Fall 1-31-2001

A more efficient document retrieval method for TEXPROS

Yin Dong
New Jersey Institute of Technology

ABSTRACT

A MORE EFFICIENT DOCUMENT RETRIEVAL METHOD FOR TEXPROS

by
Yin Dong

Document processing is a critical element of office automation. Through document classification, extraction and filing, documents are automatically placed into a knowledge base according to certain rules. Document retrieval is a process to get a document back according to a user's requirements and to show the results to the user. Hence, a good user-interface and an efficient retrieval algorithm become core parts of document retrieval.

Unlike previous browsers that have been proposed for this purpose, this dissertation develops a new browser that has a user interface with more tools, and one that has a more efficient retrieval algorithm that can deal with a wide variety of retrieval situations.

In this dissertation, from the view of an interface, the new browser provides more functions such as "zoom in" and "zoom out", (i.e. automatic scaling of the portion of a graph that is of interest to a user), and help. These functions give users an easier way to view a large graph in one window and provide users with help during the retrieval process.

The new browser also provides an algorithm that makes retrieval more efficient by using a reusable base. The Reusable Base is used to hold information that is most related to the user previous desires and the information stored in the Reusable Base is more easily used to form the OP-Net than that in the System Catalog. Hence, it eliminates

the need to go to the System Catalog to find the results. This speeds up the retrieval significantly – at least two times faster than without the Reusable Base.

Further, the new browser provides information about the folder organization and the document type hierarchy that is in addition to the OP-Net. If users know the type of documents they want, or which folder they are interested in, they can go to the particular document type or the particular folder directly.

A MORE EFFICIENT DOCUMENT RETRIEVAL METHOD FOR TEXPROS

by
Yin Dong

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer and Information Science

Department of Computer and Information Science

January 2001

APPROVAL PAGE

A MORE EFFICIENT DOCUMENT RETRIEVAL METHOD FOR TEXPROS

Yin Dong

Aug 12, 2000

Dr. Gary Thomas, Dissertation Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

Aug 12, 2000

Dr. Peter A. Ng, Dissertation Advisor — Date
Professor of Computer Science, University of Nebraska at Omaha

Aug. 10, 2000

Dr. Daochuan Hung, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

Aug 12, 2000

Dr. Ronald Curtis, Committee Member — Date
Assistant Professor of Computer Science, William Paterson University of New Jersey

Aug 12, 2000

Dr. Ajaz Rana, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Yin Dong

Degree: Doctor of Philosophy in Computer and Information Science

Date: January 2001

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer and Information Science,
  New Jersey Institute of Technology, Newark, NJ, 2001

- Master of Science in Computer Science,
  Beijing University of Astronautics and Aeronautics, Beijing, P.R.China, 1993

- Bachelor of Science in Computer Science,
  Sichuan University, Chengdu, Sichuan Province, P.R.China, 1990

Major: Computer Science

Presentations and Publications:

Yin Dong, Gary Thomas, D.C. Hung, Curtis Ronald, Rana Ajaz and Peter A. Ng,
"Efficient Retrieval Method in TEXPROS",
Journal of Systems Integration, September 2000. (Submitted)

Yin Dong, Gary Thomas, D.C. Hung, Curtis Ronald, Rana Ajaz and Peter A. Ng
"Browser with an Efficient Retrieval Method by Using Reusable Base in TEXPROS",
Fourth IAPR International Workshop on Document Analysis Systems, Rio de Janeiro, Brazil, 2000. (Accepted)

To my beloved family

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Gary Thomas and Dr. Peter Ng, who not only served as my research supervisors, providing valuable and countless resources, insight, and intuition, but also constantly gave me support, encouragement, and reassurance. Special thanks are given to Dr. Daochuan Hung, Dr. Ronald Curtis and Dr. Ajaz Rana for actively participating in my committee.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

With the explosive growth of information technology, effective information representation, retrieval and management have become the central issues in information retrieval systems. Research in this area has been conducted intensively for many years. Some of the major issues are: How should information be represented and managed in the stored database? How can information be retrieved from databases completely and precisely? Is the retrieved information presented to the user in an easily understood and friendly way? In this chapter we shall give a brief discussion of each of these issues and its related work. We shall give the organization of the dissertation at the end of this chapter.

1.1 Information Representation and Management

In today's information era, large numbers of documents are produced, stored and exchanged among offices. Hence, using computers to store and retrieve documents automatically has become urgent. Information representation and management establishes the basis for a document retrieval system. They are the core parts of automatic document processing. They involve document modeling, document classification and knowledge base management. Document modeling is concerned with ways of modeling data when it is received and the representation of the structure of the information. Document classification identifies the type of a document. Generally speaking the type of a given document can be identified based on its document's layout structure, conceptual structure and content structure. Knowledge base management deals with ways of storing

the information in the knowledge base. According to Augusto Celentano in [1], knowledge of a document can be divided into three levels: 1. Static knowledge, which describes the document type, contents and logical structure; 2. Procedural knowledge, which describes the document usage within the office in terms of its relationships to other documents, with activities and their executors; 3. Domain knowledge, which is concerned with laws, habits, rules and regulations that constrain the document's meaning and usage. Different information systems deal with the document knowledge in different ways, depending on their different purposes and different areas. In general, much work has been done in the areas of document modeling. We summarize that work as follows:

- Primitive representation: This method stores information into database without extracting any other information from it. The information retrieval process will have to go through the entire database.
- Keyword-based representation: In this method, a set of keywords in a piece of information is extracted to represent it. It is used in conventional information retrieval systems. Within this keyword framework, statistical and probabilistic models, fuzzy models, and vector space models have been proposed [2][3].
- Text-grammar representation: This is an extended keyword-based representation model. It divides a chuck of information grouped by a set of keywords into several smaller blocks according to the conceptual roles of these keywords within a text. These conceptual roles, identified using text's grammar as the model, may be described informally as the particular function served by the keywords in the overall message communicated by the text [4].

- Conceptual representation: The conceptual document model concerns the semantic and logical description of documents. Each document is modeled by means of a conceptual structure, that is, a collection of properties that describes the semantic properties of the document [1].
- Object-oriented conceptual representation: The information system can be viewed as a collection of interrelated objects, and it can also be represented by a directed labeled graph [5].

Also a great deal of research work has been done in the area of knowledge base management in information systems. We summarize this work as follows:

- File system: a file system can be viewed as the traditional document processing system. In a file system, documents are organized as files in a tree structure. Documents are represented by collections of index terms that are called representatives of all of the documents in the file. This makes retrieval very simple, but inefficient. It does not support any intelligent behaviors. For example self-indexing inverted files, which are implemented by the inclusion of an internal index in each compressed, inverted list. This method can reduce both the CPU and retrieval time simultaneously [6].
- Knowledge-based information retrieval system: Employing knowledge-based techniques in an information retrieval system embodies some aspects of human knowledge and expertise to perform the tasks that were done by human experts in the past. It makes the intelligent system behaviors possible. It empowers an information system to process and manage documents in the office environment.

- Thesaurus: Thesaurus technology is widely used in the knowledge-based information retrieval system. A thesaurus consists of a group of synonyms for each word. It has been proven in practice that using a thesaurus can improve the system performance.
- Distributed knowledge-based information retrieval system: It deals with the knowledge management in a distributed environment. In such an environment, there is a need for substantial information management flexibility, partial integration/sharing and autonomy.

1.2 Information Retrieval

With the advent of the advanced database systems and networking technology, the amount of information will grow explosively. Existing database technology has difficulty handling the surge of information networking. How can data stored in a distributed database be retrieved from various sources and then be interpreted usefully with respect to various user groups? For example, an office manager would like to summarize all of the important meetings that occurred on a particular day, with meeting times and locations, participants and the nature of the meetings (i.e. subjects). The President of a university might like to gather any existing legal documents that state the obligations and responsibilities of a newly established College of Information Science and Technology, if it is owned partially by the university. It is almost impossible to extract useful information from different sources of voluminous documents or data from the stored bases. The problems are as follows: How does a user gather the relevant data from various documents and then store that information in such a way that it could provide a user all of the implicit and explicit information? How does a user retrieve effectively and efficiently relevant information from various documents? By modeling the data, we deal

with the data gathering and analysis. We need to consider the representation of the structure of the information, document classification, and document filing and storage. These affect information retrieval. However, we can classify information retrieval into two major approaches. One is a canonical approach that is based on formal querying [7,8]. This approach requires a full understanding of data representation, storage organization, the conceptual structure of the application, and query language. Generally speaking, this approach could retrieve information directly and immediately. The other approach is an application of browsing technology that deals with vague queries by taking existing logical and physical data models into account, in comparison to the first approach. In this approach information could be obtained through repeatedly requesting data and reviewing part of the information from the information base, without any knowledge of the data representation, storage organization, the conceptual structure of the application or the query language. However, it requires more time to obtain the desired information, in comparison to the first approach. In general, much work has been done in the area of information retrieval methods. Those methods could be summarized as follows:

1. Apply interactive and primitive tools in retrieval of information. They are, for example, SQLPlus of Oracle, DBACCESS of Informix, and many others that are provided by the database vendors. But users are required to know fully the SQL language and the internal structure of the database. For example, an SQL user must know the database schemes and the relationships among the schemes.

2. Extract meaningful context from information retrieval and create indices to organize information, using various intermediate data structures, such as dictionary and thesauri. These help to search for meaningful information [9-11].
3. Extend the existing Structured Query Language (SQL) into one that allows information retrieval to be conducted easily. For example Knowledge Query Language (KQL) is an extension of SQL [1].
4. Retrieve information concurrently from the networked infrastructure with the aid of agents that cooperate with each other to retrieve information automatically from several information bases [12,13].
5. Apply artificial intelligence in information retrieval [14].

1.3 Information Display Visualization

The application of a good user interface is a central and essential issue for a successful information retrieval system. It is difficult to design a user interface to present the user a good sense of information that is well-represented in existing bases. First, the design of a good user interface has many alternatives and therefore is complex [15]. For the same functionality, there are many different ways for implementing it. Secondly, the criteria for evaluating a good design for an interface is not sufficiently well understood and established [15]. Users' behaviors and interactions with the interface are very important factors to be considered during the interface design. During the process of designing a user interface for a system, various experiments are conducted using selected participants from various groups of users, who use the system at different levels. A group of users who are not familiar with the system are used to conduct intensive tests and collect

statistical results of using the system to determine how good the user interface designs are. A general procedure could be: give users some basic training, conduct experiments by asking them to use the system to search for a series of given objects, record the search time and perform statistical studies, and comparing the benefits of several test scenarios.

A user interface of an information retrieval system is the front-end component used by all users to communicate with the system. It controls the access of information resources. It allows users to operate effectively and automatically. A good user interface can make the system friendlier to users. Most information retrieval tools provide a graphical user interface (GUI) allowing users to conduct easily a series of queries of data from a given database. Users need only to enter some keywords regarding their knowledge of the data to be searched without any knowledge about the internal database infrastructure or the query languages.

A good user interface should contain the following features:

- Grouping functionality: Divides information into different groups based on pre-defined criteria. Information is grouped together around the predefined criteria. This allows easier exploration of desired information.
- Exploration functionality: Allows users to discover useful information from the generic to the specific, taking the system architecture or infrastructure into consideration, such as the cataloging defined by the system.
- Help functionality: Provides user-help tools whenever a user needs help to use the system.
- Accuracy of the query: Gives users information as accurate as possible. This could help users find the desired information quickly.

- Evaluation of user interface: Evaluates continuously the user interface provided, by collecting automatically all possible data regarding users' behaviors and their interactions with the system through the user interface.
- Ranking obtained results: Applies rules for ordering and matching the information obtained with the queries to show the results in rank order based on a pre-defined criteria. This facilitates the process of retrieving results that are more related to the query; hence the process of retrieving information is more efficient.
- Query refinement: Allows users to refine a given query based on the obtained result and newly gained knowledge from the query. The system performance could be improved greatly if the system could give useful and helpful suggestion to the end users. This is a difficult issue. There is little successful research that presents a good method to improve this procedure [16].

1.4 Related Works

Usually, the whole process of information retrieval has the following characteristics:

- All information is entered and stored into the system according to a predefined data model.
- Users' information request to the system can be interpreted as a query.
- The system executes the query and finds the useful information in a system-defined order, such as a chronological order, or a more complicated one, e.g., a weighted value.
- The information is displayed to the user in an easy way to understand.

- Sometimes, the users explore the information. Sometimes, they reformulate their requests to get more information from the system.

Much work has been conducted in information retrieval (IR) systems. Usually in IR, the most popular approaches are based on statistics [9], semantics [10] [17], formal querying [7, 8], and on keywords [11]. IR systems are being enhanced to combine traditional techniques with AI (artificial intelligence) technique [14]. These systems have their specific use in the document processing system. Many document-processing systems have been developed in the past. Basically, they fall into four categories [47].

1. The first group deals with multimedia information including text, forms, images and audio files. MULTOS [48], for example, supports a well-defined query language and many query-processing techniques. One system was developed by Dario Lucarella and Antonella Zanzi [5] to present a graph-based object model as a uniform framework for direct manipulation of multimedia information.
2. The second group deals with text-based information retrieval. A recent work is Kabiria's [1] distributed client/server architecture, which supports the classification, filing, and retrieval of documents and maintenance of system knowledge.
3. The third group is concerned with the exchange of messages and their filtering. The goal of such systems is to help users filter, sort and prioritize messages that are addressed to them and also to help them find particular messages. Relevant work includes INFORMATION LENS [49] system and many others.
4. With the advent of the World Wide Web, networked information systems with advanced methods for browsing, searching and accessing document collection in repositories become one of the central issues that needed to be resolved.

Among these systems, the following two systems – MULTOS and Kabiria, are closely related to TEXPROS. Although these systems provide approaches to classifying documents based on their conceptual structures, these systems fail to provide users with ways of representing a document organization (such as, the folder organization in TEXPROS) from the user's viewpoint. By allowing a user to create his/her own document organization, TEXPROS greatly facilitates the management of the documents and expedites their search by narrowing the search space. In addition, in contrast to the consistent way used by the TEXPROS, these systems handle the meta-data and documents in different ways, which limit the user's ability to browse and maintain the meta-data knowledge.

