd BCS Expert System Conferenc*, Cambridge. British Computer Society.

HAYES-ROTH, F. & N. JACOBSTEIN 1994. "The State of Knowledge-Based Systems", *Communications of the ACM*, 37 (3): 27–39.

HEATH, I. 1992. *An Open Model For Hypermedia: Abstract Links From Documents*. PhD thesis, Department of Electronics and Computer Science, University of Southampton, Southampton.

HEURING, R. F. V. & R. A. BUCKLEY 1987. *Salmond and Heuring on the Law of Torts*. Sweet and Maxwell, London.

HOGGER, C. J. 1981. "Derivation of Logic Programs", *Communications of ACM*, 28 (2): 372–379.

HOLYOAK, K. J. & P. THAGARD 1989. "Anological Mapping by Constraint Satisfaction", *Cognitive Science*, 13: 295–335.

HUTCHINGS, G. A., L. A. CARR, & W. HALL 1991. "StackMaker: An Environment for Creating Hypermedia", Technical Report CSTR 91-11, Department of Electronics and Computer Science, University of Southampton.

JACKSON, P. 1990. *Introduction to Expert Systems*. Addison-Wesley, 2nd edition.

JONASSEN, D. H. 1989. *Hypertext/hypermedia*. Educational Technology Publications.

JONES, K. S. 1991. "The Role of Artificial Intelligence in Information Retrieval", *J. of the Am. Soc. for Information Science*, 42 (8): 558–565.

KIBBY, M. R. & J. T. MAYES 1989. "Towards Intelligent Hypertext". *in* McAleese, R., (ed.), *Hypertext: Theory into Practice*, chapter 12, pp. 164–172. Intellect, Oxford.

KING, A. & P. SOPER 1994. "Depth-K Sharing and Freeness", *in Proceedings of ICLP'94*, pp. 553–568.

KONSTANTINOU, V., J. SYKES, & G. N. YANNOPOULOS 1993. "Can Legal Knowledge Be Derived From Legal Text?", *in Conference Proceedings of the 4th International Conference on Artificial Intelligence and Law*, pp. 218–227.

KOWALSKI, A. 1991. "Case-Based Reasoning and The Deep Structure Approach to Knowledge Representation", *in Conference Proceedings of the 3rd International Conference on Artificial Intelligence and Law*, pp. 21–30.

KOWALSKI, R. & M. SERGOT 1990. "The Use of Logical Models in Legal Problem Solving", *Ratio Juris*, 3 (2): 201–218.

KOWALSKI, R. A. 1979. *Logic for Problem Solving*. Elsevier Publishing, North-Holland.

LINZER, P. 1988. "Precise Meaning and Open Texture in Legal Writing and Reading". *in* Walter, C., (ed.), *Computing Power and Legal Language*, chapter 2, pp. 5–12. Greenwood/Quorum Press.

LITTLEFORD, A. 1991. "Artificial Intelligence and Hypermedia". *in* Berk, E. & Devline, J., (eds.), *Hypertext/Hypermedia Handbook*, chapter 22, pp. 357–378. McGraw Hill.

LLOYD, J. W. 1987. *Foundations of logic programming*. Springer, Berlin, 2nd extended ed edition.

MAHAPATRA, R. K. & J. F. COURTNEY 1992. "Research Issues in Hypertext & Hypermedia for Business Applications", *DATABASE*, Fall: 10–18.

MARCHIONINI, G. & B. SHNEIDERMAN 1988. "Finding Facts vs. Browsing Knowledge in Hypertext Systems", *Computer*, pp. 70–80.

HOME OFFICE 1991a. "Fees Leaflet March 1991", Issued by Home Office, Nationality Division (UK).

HOME OFFICE 1991b. "Guide B: Registration as a British citizen", Issued by Home Office, Nationality Division (UK).

HOME OFFICE 1991c. "Leaflet BN 12:Information about registration as British citizen...", Issued by Home Office, Nationality Division (UK).

MCCARTY, T. L. 1977. "Reflections on TAXMAN: An experiment in artificial intelligence and leagl reasoning", *Harvard Law Review*, 90: 837–893.

MCCARTY, T. L. 1980. "The TAXMAN project:toward a cognitive theory of legal argument". *in* Niblett, B., (ed.), *Computer Science and Law*, pp. 23–43. Cambridge University Press.

MCKNIGHT, C., A. DILLON, & J. RICHARDSON 1991. *Hypertext in Contex*. The Cambridge series on electronic publishing. Cambridge University Press.

MINCH, R. P. 1989. "Application and Research Areas for Hypertext in Decision Support Systems", *Journal of Management Information Systems*, 6 (3): 119–138.

MULDER, R. V. DE & C. J. M. COMBRINK-KUITERS 1996. "Is a computer capable of interpreting case law", *Journal of Information, Law and Technology*, (1).

MULDER, R. V. DE, M. J. VAN DEN HOVEN, & C. WILDEMAST 1993. "The concept of concept in 'conceptual legal information retrieval", *Law Technology Journal*, 3 (3).

NELSON, T. H. 1967. "Getting it out of your system". *in* Schecter, G., (ed.), *Information Retrieval: A Critical Review*. Thompson Book, Washington, DC.

NIELSEN, J. 1990a. *Hypertext and hypermedia.* Academic Press.

NIELSEN, J. 1990b. "Navigation through Hypertext", *Communications of the ACM*, 33 (3): 297–310.

NIELSEN, J. & U. LYNGAK 1989. "Two Field Studies of Hypermedia Usability", *in Proceedings of the Hypertext II Conference*, York, UK.

NITTA, K., S. WONG, & Y. OHTAKE 1993. "A computational Model for Trial Reasoning", *in Proceedings of the 4th International conference on Artificial Intelligence and Law*, pp. 20–29.

OSKAMP, A. & P. H. VAN DEN BERG 1990. "Legal Expert Systems and Legal Text Retrieval Systems: How about Integration". *in* Kasperson, H. W. K. & Oskamp, A., (eds.), *Amongst friends in computers and law : a collection of essays in remembrance of Guy Vandenberghe*, Computer/law series ; v.8, pp. 269–278. Kluwer.

PAQUIN, L., F. BLANCHARD, & C. THOMASSET 1991. "Loge-expert: From a Legal Expert System to an Information System for Non-Lawyers", *in Conference Proceedings of the 3th International Conference on Artificial Intelligence and Law*, pp. 254–259.

PASHA, M. A. & P. SOPER 1995. "Conceptualising Hypertext Links Using Expert System Techniques", *in Conference Proceedings of the 7th International Conference on Artificial Intelligence and Expert Systems Applications (EXPERSYS-95).*

PETZOLD, C. 1992. *Programming Windows 3.1.* Redmond, Wash, Microsoft.

POLLIT, S. 1987. "CanSearch: An Expert-Systems Approach to Information Retrieval", *Information Processing and Management*, 23 (2): 119–138.

POULIN, D., P. BRATLEY, J. FREMONT, & E. MAKAAY 1993. "Legal Interpretation in Expert Systems", *in Conference Proceedings of the 4th Artificial Intelligence and Law*, pp. 90–99.

RADA, R. 1995. "Expertmedia and Health Care Applications". *in* Rada, R. & Tochtermann, K., (eds.), *Expertmedia - Expert Systems and Hypermedia*, chapter 4, pp. 93–106. World Scientific, London.

RADA, R. & J. BARLOW 1989. "Expertext: Expert System and Hypertext", *in Procedding of the EXSYS-89, IITT International*, pp. 21–26.

ROBINSON, J. A. 1992. "Logic and Logic Programming", *Communications of the ACM*, 35 (3): 41–65.

ROSE, D. E. & R. K. BELEW 1989. "Legal Information Retrieval: A Hybrid approach", *in Conference Proceedings of the 2nd International Conference on Artificial Inteligence and Law*, pp. 138–146.

SAARENPAA, A. 1991. "Computers and Legal Life: The Use of Computers in Legal Life and Their Role in Legal Thinking". *in* Blume, P., (ed.), *Nordic Studies in Information Technology and Law*, pp. 45–71. Kluwer Law and Taxation Publishers, Deventer, The Netherland.

SALTON, G. & M. J. MCGILL 1983. *Introduction to Modern Information Retrieval.* McGraw-Hill, New york.

SANDERS, K. E. 1991. "Representing and reasoning about open-textured predicates", *in Proceedings of the third International Conference of Artificial Intelligence and Law*, pp. 137–144.

SCHILD, U. J. 1989. *Open-Textured Law, Expert Systems And Logic Programming*. PhD thesis, Imperial College of Science and Technology, University of London, UK.

SCHLOBHM, D. & T. L. MCCARTY 1989. "ESP II: Estate planning with prototypes.", *in Proceedings of the Second International Conference on Artificial Intelligence and Law*, pp. 1–10.

SCHLUMBERGER, P. C. 1989. "SHADOW: Fusing Hypertext with AI", *IEEE Expert*, Winter: 65–78.

SCHWABE, D., B. FEIJO, & W. G. KRAUSE 1990. "Intelligent hypertext for Normative Knowledge in Engineering", *in* Rizk, A., Streitz, N., & Andre, J., (eds.), *Hypertext : concepts, systems and applications : proceedings of the first European Conference on Hypertext*, pp. 123–136. Cambridge University Press.

SERGOT, M. 1982. "A Query-The-User Facility For Logic Programing", Research Report DOC 82/18, Imperial College of Science and Technology, University of London.

SERGOT, M. 1988. "A Brief Introduction to Logic Programming and Its Application in Law". *in* Walter, C., (ed.), *Computing Power and Legal Language*, chapter 5, pp. 25–38. Green Wood/Quorum Press, New York.

SERGOT, M. 1991. "The Representation of Law in Computer Programs". *in* Bench-Capon, T. J. M., (ed.), *Knowledge-Based Systems and Legal Applications*, pp. 3–67. Acdemic Press.