In general, it is reasonable to assume that knowledge of the working domain is incomplete or unavailable during the system's design. Many IR systems are designed for a specific working domain. The limitation of these systems is their domain dependency. These systems are designed based on the availability of domain knowledge, during the system design. This limits the ability to port these systems from one domain to others with a minimum of difficulty. This also could limit the user's ability to extend the knowledge base to tailor the working domain. TEXPROS is an intelligent knowledge-based document processing system. It automates the procedure for processing, managing and retrieving document for the general working domain such as an office environment. It employs a dual model to capture the conceptual information from documents and manage them. The following is the comparison among Kabiria, MULTOS, and TEXPROS.

	Data Model	Domain	Open System
TEXPROS	simple & powerful	general	Yes
KABIRIA	complex	specific	No
MULTOS	complex	specific	No

In the meanwhile, there have been some browsers developed to support IR systems. SortTables [18] is an interface metaphor developed to support browsing. In SortTables, all items are presented in a table, where each row corresponds to a record and each column corresponds to the attribute of that record. SortTables are useful for database systems, but they are not good enough for the information retrieval systems. Some browsers have been developed to support program debugging [19]; others are to assist non-expert users to retrieve data from databases by navigating on its semantic model [20]. Many others are also to support users to access a statistical database through a semantic model of the domain [21], or they are recognized as a relevant support to query formulation [14, 22].

These systems have their own characteristics and are used in different areas. Some are more suitable for the database system, but not good enough for the IR systems. Some do not provide feedback. Some do not support the viewing of a whole information block. Some do not have a friendly interface for users. The most important problem, however, is that the efficiency of these system is low.

TEXPROS is an information retrieval system. It is also an automatic document filing and retrieval system. Documents are the major elements in the depository of the TEXPROS. Documents contain information pertinent to the interest of the users. This information must be stored in a structured manner in such a way that it can be accessed and retrieved,

and then reproduced in variety formats that can be presented to the users. For example, consider a TEXPROS document that represents a memorandum that sets up a meeting. TEXPROS allows a user to extract information regarding the "Subjects of the meeting", "Who will present at the meeting", "The time and location of the meeting" on a particular day, from a collection of documents of the meeting memo type. Such information should be able to be obtained if a user enters a request. For example, "TEXPROS, could you give a list of all the meetings I have to attend today? Please list the subject of each of the meetings, who will attend the various meetings, and the time and location for each of the meetings." To retrieve this information from TEXPROS, this information must be structured according to the meeting memo type. The information must be pre-extracted from each of the meeting memo type documents, and then be stored as a cluster called the frame instance for the corresponding documents. However, each frame instance can be regarded as a record based on the traditional relational database system. One of the major differences between the frame instance and a record is that each of the fields is of variable length and in a free format. Further, each of the fields could be nested to form a collect of subfields. To respond the query given above, TEXPROS must be able to identify all the instances of the meeting memo type, from which, only the instances for a particular day of the various meetings are retrieved. From the traditional relational database point of view, obviously, a meeting memo relational table with the relation name such as MeetingMemo must be predefined. One of the fields given must be Date (Month, Day, Year). To retrieve the all the records, which contain the date of today, the values of the primary key for the MeetingMemo must be given. However, the frame instances of meeting memo type do not contain any primary key at all. The search of

information from TEXPROS is much more intricate than the search from the relational database. Information contains in the TEXPROS is far richer than the information contained in the relational database because TEXPROS has a flexible structure that contains information in a free form that is extracted from the documents of the corresponding type. The subset of the frame instances of the meeting memo type could be identified if the type of document, such as MeetingMemo, is given and then date of the meeting is given. Therefore, the traditional information retrieval system, which is applicable in the relational database, can no longer be used.

A search engine is proposed, in which one of the major components is the browser. TEXPROS must browse through the entire file organization to search for all the frame instances of meeting memo type from all the frame instances of various types. The resulting instances can be narrowed into a subset of frame instances based on the given specified date. Traversing through the entire file organization is a tedious process. Large volume of retrieved information will be presented to the users, if the previous browser is used. Very often, various volumes of retrieved information are used for AND and OR operations in order to generate the intended information. The components for AND and OR operations must be generated from the original file organization. The retrieval process could be shortened tremendously, in magnitude of multiples of two, if these components could be reused when they are stored in a reusable base. In this dissertation, we proposed a reusable base to achieve efficient retrieval process and provide users with several tools that allow the users to view easily the result. This reusable base contains the components of the file organization, which is linked to a subsets of the documents; The components of the document type hierarchy, which is linked to a subsets of folders of the

file organization, and the component of folders that contains the values of the specified date. Information could be reconstructed by applying AND and OR operators to form new subsets of the file organization, the document type hierarchy, and folders. These new subsets can contain far few frame instances to be searched to respond to a user given request. Very often, users may wander around and around searching for the proper subsets that are related to the solutions of the request. The previous approach [39] was to re-compute these same components for reuse purpose, for they were discarded at the end of the intermediate steps of the browsing process. This is very much different from the information contains in the database for the purpose of reusability in the sense that they are subsets of the relational tables.

1.5 Organization of the Dissertation

The rest of the dissertation is organized as follows: In chapter 2, we introduce briefly the system TEXPROS with its existing browser subsystem. In chapter 3, we give the architecture of our proposed browser subsystem, which describes the main functions of every major part of the subsystem. In the next two chapters, we discuss the implementation of the first two parts of the browser. We discuss firstly the implementation of the front-end layer, first part of the browser. We then discuss the implementation of the service layer, the middle layer of the browser. The experimental results of using the proposal browser subsystem are given in chapter 6. Chapter 7 contains the conclusion and the discussion of future work. We insert the implementation of the storage layer, the last part of the browser, in appendix A [46]. And we attach the disk with the source code of the implementation prototype in appendix B.

CHAPTER 2

TEXPROS AND BROWSER SUBSYSTEM OVERVIEW

2.1 Architecture of TEXPROS

TEXPROS is an automatic document filing and retrieval system. The system provides functional capabilities for document classification [23-28], categorization [29-32], storage [31,33-35] and reproduction [34, 36], as well as extracting [23,25], browsing [37-39], retrieving [40,37,33,41], and synthesizing [40,34,36] information from a variety of documents. Figure 2.1 depicts the overall architecture of TEXPROS [33].

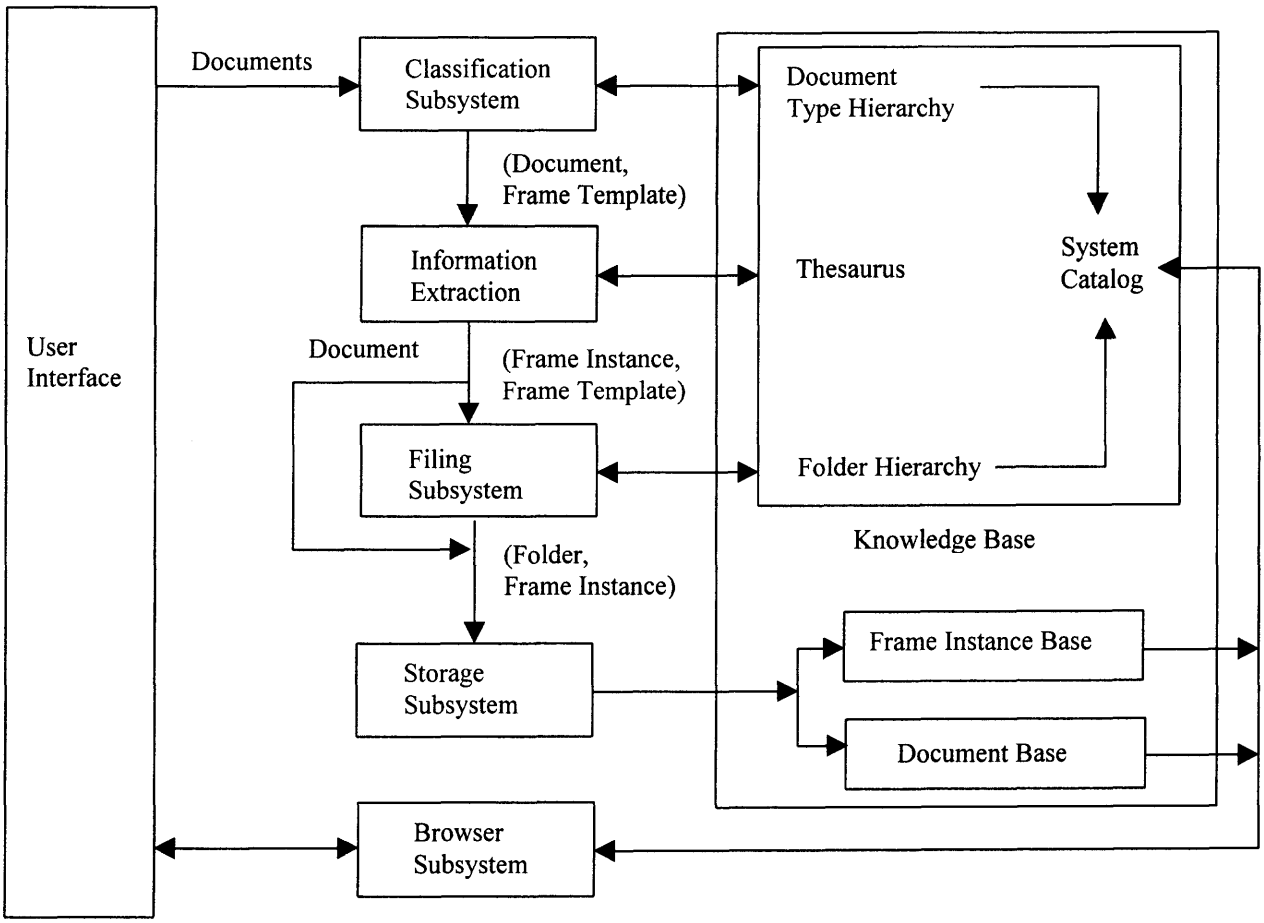


Figure 2.1 Overall Architecture of TEXPROS

TEXPROS adopts a dual model to describe, extract, file, and browse documents. This model consists of two hierarchies: a document type hierarchy and a folder organization. The frame templates are organized as the document type hierarchy (DTH). Documents are deposited into folders, and folders are organized as the folder organization that represents the user's real-world document filing system.

As shown in Figure 2.1, we see that TEXPROS has several components including classification, extraction, filing, browsing and storage. Every incoming hard copy document is entered into the SCAN/OCR that drives the scanner to scan it and save it as an image file. The classification subsystem is used to classify documents into different types based on their layout and conceptual structures. It creates a block graph for each document and classifies a document by matching its block structure graph against the existing graphs of various document types. Each document type is characterized by a frame template. After the classification subsystem, the document's type is found and the corresponding frame template is identified. The extraction subsystem is used to extract key information from the document. It extracts information based on its corresponding frame template. The extraction subsystem yields the frame instance of the document. The filing subsystem files the document into the corresponding folders. Folder organization is created by the user to represent their view of organizing their documents. A content of a folder is defined by a predicate. From implementation viewpoint, it must satisfy the predicate of the folder. To deposit the frame instance of a document into a folder, the folder only contains pointers to various frame instances, which associate with their corresponding document.

For the storage subsystem, TEXPROS employs a three-level architecture: original document base, frame instance base and folder organization. The original document base is at the bottom level, the frame instance base (repository of all frame instances) is in the middle level and the folder organization (logical representation of the storage) is at the top. Each frame instance has a pointer pointing to the corresponding original document and each folder has pointers pointing to various frame instances. Through classification, information extraction, filing and storage subsystems, the TEXPROS system gets the frame templates, frame instances and folders.

The browser subsystem operates on the frame template, frame instances and folders in the system. For example, to search a document, the browser goes through the frame templates to identify the types of documents to be searched. At least, the types of documents provide a narrow search space from which all possible document candidates for the targeted document can be found. Then the browser uses information contained in frame instances of a given document type (i.e., a frame template) to search for the targeted document. A browser subsystem is designed that is integrated with various mechanisms to provide a graphical user interface with tools and to be able to process vague queries quickly. First we use a system catalog including a thesaurus to store the knowledge about the documents. Secondly, we build a query transformation mechanism. For a given query, the query transformation mechanism searches the thesaurus for the key terms that are used by the system and constructs the normal form of the query. Thirdly, the browsing mechanism should deal with situations in which the user may not have a precise notion of what he/she is looking for. With the browser, vague queries can be

entered into the system until sufficient information is obtained to an extent that the user is able to construct a precise query for his/her request.

2.2 Dual Model of TEXPROS

TEXPROS employs a dual modeling approach to describe, extract, file, and browse documents. This model is composed of two hierarchies: a document type hierarchy that depicts the structural organization of documents, and a folder organization that represents the user's real world document filing system. An example of a document type hierarchy and an example of a folder organization are given in Figure 2.2 and Figure 2.3, respectively.

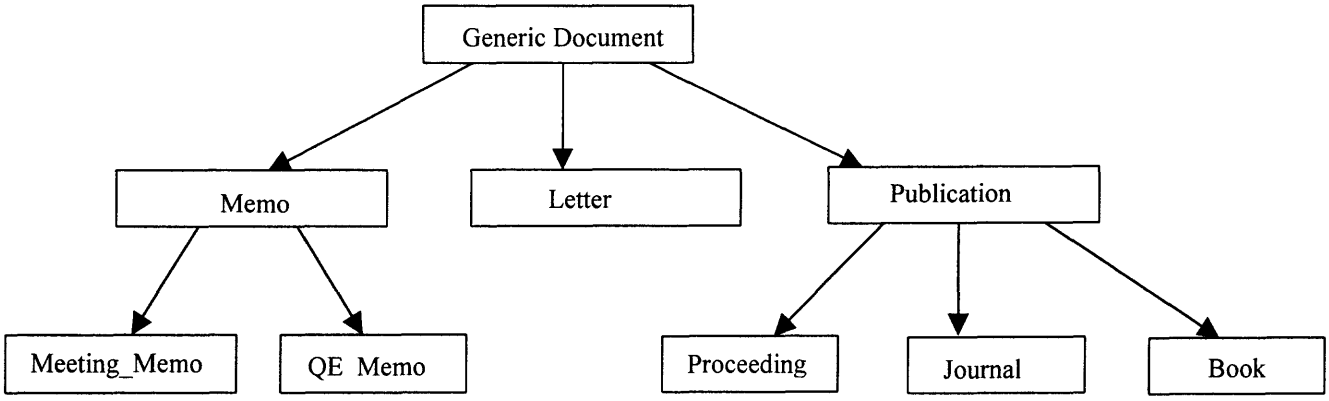


Figure 2.2 An Example of Document Type Hierarchy

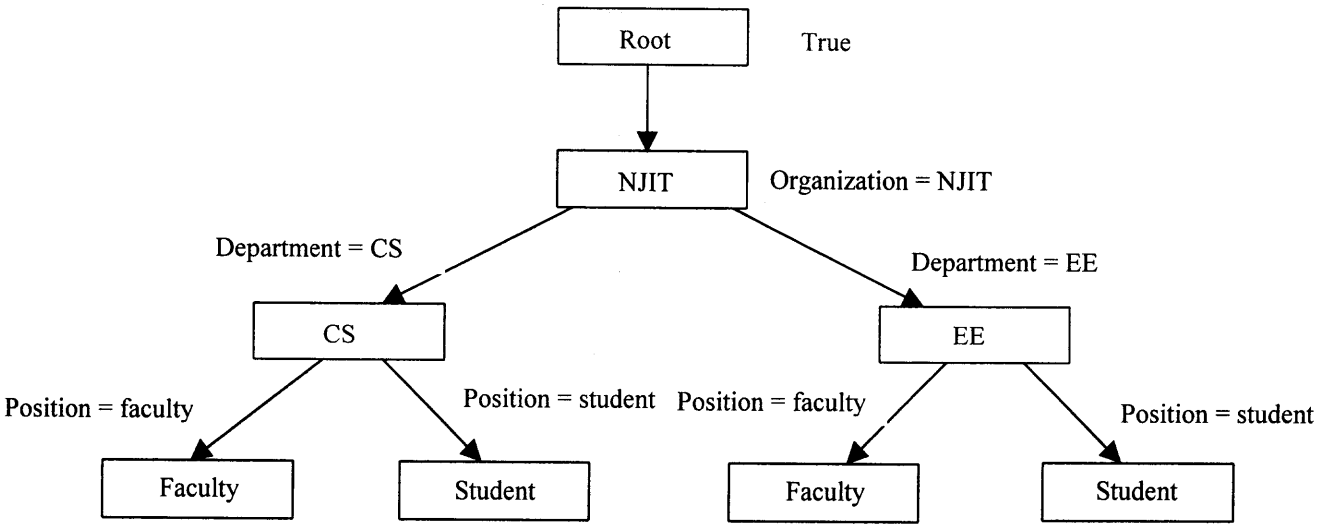


Figure 2.3 An Example of Folder Organization

The frame templates are organized as the document type hierarchy (DTH). And documents (or their frame instances) are deposited into folders and folders are organized as the folder organization that represents the user's real-world document filing system.

In TEXPROS, documents of sharing pre-defined properties are grouped together to form different classes. Each class is characterized by a frame template, which describes the common properties in terms of attributes of documents of the class. The frame templates form a document type hierarchy, and the relationship between members in the document type hierarchy is an is-a relationship. After it is processed in classification subsystem, a particular document yields a frame instance by instantiating its frame template. The frame instance represents a synopsis of the document. Different frame instances are grouped into a folder based on the user's view. A folder organization is defined according to the user's view of the document filing organization. Document filing organization is obtained by repeatedly dividing documents into groups until well-defined

groups are reached. Information about the dual models – document type hierarchy and folder organization, is stored in the system catalog. In Figure 2.4, given a document of a meeting memo (a), its frame template (b) and its corresponding frame instance (c) are obtained.

New Jersey Institute of Technology

Department of Computer &Information Science
Ext. 1550

MEMORANDUM

TO: Faculty, Special Lecturers, TA's
FROM: CIS Department
DATA: April 26, 1999
SUBJ: Student Evaluation

* * * * * * * * *

You must come to the CIS office and pick up your student evaluation forms from one of our staff as soon as possible.

(a)

Sender		
Receiver		
Subject		
MemoDate		
MtgDescription	MtgDay	
	MtgPlace	

(b)

Sender	CIS Department	
Receiver	Faculty, Special Lecturers, TA's	
Subject	Student Evaluation	
MemoDate	04/26/99	
MtgDescription	MtgDay	as soon as possible
	MtgPlace	CIS Department

(c)

Figure 2.4 (a) A Meeting Memo (b) The Frame Template of Meeting Memo (c) The Frame Instance of the Memo in (a)

2.3 Previous Work

TEXPROS is an intelligent document processing system. An *object network*, which is a semantic network, was first proposed [33], to represent the meta-data and information about the database in the browser subsystem. The *object network* has the following characteristics:

- The schema elements are captured in the *object network*. The schema elements are represented as four vertical levels: the documents, the folders, the frame templates and the attributes.
- The dual model is also captured in the *object network*. Both the descriptions of the document type hierarchy with their contents and the folder organization is unified as a single description. A horizontal level is added to represent the relationship among folders and the relationship among frame templates.

- Likewise, data elements are captured in the *object network*. The concept of access by a specific value allows users to retrieve all the occurrences of an attribute value from the database. The occurrences of an attribute value are in terms of attributes under which the given value appears [33].
- Finally, a snapshot of the system catalog is provided at any time by the *object network* and is always consistent with the System Catalog.

But using an *object network* as the representation of information has some disadvantages. Firstly, it is not easy to identify the related information from an object in the realm of *an object network*. Secondly, the *object network* cannot fully support the associations between the frame instances and the objects.

Hence, the model – *operation network* (OP-Net) was proposed [39]. OP-Net keeps all the *object network* functions and captures the associations between frame instances and the objects. A graphical interface was built to assist a user to browse through information repository during the retrieval process. Upon receiving the user's request, the system displays an OP-Net. However, even with these improvements, the previous browser still has some disadvantages:

- It does not reuse previous query results even if the next one is strongly related to it. For example, assume that the first query is a ^ b ^ c and that the next query is a ^ b ^ c ^ d or a ^ b ^ c + d. The previous browser can not use the result of a ^ b ^ c to get the result of a ^ b ^ c ^ d or a ^ b ^ c + d.
- The algorithm for retrieval is time-consuming.

- The resultant set is often too large. The algorithm finds all related information even if most of the resultant set is less useful. Even if any information of interest to the user are in the resultant set, it will still be difficult to find what the user really wants because the resultant set is too large.
- The tools provided in the graphical interface do not give a user an easy way to view the result. Because the resultant set is often so large, it takes too long for the user to find the information he/she wants.
- There is no ranking mechanism to show the most relevant information first.

In this dissertation, we begin with redefining the OP-Net. We add the "AND" and "OR" operations to the OP-Net. This makes it easier to construct OP-Net based on the previous result. We propose a reusable base to hold basic components for speeding up retrieval process.

2.4 Basic Knowledge in Browser Subsystem

In TEXPROS, there are four kinds of objects: folder, frame template, attribute and value. We give the formal definition of Object.

Definition 1. Object

An object is a two-tuple, Object = [Name, Type], where:

1. Name is a string that identifies the name of the object.
2. Type that identifies the type of the object has four values. Type belongs to {Folder, FrameTemplate, Attribute, Value}.

We use the notations [Name, Type] and Type (Name) interchangeably. From the definition of object, different types of objects can have the same name.

We define formally a frame instance repository as a place for keeping the frame instances associated with an object of a given query.

Definition 2. Frame Instance Repository (FIR)

A frame instance repository is a three-tuple FIR = [Obj, Po, FI] where:

1. Obj is an object [Name, Type].
2. Po is a predicate defined on the object.
3. FI = {fi | fi is a frame instance ID that satisfies Pop ^ Po}, where Pop is the predicate derived from a users' query.

We use FIR (Obj) to refer to the frame instance repository corresponding to an object Obj.

Now we introduce the basic structure OP-Net (operation network) in the browser. In TEXPROS, we use OP-Net to represent relationships between FIRs. It includes the relationships between frame instances and objects.

Definition 3. Operation Network (OP - Net)

An operation network (OP - Net) is a two-tuple

OP-Net = [Top, R (V, V)] where:

1. Top is a topic related to a user's query.
2. R(V, V) is a relation matrix between FIRs, where
   - V is a set of FIRs, which is related to the predicate Pop, derived from the user's query Top.
   - For every v1 and v2 in V, initialize R (v1, v2) = 0.
   - For every v1(obj1) and v2(obj2) in V,

If (((type of obj1 is a folder and obj2 is a template) or (type of obj1 is a template and obj2 is an attribute) or (type of obj1 is an attribute and obj2 is a value)) and (the intersection set of v1.FI and v2.FI is not empty))

Then R (v1, v2) = R (v2, v1) = 1.

From the definition of OP-Net, we know OP-Net is connected to a specific topic. The definition of OP-Net contains two parts. The first part is Top, which is the topic related to a user's query. Top is a Boolean expression that uses "AND" or "OR" to connect each part that is an object. Top is transformed from the users' input query, which is a Boolean expression in disjunctive normal form. The first part of the definition of OP-Net shows that an OP-Net changes according to a topic. Different topics correspond to different OP-Nets. The second part of definition of OP-Net defines the main property of the OP-Net. It tells which nodes are involved and what relationships exist among them. In this way, we obtain the frame instances that correspond to each node. We can map the OP-Net into a graph and display it to the user. The skeleton of the OP-Netis is given in Figure 2.5.

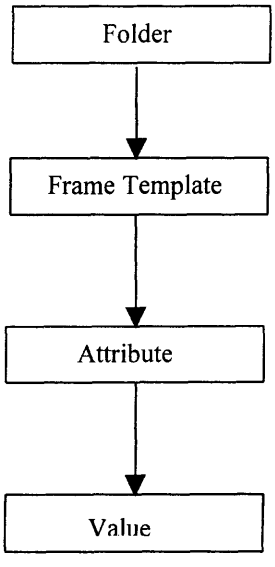


Figure 2.5 Skeleton of an OP-Net

CHAPTER 3
BROWSER SYSTEM ARCHITECTURE

In this chapter, we present the architecture of the browser subsystem and discuss the main functions of its major parts.

3.1 Architecture of Browser System

A browser is a user interface that helps users to get the information from the information storage of the system. As shown in Figure 3.1, a browser system is divided into the three components: front-end layer, service layer, and storage layer.

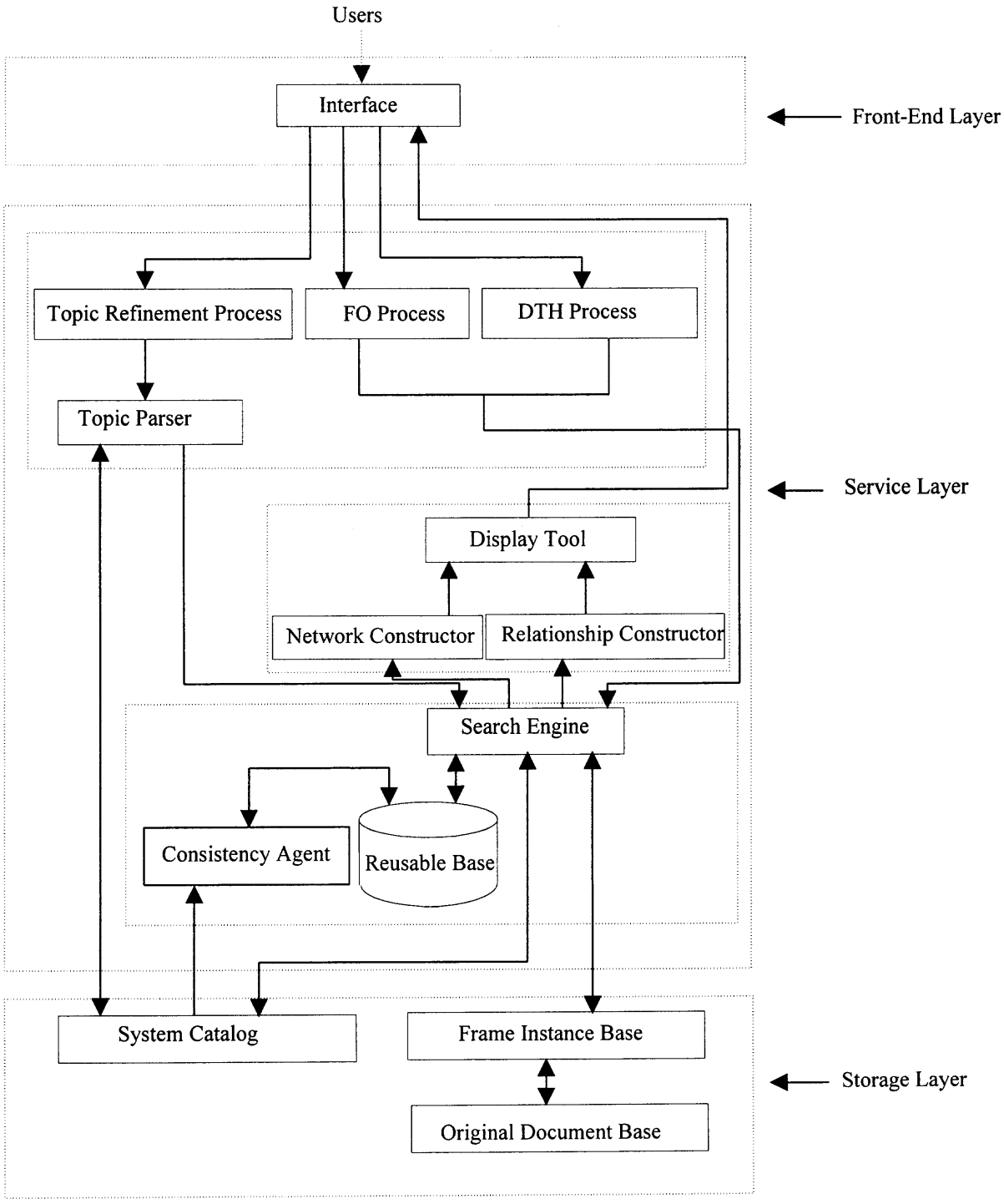


Figure 3.1 Architecture of the Browser System

3.2 Front-End Layer of Browser

The front-end layer includes a user interface. For an information system, a user interface is a front-end program that interacts with the user and controls an underlying information retrieval system accessing information resources [15]. The main goal of the user interface is to empower the user with the capability to operate effectively and automatically, without the need of any experienced human intermediary.