SERGOT, M., F. SADRI, R. A. KOWALSKI, F. KRWACZEK, P. HAMMOND, & H. T. CORY 1986. "The British Nationality Act as a Logic Program", *Communications of the ACM*, 29 (5): 370–386.

SERGOT, M. J. & Y. A. COSMADOPOULOS 1990. "The logic programming system Skilaki: design and implementation", Technical report, Department of Computing, Imperial College of Science and Technology, University of London, UK.

SKALAK, D. B. & E. L. RISSLAND 1991. "Argument Moves in a Rule-Guided Domain", *in Proceedings of The Third International Conference on Artificial Intelligence and Law*, pp. 1–11.

SOPER, P. & T. BENCH-CAPON 1994. "Coupling Hypertext and Knwledge Based Systems: Two Application in the Legal Domain", *Artificial Intelligence and Law*, 2: 293–313.

SOPER, P., W. HALL, M. A. PASHA, & I. HEATH 1993. "Knowledge Based Hypermedia", *in Proceedings of the Workshop on Logic Programming Support Environments*, University of Edinburgh, UK.

SOPER, P. & M. A. PASHA 1994a. "Knowledge Based Hypermedia for Information Management", *in Proceedings of the 14th annual conference of the British Computer Society Specialist Group on Expert Systems(ES-94*, Cambridge, UK.

SOPER, P. J. & M. A. PASHA 1994b. "KBH: A framework for managing legal information", *in Proceedings of the Third Conference on Information Technology and its Applications*, pp. 150–158, Leicester, UK.

SOPER, P. J. & M. A. PASHA 1994c. "Logic Programming Techniques for Handling Navigational Problems of Hypermedia Systems", *in Proceedings of the 10th Logic Programming Workshop*, University of Zurich, Switzerland.

STEEL, B. D 1993. *LPA 386-Prolog for Windows: Programming Guide*. Logic Programming Association Ltd, London, UK.

STERLING, L. & A. LAKHOTIA 1988. "Composing Prolog Meta-Interpreters", *in Proceedings of the 5th International Conference of Logic Programming*, pp. 386–403.

SUSSKIND, R. E. 1987a. *Expert System in Law: A Jurisprudential Inquiry*. Clarendon, Oxford.

SUSSKIND, R. E. 1987b. *Expert Systems in Law*. Oxford University Press, Oxford.

TANNENBAUM, A. S. 1992. *Moderen Operating Systems*. Prentice-Hall.

TIMPKA, T., H. MARMOLIN, & N. HALLBERG 1995. "Expert media in Ambulatory Clinics". *in* Rada, R. & Tochtermann, K., (eds.), *Expertmedia - Expert Systems and Hypermedia*, chapter 5, pp. 107–136. World Scientific, London.

TOCHTERMANN, K. & V. ZINK 1995. "Artificial Intelligence and Hypermedia". *in* Rada, R. & Tochtermann, K., (eds.), *Expertmedia - Expert Systems and Hypermedia*, chapter 3, pp. 61–89. World Scientific, London.

TOMEK, I. & H. MURER 1992. "Helping The User to Selecet a Link", *Hypermedia*, 4 (2): 111–123.

V.D.L. GARDNER, A. 1987. *An Artificial Intelligence Approach to Legal Reasoning*. MIT Press, Cambridge, Massachusetts.

V.OSSOS, G., J. ZELEZNIKOW, T. DILLON, & V. V.OSSOS 1991. "An example of Integrating Legal Case Based Reasoning with Object-Oriented Rule-Based System: IKBALS II", *in Conference Proceedings of the 3th International Conference on Artificial Intelligence and Law*, pp. 31–41.

WILDEMAST, C. A. M. & R. V. DE MULDER 1992. "Some Designe Considerations for a conceptual Legal Information Retrieval System". *in* Grutters, C. A. F. M., van den Herik, J. A. P. J. B. H. J., Schmidt, & de Vey Mestdagh, C. N. J., (eds.), *Legal Knowledge Based Systems: Information Technology and Law*. JURIX'92, Koninklijke Vermande, Lelystad, NL.

WOLSTENHOLME, D. E. 1989. "Amalgamating Regulation and Case-based Advice System Through Suggested Answers", *in Proceedings of the 2nd International Conference on AI and Law*.

WOLSTENHOLME, D. E. 1991. *External Data in Logic-Based Avice Systems*. PhD thesis, Imperial College of Science, Technology and Meicine, University of London.

YANG, S., D. ROBERTSON, & J. LEE 1993. "KICS: A Knowledge-Intensive Case-Based Reasoning System for Statutory Building Regulation and Case Histories", *in Proceedings of the 4th International conference on Artificial Intelligence and Law*, pp. 254–263.

YODER, E. & T. C. WETTACH 1989. "Using Hypertext in a Law Firm", *in Conference Proceedings of the Hypertext'89*.

ZHUOXUN, L., H. DAVIS, & W. HALL 1992. "Hypermedia Links and Information Retrieval", *in Presented at British Computer Society 14th Information Retrieval Colloqium*, Lancaster University.