In our system, we provide a graphical interface for the user because it is easy to learn. Using an interface, the user follows such steps as an overview first, then zoom and filtering, finding details, and finally repeating the process [42].

3.3 Service Layer

The service layer provides services to the first layer based on the third layer. It includes the Topic Refinement, the Folder Organization, and the Document Type Hierarchy Processes, as well as the Topic Parser, Display Tool, Constructors, Search Engine, Reusable Base and Consistency Agent. These are the core parts of the browser system. From the browser perspective, the layer provides three main services:

- **Topic Refinement Process**: This process is employed when a user has only a vague idea what he/she wants, or when a user wants an overview of the relationship between some objects. Upon the receipt of a topic issued by the user, the Topic Parser translates the user's query into a normalized topic and splits the normalized topic into smaller basic components. Upon the receipt of a normalized topic from the Topic Parser, the Search Engine searches related information in the Reusable Base and the System Catalog. Once the result is returned from the Search Engine, the Network

Constructor builds a corresponding OP-Net. The Reusable Base is used to get answers quickly since it holds some frequently used basic components that can be used to construct an OP-Net directly. After the Network Constructor constructs an OP-Net, the Display Tool displays OP-Net as a graph.

- **Folder Organization Process**: This process is activated when the user tries to look at some particular folder or what he/she wants are in some particular folder. After receiving the specification from Folder Organization Process, the Search Engine searches the System Catalog to get the corresponding folder organization. The Search Engine provides the relationship between the folders and the corresponding information about every folder to the constructor. Depending on the result from the Search Engine, the Folder Organization Relationship Constructor constructs the corresponding folder organization. After the arrival of the result from the Folder Organization Relationship Constructor, the Display Tool displays the folder organization as a graph. And using the Display Tool, the user can operate on the graph and get more detailed information about every node, such as corresponding frame templates and frame instances. Each node on the graph is a folder. The user can even make a query to obtain additional information during reviewing of the graph.
- **Document Type Hierarchy Process**: This process is employed when the user tries to look at some particular document type or what he/she wants are in a particular document. After receiving the specification generated from Document Type Hierarchy Process, the Search Engine searches through the System Catalog to get the corresponding document type hierarchy. The Search Engine provides the Document Type Hierarchy Relationship Constructor with a relationship among a set of frame

templates and the corresponding information about every frame template. The Document Type Hierarchy Relationship Constructor constructs the corresponding document type hierarchy based on the result obtained from the Search Engine. After receiving the result from the Document Type Hierarchy Relationship Constructor, the Display Tool displays the document type hierarchy in a graph. And, using the Display Tool, the user can operate on the graph to get more detailed information about every node, such as the corresponding folders and frame instances. Each node in the graph is a frame template.

We further divide the service layer into three phases. The first phase is preprocessing phase, which is crucial to the efficiency of the search. The preprocessing phase contains the Topic Refinement Process, the Folder Organization Process, the Document Type Hierarchy Process and the Topic Parser. The second phase is the discovery phase that provides the algorithms to search information through the Reusable Base and the System Catalog. The discovery phase makes use of the Search Engine, the Reusable Base and the Consistency Agent. The third phase is postprocessing phase, which deals with how to organize and display the results to the user. The postprocessing phase contains the Display Tool and the Constructors.

1. Preprocessing Phase

The preprocessing phase is crucial to the efficiency of the overall process. According to the results in different domain areas and applications, preprocessing can require as much as 80 percent of the total effort [43]. Through the preprocessing phase, we can limit the size of the search space and the time requirement of the discovery phase. In the preprocessing phase, a user selects one of the processes: the Topic Refinement Process,

the Folder Organization Process, or the Document Type Hierarchy Process. Then the Topic Parser, like a converter, analyzes the user's query to obtain key terms that will be used in the following search.

- **Topic Refinement Process**

The Topic Refinement Process deals with the knowledge provided by OP-Net, which defines the relationships among folders, frame templates, attributes and values. When the user inputs the query the first time, the Topic Refinement Process is performed because the user refines the original null query. During the process of browsing, after gathering some knowledge, the user can refine the original query, which is equivalent to giving a totally new query, or modify the original query by using "AND" or "OR" with other condition.

- **Folder Organization Process**

The Folder Organization Process deals with the knowledge of the relationship among folders. In the preprocessing phase, the main goal in the Folder Organization Process is to help the user select the correct folder name.

- **Document Type Hierarchy Process**

The Document Type Hierarchy Process deals with the knowledge of the relationship among documentation types. In the preprocessing phase, the goal in the Document Type Hierarchy Process is to help the user to select the correct document type.

- **Topic Parser**

The Topic Parser is used to translate user's query into a normalized topic, and splits the normalized topic into some smaller basic components. It includes five functional units: Topic Input, Key-term Replacement, Object Identification, Topic Transformation and

Topic Splitter. It converts the user's original query into a form that could be systematically interpreted and understood by the Search Engine, and therefore performed easily and efficiently search.

2. Discovery Phase

The discovery phase's task is to find what the user is interested in. It includes the Search Engine, the Reusable Base and the Consistency Agent. The Search Engine provides algorithms to search information through the Reusable Base and the System Catalog. The Reusable Base is used to improve the efficiency of the search process. Although Consistency Agent does not contribute directly to the search, it provides the algorithm to keep the Reusable Base consistent with the System Catalog.

- **Reusable Base**

The goal of the Reusable Base is to reuse the past solution by adaptation, to evaluate the proposed solutions and finally to learn from new experience and to store the results for future use [44]. The Reusable Base is used to implement software reusability. It leads to a new view of databases where data are not only stored occasional access but are also intended to be reused. We store basic frequently used components (a simple topic that has no "AND" and "OR" between clauses and its related OP-NET) in the Reusable Base.

- **Search Engine**

The Search Engine receives input from Topic Parse, the FO process, or the DTH process. Based on the input, it provides some algorithms to search the Reusable Base and the System Catalog. It sends the results obtained from the Reusable Base and the System Catalog to the constructors.

- **Consistency Agent**

The task of Consistency Agent is to keep consistency between the Reusable Base and the System Catalog. Because the Reusable Base is dynamic, the agent continuously monitors the difference between the Reusable Base and the System Catalog.

3. Postprocessing Phase

In the postprocessing phase, the results obtained from the discovery phase are organized and displayed for the user. It includes the constructors and the Display Tool.

- **Constructor**

Based on its uses by the different processes, the Constructor is divided into three different constructors: the Network Constructor, the Folder Organization Relationship Constructor and the Document Type Hierarchy Relationship Constructor.

In the Topic Refinement Process, the Network Constructor uses related information obtained from the Reusable Base or the System Catalog to construct an OP-NET. AND and OR operators for Op-Nets are provided to combine the obtained results from several simple queries to form a final result.

In the Folder Organization Process, the Folder Organization Relationship Constructor constructs the relationships between folders using the results from the Search Engine.

In the Document Type Hierarchy Process, the Document Type Hierarchy Relationship Constructor uses the results from the Search Engine to construct the document type hierarchy.

- **Display Tool**

The system provides users with a graphical interface. A user can use "scroll up" and "scroll down" view the entire OP-Net, the folder organization, or the document type

hierarchy in one window. "Zoom in" or "zoom out" let the user view a particular part of the graph. "Help" is used to help the user overcome problems during the retrieval process.

3.4 Storage Layer of Browser

The storage layer provides services to all processes of the TEXPROS: document classification, extraction, filing, and browsing. It contains the core parts of the system, namely, the System Catalog, the Frame Instance Base, the Original Document Base and the Thesaurus.

- **System Catalog**

The System Catalog is a set of system frame templates and frame instances. The meta-data information is classified into different classes. Each class is represented by a system frame template. The frame instances are employed to store the meta-data knowledge. The browser searches through the System Catalog to get what the user needs.

- **Frame Instance Base**

The frame instances are stored in the Frame Instance Base. The advantage of the hierarchically organizational structure of Frame Instance Base is to find the needed frame instance efficiently. In the Frame Instance Base, frame instances are clustered according to their document types.

- **Original Document Base**

Every document is stored in the Original Document Base and its address of the physical storage device is recorded as the document identifier. In our TEXPROS system, we

employ a dual modeling approach for retrieval. Hence, most of the time users do not need to access an original document.

- **Thesaurus**

A thesaurus is widely used in the information retrieval area. In practices, the thesaurus has been proved to be a valid tool to support information retrieval. Thesaurus terms would be used when the actual words used by the user in a query do not match the way the concepts are expressed in the document [42]. It also has a central role in improving the usefulness of articulated knowledge in some specific working domains. It facilitates the acquisition of additional knowledge required to improve the information retrieval system.

CHAPTER 4

FRONT-END LAYER – USER INTERFACE

In this chapter, we will present the front-end part of the browser subsystem – the User Interface.

One of the significant objective of an IR system is to present its users as much relevant information as possible. Besides the retrieval mechanisms, interactive IR systems must also be concerned with the design of appropriate display mechanisms that present the retrieved information in the "best possible manner" [45]. In our system, we provide an user-friendlier interface compared to the previous browser. We say a system is user-friendly if the system uses natural language (not only keyword) or graphs in its response to a query and uses familiar icons to symbolize operations. There is a different meaning of the term "user-friendly" for novice and expert users. We treat novice and expert users differently. For example, novice users need more help to find the desired data, while experts probably have a better idea what they want and where to find it. In our system, we try to train novice users to gain experience through the retrieval process. A novice can use natural-like language to input a query and the system will show the corresponding Boolean Expression. And, the next time the novice will know more about the best way to structure a query by entering a Boolean Expression directly. Experts can override some functions if they think they do not need them or if they want to speed up the search process. In our system, we provide a graphical interface to users. Compared to a textual interface, a graphical interface is more vivid and direct. We provide more tools to the user in the user interface compared to the previous browser. This gives user a more

convenient way to view the graph. In designing the interface, we focus on the following two concerns: The User Interface should:

1. Make the elements of a graph easy to view and understand; and
2. Allow users to view the retrieving results from generic to specific.

In our system, the first concern can be accomplished through the use of "scroll up" and "scroll down" technology for allowing users to view the entire graph in one window, the use of "Zoom in" and "Zoom out" technology for allowing users to view any specific part of the entire graph, and the use of different color lines to connect different objects in the graph for users to view the graph clearly. There are many other features that could be considered in a front-end interface. For example, we could display a graph in three-dimensional way, rather than in two-dimensional way.

The second will be achieved by giving users the architecture of the graph, and then by providing users mechanisms to access any related information such as the corresponding frame instances, clear through to the corresponding original document. We will give an example in the next chapter when we discuss the Display Tool.

4.1 Display Procedure

In general, any typical interaction with an information retrieval system proceeds as follows:

- The user expresses a request as a query that is interpretable by the system;
- The system matches the query with information and retrieves the related information;
- At the first stage of display, a rough graph or an overview graph corresponding to the user's query is displayed;

- At the second stage of display, the user is able to operate on the graph displayed in the first stage and is able to obtain detailed information. In this step, our browser provides "zoom-in" and "zoom-out" functionality allowing the user to select some particular nodes of specific interest easily; and
- After the user goes through sufficient information, the query can be refined and then the second stage can be repeated.

In the front-end layer, we provide the user with three types of information based on their different requests. The first type of information is information about the OP-Net, which describes the relationship among folders, frame templates, attributes and values. The second type of information is information about the folder organization, which describes the relationship among folders. The third type of information concerns the document type hierarchy, which depicts the relationship among different types of documents. The display process for each of these three cases is similar, except for the following: For the first case, at the first stage of display, an OP-Net corresponding to the user's query is displayed. In the second case, at the first stage of display, the graph of the folder organization corresponding to user's query is displayed. And for the third case, a graph of the document type hierarchy corresponding to the user's query is displayed. For the first case, at the disposal of the user, the user is able to operate on the second stage OP-Net graph which is a refinement of the first stage graph and the user can get detailed information. For example, the user can get the related frame instances of each node in the OP-Net and can access the original documents. For the second case, the user can also operate on the second stage graph of the folder organization (which is also a refinement of the first stage graph) and the user can get more detailed information. The user can get

the related frame instances of each node in the graph, the frame templates that are involved, and therefore can access the original documents based on the frame instances and their corresponding frame templates obtained. For the third case, the user can operate continuously on the second stage graph of the document type hierarchy (which is still a refinement of the first stage of graph) and the user can get more detailed information. The user can get the related frame instances of each node in the graph, the folders that are involved, and can access the original documents.

For the overview of the OP-Net, the folder organization, or the document type hierarchy that is displayed, although it is very crowded, does make several aspects of the underlying information apparently:

- The graph of the OP-Net represents the architecture of OP-Net. Some folders are at the first level. At the second and third level, there are corresponding frame templates and corresponding attributes. And, at the last level there are corresponding values. The graph of the folder organization shows the parent – children relationship between folders. The graph of the document type hierarchy shows the relationship between different document types.
- It gives users a context for locating and reviewing further information. For example, you can position the cursor over the frame template node and select the "view frame instance" items, one item at a time, to get a more detailed view – the corresponding frame instances, etc. We will give an example in the next chapter when we discuss the Display Tool.

4.2 Supporting USER Help

In our interface, we support the user Help function. The proposed Help functions in the user interface are based on the work addressed in [15]. We provide two types of help to the user, namely, the technical and conceptual help functions.

- Technical help: It enables the user to interact with the system in an effective way. For example, the help function can highlight the role of a certain control option.
- Conceptual help: It supports the user to overcome problems during the information retrieval process. We divide conceptual help function into two different detailed help functions.

1. Terminological help: It enriches the user by providing the vocabulary by suggesting a list of synonymous terms when needed. For example, when the user only knows part of the word he/she wants to search. For example, by entering a word "ja", the system cannot find the key term for ja and the system displays all strings that includes "ja" as a substring, such as "jack", "jason", "james", "jane", "Vijaylakshmi Gaddipati", etc. The user can then select one from the list of these words and then proceed continuously to search.
2. Strategic help: It improves effectiveness when the user conducts a retrieval process. For example, when the resulting set contains no match for a given query, the system makes suggestions to help the user refine the query. In our system, the operations "AND" and "OR" are provided in the input query. In the next Chapter, we will see the input query will be normalized through the Topic Parser. As a matter of fact, the "AND" operation can produce an empty query result, while the "OR" operation typically does not. For example, from a normalized topic, the "OR" operation

combines all possible "AND" subtopics to form disjunctive predicates. If every "AND" subtopic produces an empty result, the final result will also be empty. In this case, we list all possible "AND" subtopics to the user. The user can then change at least one "AND" subtopic to try to avoid an empty result.

Help is provided through a dialogue – an interaction between the user and the system. There are different kinds of forms that can be adopted during the interaction:

- Contextual vs. Generic: The type of help depends on the nature of the query. If it depends on the specific user's behavior or situation (such as, the user aids one of their terms which are synonymous to the key term to the query), the response is contextual. It is generic if it refers to general aspects or guidelines (such as, the user is advised to reduce the number of AND operator in such a way as to enlarge the number of retrieval results).
- Prompted vs. Unprompted: Help will prompt if it follows an explicit user request (such as, if the user asks for synonymous terms). Help will be unprompted if the response is given automatically by the system in certain situations (such as, the user's action is inconsistent with the query by reducing synonyms used to make the retrieval result-set empty).
- User- vs. System-controlled. In a system-controlled dialogue, a list of terms is presented to the user and he/she is asked for confirmation or selection, and continuation to proceed. A user-controlled dialogue may be an interaction by which the system provides the user a browser and the user has to decide how to navigate it.

4.3 Functionality for User Interface

We provide a graphical interface that allows the user to conduct the following functions. (Examples for the following features will be given in the next chapter when we discuss the Display Tool.)

1. Provide the system with a query which states the information needed in terms of a Boolean expression. In addition the graphical interface allows the user to enter a query using natural language, in which case the system displays the corresponding Boolean Expression. And the user can use the Boolean Expression to continue searching. As the user gains experience from using Boolean Expressions, he/she can input the query in the form of Boolean Expression directly later.
2. View the content in a graph form and to operate on the graph. For example, when the folder organization is displayed, the user can double click on a node and get the corresponding frame instances and original document.
3. Select terms to be inserted into a query reformulation. Using the Help function, the user can get suggestions for modifying the query.
4. View a specific part of the graph and restore the original graph when using "Zoom in" and "Zoom out". When we use "Zoom in" to get the specific part of the original graph, the connections between this specific part and the rest part of the original graph are not lost.
5. Classify retrieved information into categories such as "useful", "not useful", "relevant", and "not relevant". Then the user can select most relevant or most useful information or document to view first. This will save time from the user to find the

exact information, if the classification method is good. (This feature is not part of the current implementation but deferred for future work.)

CHAPTER 5

SERVICE LAYER

In this chapter, we present in detail the second part of the browser subsystem – the service layer. The service layer provides services to the front-end layer through a search of the storage layer. This is the main part of the browser subsystem. To improve the system's performance, we also introduce the Reusable Base. We divide the service layer into three phases: the preprocessing phase in which a query is formalized; the discovery phase in which the search is performed; and the post-processing phase in which the search results are organized. We address each phase separately.

5.1 Preprocessing Phase

The preprocessing phase is very important for each search. An efficient preprocess can speed up a search. We address each of the three different processes of the preprocessing phase separately.

5.1.1 Folder Organization Process

The Folder Organization Process is used when the user tries to look at, or search in a particular folder. When a query is a folder name, the folder organization process is used. After the user enters input, the system checks to see if it is a folder name. If it is, the correct folder name is sent to the Search Engine. If it is not, the system will display a list of folder names automatically and the user can select a particular folder name from

among them, and send it to the Search Engine. In the preprocessing phase, the main goal is to help the user select the correct folder name.

5.1.2 Document Type Hierarchy Process

The preprocessing phase for the Document Type Hierarchy Process is similar to that for the Folder Organization Process. The Document Type Hierarchy Process is used when the user tries to look at a particular document type or tries to find a particular document type in the document type hierarchy. The input for the Document Type Hierarchy Process is the name of one particular document type. After the user enters a query, the system checks to see if the input is a document type. If it is, the input is sent to the Search Engine for the next step. If it is not, the help function will give the user a list of names of document types and the user can select one and input it again. After the preprocessing phase, the correct document type's name is sent to the Search Engine.

5.1.3 Query Refinement Process

For the Query Refinement Process, the Topic Parser is the important component in the preprocessing process. A clear topic statement not only produces a more efficient search, but it will also help a user clarify his/her understanding of the system and what he/she wants from the search process. The function of Topic Parser is to translate the user's query into a normalized topic and to split the normalized topic into smaller basic simple topics (without "ANDs" and "ORs" between). It includes five function units: Topic Input, Key-term Replacement, Object Identification, Topic Transformation and Topic Splitter.

The Topic Parser converts the user's original query into a form that is easier and more efficient for the browser to perform a search. The following is the structure of the Topic Parser:

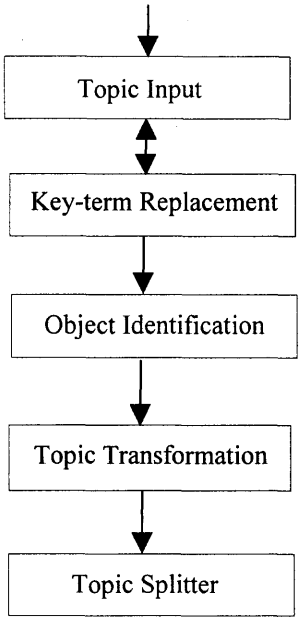


Figure 5.1 Flow of Topic Parser

Topic Input

The input is a query issued by the user. Experienced users usually can use SQL or Boolean expressions to input their queries. Novice users tend to use natural language in their inquiries. SQL is more difficult for users than either Boolean expressions or natural language. Users need to learn SQL syntax in order to use it. For ordinary non-expert users SQL is not a good choice. Natural language is the easiest for most users, especially novice users. But, such a natural language query is not a computer-friendly encoding, which means it is difficult for a computer to deal with using current technology.

Compared to SQL or natural language, Boolean expressions are easier than SQL for users because SQL has a more complex syntax, and is easier than natural language for the computer. Hence, we will encourage users to employ a Boolean expression as their input. In the future, when natural language query is entered, the corresponding Boolean expression will be displayed to the user once the search is completed. Then, the next time, the user will know more about the best ways to structure a query by entering a Boolean expression directly. We provide AND, OR, (, and) operations for Boolean expressions. AND has precedence over OR. The following is an example of users' input:

(dsanders OR png) AND CIS

Key-term Replacement

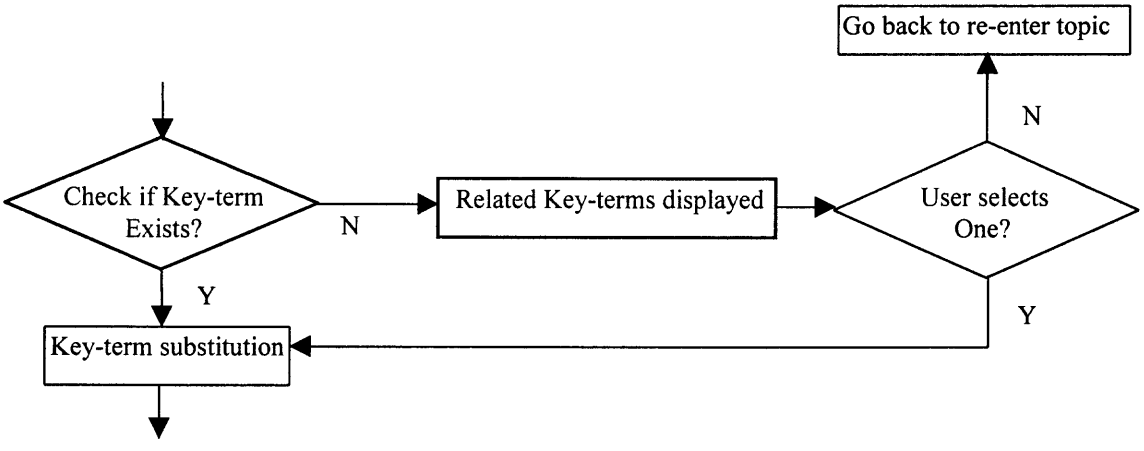


Figure 5.2 Flow of Key-term Replacement

The Key-term Replacement is used to change user's words into terms that are used by the system or to give the user related key-terms if there is no match in the thesaurus. Figure 5.2 is a flow diagram for the Key-term Replacement process. A user's input may contain his/her own words because he/she does not know the exact terms used by the system. Therefore, we replace them with the key terms that are used in TEXTPROS. It is

convenient and much more efficient for the system to do a search using key-terms. To do this, we should consult with the thesaurus that provides a mapping between the key-terms and its synonymous terms. If we cannot find a key-term corresponding to the user's input, we provide several related choices to the user. This is done by Related Key-term Display. Here we consider two situations: 1. Display all key terms in which the user's input is a substring; 2. Display all key terms that are no more than two letters different from the input. For the first case, it is very useful when you only know part of the query. For example you want to look for the author of a book, but you only remember his/her name is *Jul*. When you input Jul, and the system cannot find the key term for it. Then the system will display all key terms that include Jul as a substring and the user picks one he/she needs. The second case deals with a mistyping of at most two letters. It includes all Key-terms that differ by one letter and those that differ by two letters from the input. It also considers letter reversed as a special case. It is useful for correcting general typographical errors. (Of course, we could consider other situations, such as words that differ by three letters, those that differ by four letters, words that are similar sounding, but have different spelling, etc. Since it is not conceptually different to include broader categories of word-similarities, we will restrict ourselves to the categories of word similarities listed here in this dissertation.) A user can use one of the key-terms the system provides or re-enter a topic. For example, if the user enters IS, a term that is not a system key-term, the system provides several words: CIS, INS, each of which is one or two letters different from IS, or each of which contains IS as a substring. The user has a choice to use CIS as one of the subtopics or enter a new query.

After the Key-term replacement, the example given above might be changed to: (D.Sanders OR P.Ng) AND CIS

Object Identification

The Object Identification is used to find the type of each term in the topic. There are four types of objects in TEXPROS: folder, frame template, attribute and value. Through the Object Identification, each term is associated with the types and represented as an object – ObjType (ObjName). If one term has more than one type, it is represented as ObjType1 (ObjName) OR ObjType2 (ObjName)... which means we use OR operator to associate all related types to that object. The example above is transformed to: (Value (D.Sanders) OR Value (P.Ng)) AND (Value (CIS) OR Folder (CIS))

Topic Transformation

The Topic Transformation is used to normalize a query. After the completion of the Topic Transformation, the topic is in disjunctive normal form, which means there are disjunctive relationships among each group and there are conjunctive relationships among each term in each group. After this step, the example above becomes:
(Value (D.Sanders) AND Value (CIS)) OR (Value (D.Sanders) AND Folder (CIS))
OR (Value (P.Ng) AND Value (CIS)) OR (Value (P.Ng) AND Folder (CIS))

Topic Splitter

When the Topic Splitter receives a normalized query from the Topic Transformer, it splits the topic into some smaller basic subtopics if possible. First, it separates the

normalized query into several groups. Each item in a group has only a conjunctive relationship to other items in the group and the relationship between groups is disjunction. Secondly, the Topic Splitter separates each group into several items. Each item is an object.

The example above is changed to:

group1: Value(D.Sanders), Value(CIS).

group2: Value(D.Sanders), Folder(CIS).

group3: Value(P.Ng), Value(CIS).

group4: Value(P.Ng), Folder(CIS).

Eventually we will get all different objects. In the above example they take the form: Value (D.Sanders), Value (CIS), Folder (CIS), and Value (P.Ng). These objects are now ready for the next step – the discovery phase.

5.2 Discovery Phase

In this section we discuss the second phase of the service layer – the discovery phase. The discovery phase includes three parts: a Search Engine, a Reusable Base and a Consistency Agent. The task of the discovery phase is to find the material the user is interested in and to keep the material in the Reusable Base consistent with that in the System Catalog. The Search Engine provides algorithms to search the Reusable Base and the System Catalog. The Reusable Base is used to improve the efficiency of the search process. Finally, the Consistency Agent guarantees consistency between the Reusable Base and the System Catalog.

5.2.1 Reusable Base

The goal of the Reusable Base is to reuse the results of previous searches through adaptation of the previous searches. The Reusable Base also evaluates a proposed solution to a search and, finally, it learns from this new experience and stores the solution for future use [44]. Using Reusable Base leads to a new view of databases where data are not only stored occasional access, but are also intended to be reused. To perform this task, we store only basic components in the Reusable Base (i.e. only simple topics in which there are no "AND" and "OR" commands, and their related OP-Nets).

5.2.1.1 Structure of the Reusable Base The Reusable Base will improve retrieval process efficiency for two reasons: First, the Reusable Base attempts to hold information that is most related to the user previous desires. Second, the information stored in the Reusable Base is more easily used to form the OP-Net than that in the System Catalog. In addition, the structure of the Reusable Base is designed to be accessed quickly. To achieve this goal, the Reusable Base is designed to have two parts: one part is used to achieve accessing very quickly; and the other part is used to hold the information the user may be most interested in. Hence, the Reusable Base is divided into two levels: the first level is Cache and the second level is the Basic Component Base. The Cache is used to achieve accessing very quickly, and the Basic Component Base is used to store the information that may be of interest to the user. We put a small Victim Cache between these two levels. The Victim Cache is an attempt to improve the retrieval further. The following is the structure of the Reusable Base:

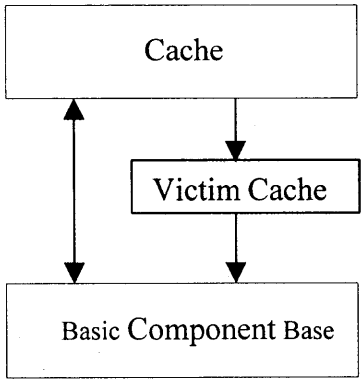


Figure 5.3 Structure of Reusable Base

5.2.1.2 The First Level of the Reusable Base – Cache The first level of the Reusable Base is the Cache. In this dissertation the term "cache" is used to mean a memory buffer. It is different from Cache in OS, which is hardware. Based on the assumption that the probability that the results of a previously searched query will be needed again in a subsequent search is higher than those that have not yet been searched by a particular user, the Cache only holds a limited number of basic components that were recently used. For the same reason, we use LRU algorithm to place components into the second level when a new component enters the Cache and the Cache does not have enough space to hold it. The following is the logic structure of the first level of the Reusable Base:

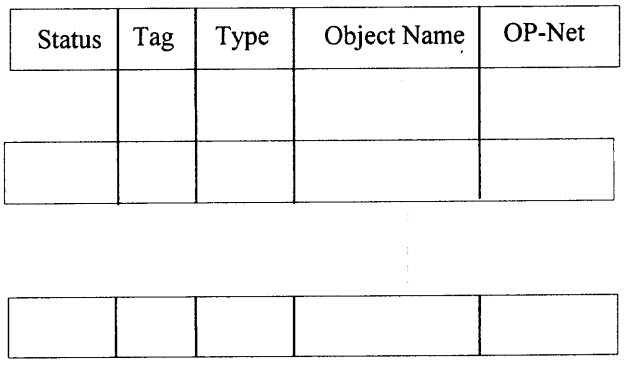


Figure 5.4 Structure of First Level of Reusable Base -- Cache

There are five columns in Figure 5.4: status, tag, type, object name and pointer. We use status to designate when a basic component first occurs in the Reusable Base. "Status" has two values: 1 that means that it is the first occurrence of a basic component in the Reusable Base, or 0 that means the basic component comes from the second level of the Reusable Base. When we use the LRU algorithm, we must decide to replace or delete the basic component (i.e. when the status value is 0, we simply delete it because it is already in the second level; if the value of the status is 1, we must write it to the second level because it is new to the Reusable Base). "Tag" is used to tell which component should be replaced when using the LRU algorithm. When the component is put in the first level or it is being used, the value of its tag is set to 0 and the values of tags for other components in the first level will be incremented. When using the LRU algorithm to replace components, the component with the largest tag value is selected. "Type" is used to distinguish different types of object. We give the following values for type:

0 -- folder

1 -- template

2 -- attribute

3 -- value

"OP-Net" is used to hold the OP-Net corresponding to the object.

In the Cache, the OP-Net is represented by the structure ObjNodeLinkList, which is a data structure that is used inside. To access the information in the Cache, the class Cache is defined, and it includes the following functions:

1. Cache(): It is a constructor that is used to initialize the object.

2. void resync(String oName, String oType): when an OP-Net is deleted from the Basic Component Base, check to see if it is in the first level – Cache. If yes, mark it as a new component to the Reusable Base.
3. void insert(Obj obj, ObjNodeLinkList objNLList): Insert a query and its OP-Net, which come from the System Catalog, into the Cache.
4. void manageRBase():When a new component enters and the Cache is full, use LRU algorithm to remove the least recently used one. Create a thread to deal with writing to the Basic Component Base. This last step minimally affects the retrieval time.
5. void printReuseBase(): Print all the items out in the Cache.
6. void saveData(): After retrieval is finished, check the status of all slots in the Cache, and call function putToSecondLevel() to write all new components to the second level of the Reusable Base – the Basic Component Base.
7. void purgeRBase(): Delete all items in the first and second level of the Reusable Base.

5.2.1.3 The Victim Cache Victim Cache is a small cache placed between the first and the second levels of the Reusable Base. Initially, we included the Victime Cache because we believed that it might be useful to improve the efficiency of retrieval process. This assumption was based on two considerations: 1. The probability that the result of a previously searched query will be needed again in a subsequent search is higher than for those never searched, which is exact the same reason we have the first level of the Reusable Base – the Cache. 2. When a component is deleted from the first level, it is not necessary to write it to the second level immediately if the victim cache is used. Hence

writing a component from the first level of the Reusable Base to the second level of the Reusable Base will not affect the retrieval time. The Victim Cache contains only components that are discarded from the first level because of misses, (which means that the object cannot be found in the first level). The Victim Cache is checked on a miss to see if it has the desired data before going to the second level of the Reusable Base. If it is found there, the victim component and the component from the first level are swapped. The architecture of the Victim Cache is as following:

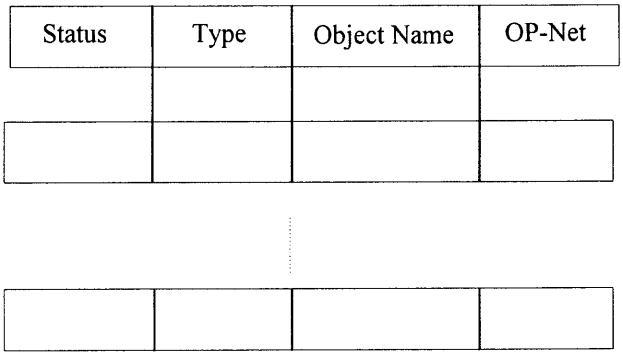


Figure 5.5 Structure of Victim Cache

Status, Type, Object Name and OP-Net in Figure 5.5 have the same meaning as in the first level of the Reusable Base.

When we implemented the Victim Cache, we found that the Victim Cache is equivalent to an extension of the first level of the Reusable Base – the Cache. For example if there are n entries in the Cache and m entries in the Victim Cache, it is the same as we having m+n entries of the Cache without the Victim Cache. Hence, we deleted the Victim Cache from the prototype implementation.

5.2.1.4 The Second Level of the Reusable Base – Basic Component Base The second level of the Reusable Base is the Basic Component Base. It is a small database. When we have basic components, we put it in the Reusable Base. Initially the Reusable Base is empty. The Reusable Base gets the basic components gradually as the System Catalog is searched. After we have enough basic components in the Reusable Base, most of the time we can construct the needed graphical information, such as OP-Net, directly from the Reusable Base. This means that we do not need to scan the System Catalog in every search. This reduces the response time for a search by at least two times (see chapter6). We mentioned before that basic components in the Reusable Base means simple topics in which there are no "AND" and "OR" commands, and their related OP-Nets. The following is the structure of Basic Component Base (N used in the following table represents some number):

```
table QueryGrp (
    QueryName    varchar2(N),
    QueryType    varchar2(N),
    OP_Net_ID    number
);

table Parent_child_relation(
    OP_Net_ID    number,
    ParName      varchar2(N),
    ParType      varchar2(N),
    ChildName    varchar2(N),
    ChildType    varchar2(N)
);
```

```
table Obj_FI(
    OP_Net_ID    number,
    ObjName      varchar2(N),
    ObjType      varchar2(N),
    FI_ID        number
);
```

Figure 5.6 Structure of Second Level of Reusable Base – Basic Component Base

Table QueryGrp describes the relationship between a query and its corresponding OP-Net. A simple query (without AND or OR commands) is an object. Hence it has type and name. QueryName represents the name of the query, and QueryType represents the type of the query. OP_Net_ID is an ID number given to the query and it is used to get the information for the corresponding OP-Net.

Table Obj_FI describes each node in the OP-Net. OP_Net_ID is an ID number of the OP-Net. Every node in the OP-Net is a FIR (frame instance repository), and FIR includes an object and its frame instances. Hence we use ObjName to represent the name of the object, use ObjType to represent the type of the object, and use FI_ID to represent a frame instance of the object.

Table Parent_child_relation describes the parent-child relationship between nodes in the OP-Net. OP_Net_ID is an ID number of the OP-Net. ParName and ParType represent the name and type of the parent and ChildName and ChildType represents the name and type of the child.

Based on the assumption that the second level of the Reusable base is not too large, we put all information about every node in the OP-Net into table Obj_FI, and all information about the parent-child relationships in table Parent_child_relation. If the second level of the Reusable Base is relatively larger, we can divide each table into four tables. Each of the four tables deals with a different types of the query: the first deals with folders, the

second deals with frame templates, the third deals with attributes and the fourth deals with the values. When we get the type of the query, we need search only one of the four tables instead of one large table. This obviously can save retrieval time.

To access the information in the Basic Component Base more easily, a class BaseManager is defined, which holds all the functions needed to access the Basic Component Base. We list the functions as follows:

1. BaseManager(): It is a constructor that is used to initialize access to the database.
2. int getOpId(Obj obj): Given a query, get the ID number for the corresponding OP-Net. If there is no query in the table QueryGrp that matches the given query, return 0.
3. Vector getFI(int Op_Id, String objName, String objType): Given an ID number of the OP-Net and the object, get the frame instances of the object in the OP-Net.
4. Vector getObjFi(int s): Given an ID number of the OP-Net, get all nodes in the OP-Net and the relationships between the nodes and their frame instances. This function calls the function getFI(int Op_Id, String objName, String objType) to get the frame instances of each nodes.
5. Vector getPar_Chr(int s): Given an ID number of the OP-Net, get the parent-child relationships of nodes in the OP-Net.
6. void printQueryGrp(): Print all the entries in the table QueryGrp.
7. static int availableId(): Get the ID number of the OP-Net in the table QueryGrp that will be deleted from the second level of the Reusable base. Based on the assumption that every component is equal, Round Robin algorithm on the slot of the second level of the Reusable Base is used here. This function is used in the following situation: When a new component needs to be inserted into the second level, but the second

level is full. Then use function availableId() to get the record number that will selected to be deleted.

8. int getAvailOpId(): Assign an available ID number to the corresponding OP-Net for a new incoming component.
9. int getNoQueryGrp(): Get the number of the records in the table QueryGrp.
10. void addQueryGrp(Obj obj, int Op_Id): Add an element to the table QueryGrp.
11. void addParChr(int Op_Id, String parName, String parType, String chrName, String chrType): Add an element to the table Parent_child_relation.
12. addObjFi(int Op_Id, String ObjName, String ObjType, Vector Fi_Id): Add an element to the table Obj_FI.
13. void delQueryGrp(Obj obj): Given an query, delete the corresponding element from the table QueryGrp.
14. void delQueryGrp(int opId): Given an ID number of the OP-Net, delete the corresponding element from the table QueryGrp.
15. void delParChr(int Op_Id): Given an ID number of the OP-Net, delete all corresponding items from the table Parent_child_relation.
16. void delObjFi(int Op_Id): Given an ID number of the OP-Net, delete all corresponding items from the table Obj_FI.
17. void delOpNet(Obj obj): Given an query, delete the corresponding OP-Net from the Basic Component Base. First it calls function getOpId(obj) to get the ID number of the OP-Net according to the query. Then it calls function delQueryGrp(obj) to delete the corresponding item from the table QueryGrp. And then it calls the function delParChr(ID) to delete all corresponding items from table Parent_child_relation.

Finally it calls the function delObjFi(ID) to delete all corresponding items from the table Obj_FI.

18. Vector getObj(int Op_Id): Given an ID number of the OP-Net, get the corresponding query from the table QueryGrp.
19. void delOpNet(int Op_Id): Given an ID number of the OP-Net, delete all corresponding items from the Basic Component Base. Similarly as the function delOpNet(Obj obj), it calls function delQueryGrp(obj), delParChr(ID) and delObjFi(ID). In addition, when the component is deleted from the Basic Component Base, it must be checked to see if it is in the first level of the Reusable Base – the Cache. If it is in the Cache, the status of it in the Cache should be changed and made to be a new component in the Reusable Base before it is deleted from the Basic Component Base. This is done by calling the function resync() that is defined in the management of the Cache.
20. void purgeSecondLevel(): Delete all items in the first level and the second level of the Reusable Base.
21. Update_Level2 (Type, Name): This function is used when the component of the System Catalog is changed and the Consistency Agent sends a message to the Reusable Base. Update_Level2 keeps the Reusable Base consistent with the System Catalog. First find the corresponding query (object); then modify its corresponding OP-Net. (We will talk the detail later.)

5.2.1.5 Cooperation between the First Level and the Second Level of the Reusable Base Although both the first level and the second level of the Reusable Base hold information about the OP-Net, the OP-Net is stored in different forms in the two levels. In the first level – the Cache, the OP-Net is represented by a data structure ObjNodeLinkList, which simplifies drawing the graph. In the second level – the Basic Component Base, the OP-Net is represented by several tables that are designed to be stored in the database. Hence, we provide some functions to take care of the format difference between the OP-Nets in these two levels. The following are some of the functions:

1. ObjNodeLinkList getFromSecondLevel(Obj query): Given the query, get the corresponding information about the OP-Net in the second level, and change it to the ObjNodeLinkList, which is used to represent information concerning the OP-Net in the first level of the Reusable Base – the Cache.

   Algorithm:

   BEGIN

   Call function getOpId(query) to get the ID of the corresponding OP-Net;

   Call function getObjFi(ID) to get all nodes and their frame instances in the OP-Net. And put the result in Vector node;

   Get the number of the nodes in the OP-Net. Assume it is N;

   Call function getPar_Chr(ID) to get the parent-child relationships between nodes in the OP-Net. And put the result into the Vector par_chr;

   For every element in Vector node, construct the corresponding ObjNode:

{

Get the ith element of the Vector node;

Get the ith element's name, type and its frame instances;

Get all the children of the ith element by searching the Vector par-chr;

Construct the ObjNode and insert it into the linked list of the ObjNode;

}

return the result – an instance of the ObjNodeLinkList;

END

2. void putToSecondLevel(ObjNodeLinkList onll, Obj query): Given a query and its OP-Net that is represented as a ObjNodeLinkList, change the forms of the OP-Net and put it into the second level. This function is used in two situations:

- When the search is finished, write all the new components from the first level directly backs into the second level of the Reusable Base.
- When the component needs to be deleted from the Cache, check to see if it is a new component to the Reusable Base. If yes, write it to the second level of the Reusable Base.

Algorithm:

BEGIN

Call function getAvailOpId() to get the ID number for the OP-Net that will be add to the Basic Component Base.

Call function addQueryGrp(query, ID) to insert the item into the table QueryGrp;

For every ObjNode in the ObjNodeLinkList

```
    {
        Get the name of the ObjNode and assume it is Name;
        Get the type of the ObjNode and assume it is Type;
        Get the frame instances of the ObjNode and assume it is FIR;
        Call function addObjFi(ID, Name, Type, FIR)to insert the item into the table
        Obj_Fi;
        Get the children of the ObjNode;
        For every child:
        {
            Get the name of the child and assume it is chrName;
            Get the type of the child and assume it is chrType;
            Call function addParChr(ID, Name, Type, chrName, chrType) to insert
            the item into the table Parent_child_relation;
        }
    }
END
```

5.2.2 Search Engine

The Search Engine provides algorithms for the search in the System Catalog and the Reusable Base. The Search Engine provides different algorithms for three different processes. We will discuss each of them separately.

5.2.2.1 Folder Organization Process The goal of the Folder Organization Process is to establish the folder organization based on the user's query – i.e. the folder name. After getting the correct folder name, the Search Engine will go to the System Catalog to continue the search.

Given a folder name fd_name, the search process proceeds as follows:

1. Find all the ancestors of fd_name using BFS (Breadth-First-Search). We use BFS algorithm because it is easy for implementation here.

   Put the fd_name in a Vector par_folder[0];

   Set index for search: initially set index j = 0;

   while (true)

   {

   Get the jth element of Vector par_folder;

   Search the System Catalog to get the parents of jth element and put them into Vector par_folder:{

   First we use getFDParAsc (fd_name) to get the parent relation of fd_name from the System Catalog;

   And from the parent relation of fd_name, extract the parents of fd_name.

   }

   If there are no parents for jth element, break;

   Increment j;

   }

2. Find all the descendants of fd_name using BFS:

   The algorithm for finding all the descendants of fd_name is similar to the algorithm for finding all the ancestors of fd_name. Put the result into Vector child_folder.

3. Combine the result from step1 and step2, get the folders in the Folder Organization:

   Do the union of Vector par_folder and Vector child_folder. Assume the result is put into Vector Temp, we get Temp = par_folder U child-folder.

Finally, all folders in the Folder Organization will be in Vector Temp.

5.2.2.2 Document Type Hierarchy Process The goal of the Document Type Hierarchy process is to establish the document type hierarchy based on the user's query (i.e. the name of a document type). After getting the correct name of the document type, the search engine will go to the System Catalog for the search.

Given a name of a document type dt_name:

1. Find all the ancestors of dt_name using BFS (Breadth-First-Search). We use BFS algorithm because it is easy for implementation here.

   Put the dt_name in a Vector par_document[0];

   Set search index j = 0;

   while (true)

   {

   Get the jth element of Vector par_document;

   Search the System Catalog using getFTParN (dt_name) to get the parent of jth element and put it into the Vector;

   If there are no parents for jth element, break;

Increment j;

}

2. Find all of the descendants of dt_name using BFS:

   The algorithm to find all of the descendants of dt_name is similar to the algorithm for finding all of the ancestors of dt_name. The results of the search are placed into Vector child_document.

3. Combine the results from steps 1 and 2, get the document types in document type hierarchy: Do the union of Vector par_document and Vector child_document.

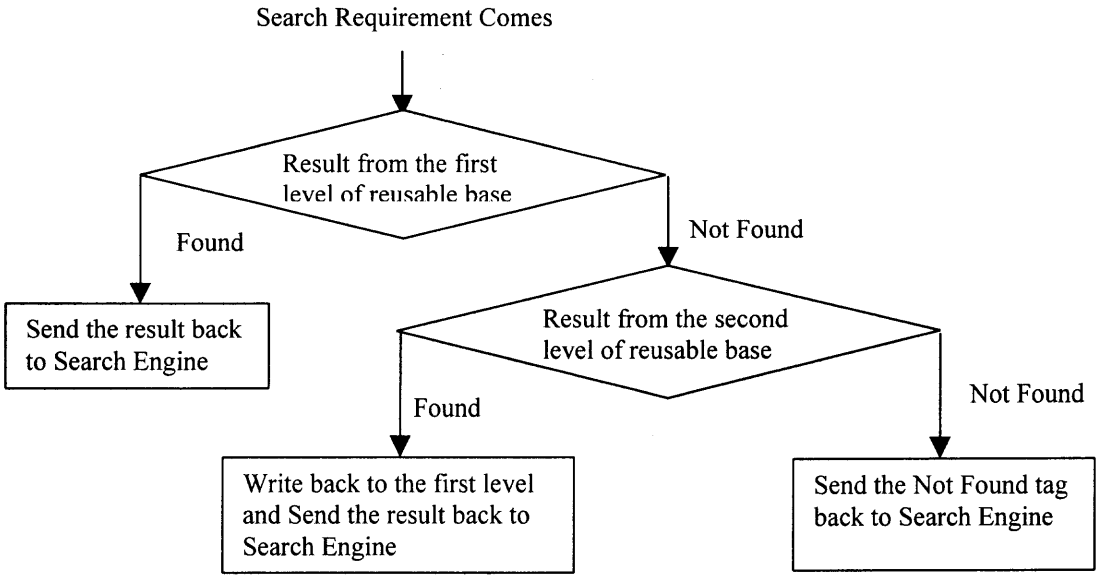
Assume the result is put into Vector Temp, we get Temp = par_document U child_document.

Finally, all document types in the document type hierarchy will be in Vector Temp.

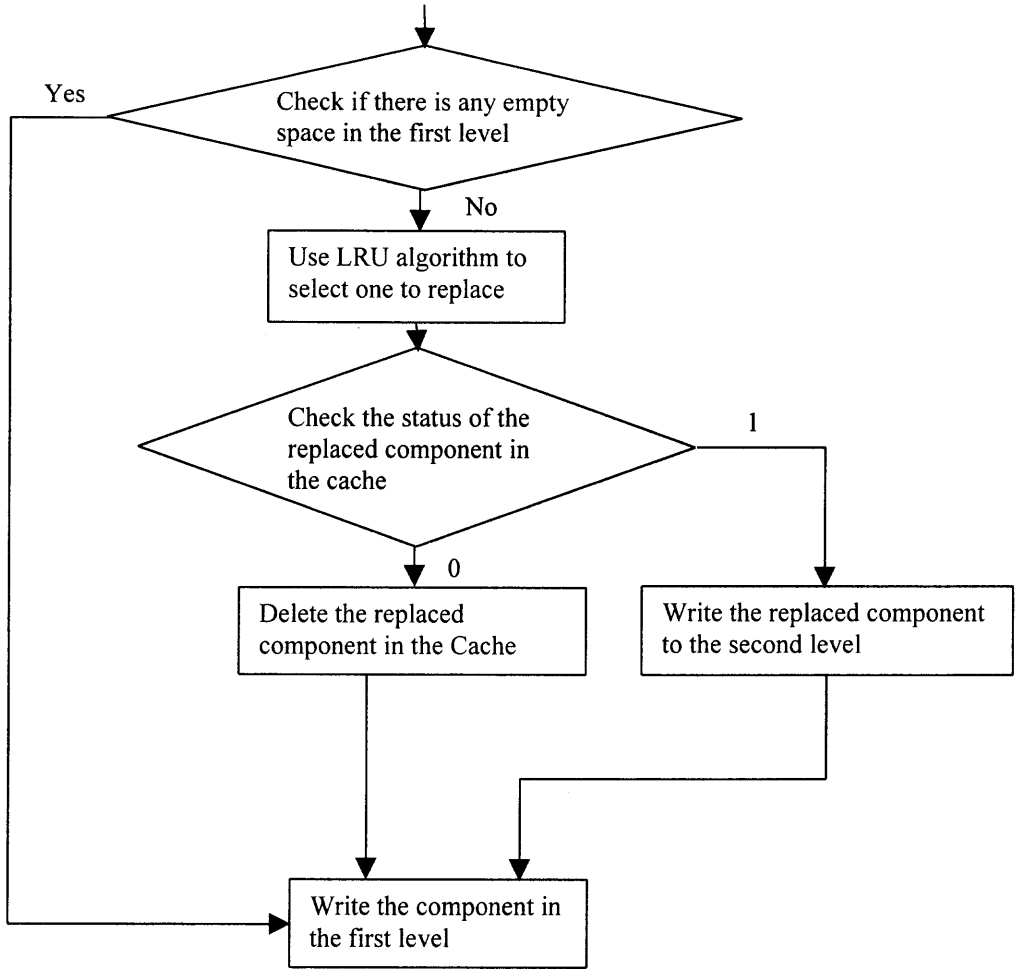
5.2.2.3 Query Refinement Process In the Query Refinement Process, the Search Engine receives inputs from the Topic Parser. Based upon the query, it provides algorithms to search the Reusable Base and the System Catalog. After getting the result from the Reusable Base and the System Catalog, it sends the result to the Network Constructor. The following are some of the algorithms:

1. The algorithm to search the Reusable Base: It was developed as part of this dissertation. Given a query, the search process identifies the corresponding OP-Net from the Reusable Base. The Reusable Base is used to achieve efficient retrieval. Based on its two level structure, we choose the following algorithm: Use a string-match algorithm to see if the object of the query occurs in the Cache. If it does, return the corresponding OP-Net. Otherwise, go to the second level of the Reusable Base

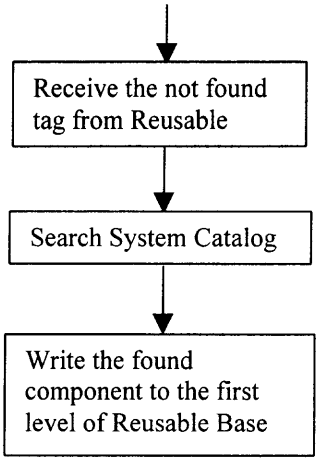
and check to see if the component can be found there. If it can, send the OP-Net back and write it to the Cache. Otherwise, send NULL back and go to the System Catalog to find it.



2. The algorithm to write to the first level: When a new component results from a query, it is put into the first level of the Reusable Base if there is any empty place there. Otherwise, a component from the first level should be deleted. We use LRU algorithm to select one of the existing components to be replaced based on the reason for the existence of the first level (the first level holds components that is recently used).



3. The algorithm to write to the first level when the match is not found in the Reusable Base: When the result cannot be found in the Reusable Base, we must go to the System Catalog for search.



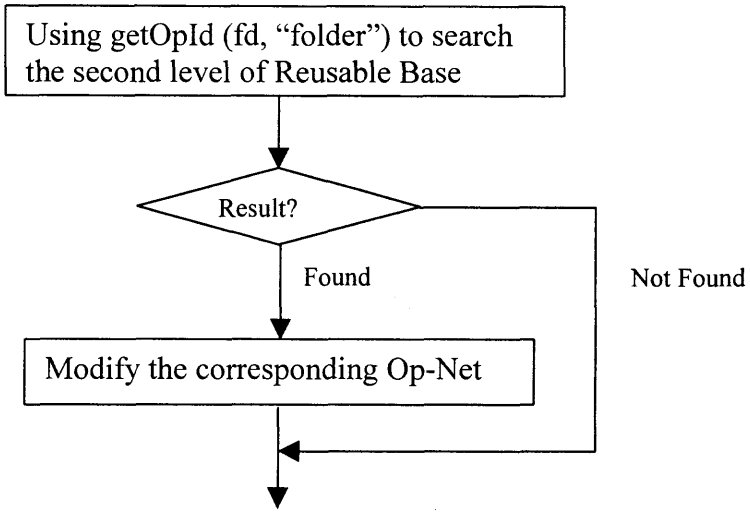
Call algorithm 2 to write the found match to the first level of the Reusable Base.

5.2.3 Consistency Agent

We say that the Reusable Base is a dynamic database. There are two reasons for this statement. The first reason is that initially the Reusable Base is empty. When a new search topic enters the system, it is added to the Reusable Base. Gradually the numbers of elements in the Reusable Base increases. The second reason is that it changes dynamically according to changes in the Folder Organization and the Document Type Hierarchy in the System Catalog. When a new document comes into the TEXPROS system, it is classified and filed, and is put into corresponding folders and a frame template. We must check to see if, in this process, the elements in the Reusable Base are affected. The last process that assures consistency between the Reusable Base and the System Catalog is the purpose of the Consistency Agent. Through this process, the user will not encounter conflicting result.

Assume that a new document d comes into the system through classification and filing processes of TEXPROS. Further assume that it is put into a folder set FD and its document type is t. In addition assume the set of attributes in d is Att, and assume the set of value in d is Val. Now we need go to the Reusable Base to check to see if there are any changes.

- For each element fd in set FD, check to see if there is an item whose name is fd and whose type is folder. If yes, adjust corresponding Op-Net. If no, do nothing.



Next we discuss the modification of the corresponding Op-Net. Assume the original Op-Net is Op_Net1. First we need to construct the Op-Net Op_Net2 for document d. Since we know the set of attributes in d is Att and we assume the set of values in d is Val.

Op_Net2 = [Top2, R2 (V, V)], where

Top2 = fd;

V = {fd} U {t} U Att U Val;

Initially for every v1, v2 in V, R2 (v1, v2) = 0;

R2 (fd, t) = 1;

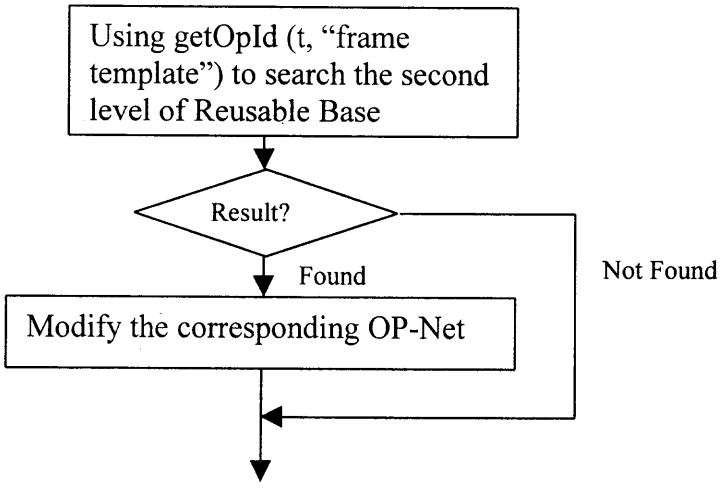
For every att in Att, set R2 (t, att) = 1;

For every att in Att and its corresponding val in Val, set R2 (att, val) = 1.

Now we obtain Op_Net2.

The new OP-Net corresponding to folder fd is Or (Op-Net1, Op_Net2).

- Check to see if there is an item whose name is t and whose type is frame template. If yes, adjust the corresponding OP-Net. If no, do nothing.



Next we discuss the modification of the corresponding Op-Net. Assume the original OP-Net is Op_Net1. First we need to construct the OP-Net Op_Net2 for document d. We know that the set of attributes in d is Att and we assume the set of values in d is Val.

Op_Net2 = [Top2, R2 (V, V)], where:

Top2 = t;

V = FD U {t} U Att U Val;

Initially for every v1, v2 in V, R2 (v1, v2) = 0,

For each fd in FD, set R2 (fd, t) = 1;

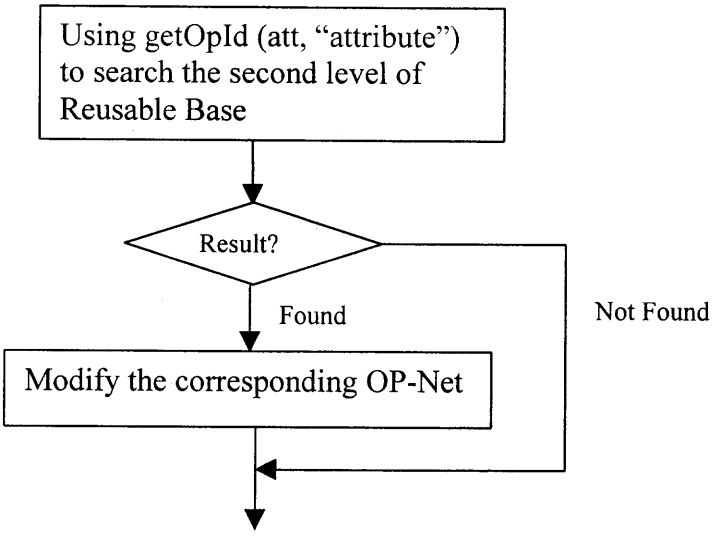
For every att in Att, set R2 (t, att) = 1;

For every att in Att and its corresponding val in Val, set R2 (att, val) = 1.

Now we get the Op_Net2.

The new OP-Net corresponding to template t is Or (Op-Net1, Op_Net2).

- For each element att in Att, check to see if there is an item whose name is att and type is an attribute. If yes, adjust corresponding Op-Net. If no, do nothing.



Now we discuss the modification of the corresponding OP-Net. Assume the original OP-Net is Op_Net1. First we need to construct the OP-Net Op_Net2 for document d. We know that the set of attributes in d is Att and we assume the set of values in d is Val. Assume the corresponding value for attribute att is val in Val,

Op_Net2 = [Top2, R2 (V, V)], where:

Top2 = att;

V = FD U {t} U {att} U {val};

Initially for every v1, v2 in V, R2 (v1, v2) = 0,

For each fd in FD, set R2 (fd, t) = 1;

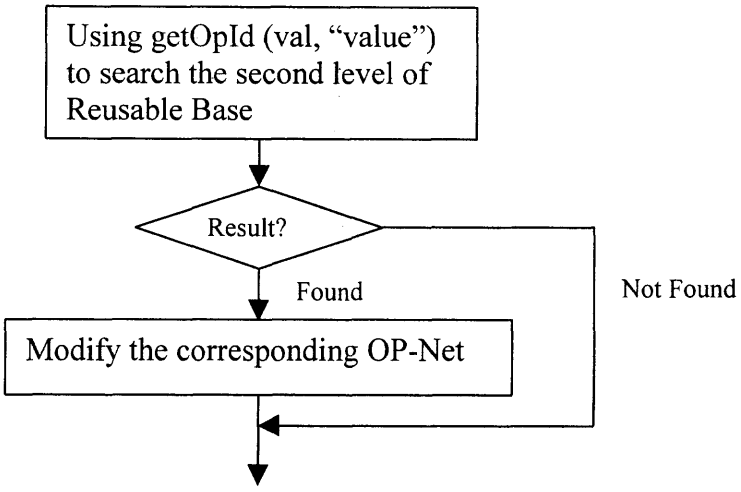
Set R2 (t, att) = 1;

Set R2 (att, val) = 1.

Now we get the Op_Net2.

The new OP-Net corresponding to attribute att is Or (Op-Net1, Op_Net2).

- For each element val in Val, check to see if there is an item whose name is val and type is a value. If yes, adjust corresponding Op-Net. If no, do nothing.



Now we discuss the modification of the corresponding OP-Net. Assume the original OP-Net is Op_Net1. First we need to construct the OP-Net Op_Net2 for document d. We know that the set of attributes in d is Att and we assume the set of values in d is Val. Assume the corresponding attribute for val is att in Att,

Op_Net2 = [Top2, R2 (V, V)], where:

Top2 = val;

V = FD U {t} U {att} U {val};

Initially for every v1, v2 in V, R2 (v1, v2) = 0,

For each fd in FD, set R2 (fd, t) = 1;

Set R2 (t, att) = 1;

Set R2 (att, val) = 1.

Now we get the Op_Net2.

The new OP-Net corresponding to attribute att is Or (Op-Net1, Op_Net2).

Because the OP-Net that is gotten from the System Catalog is in the form of data structure ObjNodeLinkList, function putToSecondLevel() is called to put the result into the Basic Component Base.

5.3 Post-processing Phase

The Post-processing phrase includes Constructors and the Display Tool, which organizes the results obtained from the discovery phase and displays the results to the user. According to the different processes, we further divide the Constructors into three small constructors: the Network Constructor, the Folder Organization Relationship Constructor, and the Document Type Hierarchy Relationship Constructor.

5.3.1 Network Constructor

After we obtain the relevant information from the Reusable Base or the System Catalog, we can construct an OP-Net. After a normalized query is obtained through the Topic Parser, every object involved in the query is sent to the Search Engine. The Search Engine searches the Reusable Base and the System Catalog, to get the OP-Net for each object. To obtain the final resulting OP-Net, some operations are needed to combine the separate OP-Nets that represents every object. To complete this task the AND and OR operators are necessary. Hence, we provide operations AND and OR between the OP-Nets:

AND (OP-Net1, OP-Net2) where OP-Net1 = [Top1, R1(V1, V1)] and OP-Net2 = [Top2, R2(V2, V2)], is an OP-Net = [Top, R(V, V)] in which:

Top = Top1 ^ Top2,

FI = {fi | fi is a frame instance ID that satisfies Pop1^Pop2}, where Pop1 is the predicate derived from the query Top1 and Pop2 is the predicate derived from the query Top2.

Define a temporary variable as Vtmp = {v | v in V1 or v in V2, if v in V1 and v in V2 then v.FI = union of v.FI1 and v.FI2},

And we get V = {v | v in Vtmp, intersection of v.FI and FI is not empty},

- For every v in V, v.FI = intersection of v.FI and FI.
- For every v1 and v2 in V, initialize R(v1, v2) = 0.
- For every v1 and v2 in V, if R1 (v1, v2) =1 or R2 (v1, v2) = 1, then set R (v1, v2) = 1.

OR (OP-Net1, OP-Net2) where OP-Net1 = [Top1, R (V1, V1)] and OP-Net2 = [Top2, R(V2, V2)], is an OP-Net = [Top, R(V, V)] in which:

Top = Top1 v Top2, V = V1 v V2,

- For every v in V, v.FI = union of v.FI1 and v.FI2.
- For every v1 and v2 in V, initialize R (v1, v2) = 0.
- For every v1 and v2 in V1, R (v1, v2) = R1(v1, v2).
- For every v1 and v2 in V2, R (v1, v2) = R2(v1, v2).
- For every v1(obj1) in V1 and v2(obj2) in V2:

1. If type of obj1 is folder and type of obj2 is template, the intersection set of v1.FI and v2.FI is not empty, set R(v1, v2) = R(v2, v1) = 1.
2. If type of obj1 is template and type of obj2 is attribute, the intersection set of v1.FI and v2.FI is not empty, set R(v1, v2) = R(v2, v1) = 1.
3. If type of obj1 is attribute and type of obj2 is value, the intersection set of v1.FI and v2.FI is not empty, set R(v1, v2) = R(v2, v1) = 1.

- For every v1(obj1) in V2 and v2(obj2) in V1:

1. If type of obj1 is folder and type of obj2 is template, the intersection set of v1.FI and v2.FI is not empty, set R(v1, v2) = R(v2, v1) = 1.

2. If type of obj1 is template and type of obj2 is attribute, the intersection set of v1.FI and v2.FI is not empty, set R(v1, v2) = R(v2, v1) = 1.
3. If type of obj1 is attribute and type of obj2 is value, the intersection set of v1.FI and v2.FI is not empty, set R(v1, v2) = R(v2, v1) = 1.

5.3.2 Folder Organization Relationship Constructor

Through the Search Engine, we obtain the result – a Vector V that contains all folders in the folder organization. To make the display easier, we construct a structure ObjectNodeLinkList L. We extend every element in V as an element in L. The ObjectNodeLinkList is a linked list for ObjectNodes. The structure of ObjectNode is as follows:

```
ObjectNode
{
        String ObjName;
        String ObjType;
        Boolean bVisited;
        ObjLinkList ollAdjList;
        Vector vFIR;
}
```

Every element in L contains folder name (ObjName), folder type (ObjType: of course it is a folder), a Boolean variable which indicates if the folder was visited (bVisited: used for status check when displaying), Vector FI (vFIR) which contains the frame instance in the folder, and a Object Link List (ollAdjList) which links all the children of the folder

together. These children should occur in V. Otherwise it cannot be in the folder organization. The algorithm for the Folder Organization Relationship Constructor is as follows: Assume the original input folder name is fd,

Initially Object Node Link List L is set to null;

Use getFIIDfromFDN(fd) to get the frame instances of the folder fd, and set it to FI;

For every element v in Vector V:

{

Construct the node for Object Node Link List L:

{

Set folder name to v;

Set folder type to Folder;

Set Boolean variable visited to false;

Use getFIIDfromFDN(v) to get the frame instances of v from the System Catalog, and set it to FI1;

Set the frame instances for v to intersection of FI and FI1;

Construct Object Link List to contain all the folders that occur in V and that are a child of v;

}

}

5.3.3 Documentation Type Hierarchy Relationship Constructor

The relationship constructor for the document type hierarchy is similar to the relationship constructor for the folder organization. Through the Search Engine, we obtain the result – a Vector V that contains all document types in the document type hierarchy. Similarly we construct a structure ObjectNodeLinkList L (with the same structure as in Folder Organization Relationship Constructor). We extend every element in V to be an element in L. The algorithm for the Document Type Hierarchy Relationship Constructor is as follows: Assume the original input frame template name is ft,

Initially Object Node Link List L is set to null;

Use getFIIDfromFTN(ft) to get the frame instances of the frame template ft, and set it to FI;

For every element v in Vector V:

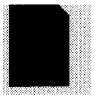
```
{       Construct the node for Object Node Link List L:
        {       Set name of the document type to v;
                Set type to Frame Template;
                Set Boolean variable visited to false;
                Use getFIIDfromFTN(v) to get the frame instances of v from
                system catalog, and set it for FI1;
                Set the frame instances of v to the intersection of FI and FI1;
                Construct Object Link List to contain all the document types that
                occur in V and that are a child of v;
        }
}
```

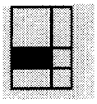
5.3.4 Display Tool

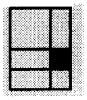
After getting the result from the Constructors, the Display Tool provides tools to display the results. There are some basic operations that are used by all processes, such as draw object and adjust layout.

1. Draw Object: we define an abstract base class drawObj, which is used when you want to draw an object. We know that there are four kinds of objects in TEXPROS: folder, frame template, attribute and value. Hence, we define four subclasses of base class drawObj. Each of them is responsible for drawing a folder, a frame template, an attribute or a value, respectively. Different kinds of objects have differently shaped polygons and it is easy for a user to distinguish among these different objects. The following are the different shaped polygons for each of the four different types of the objects:


Folder

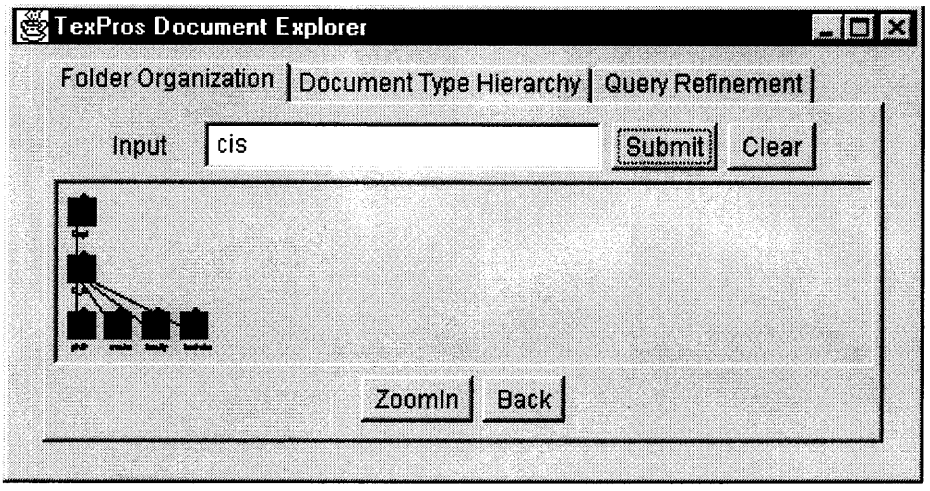

Frame Template


Attribute

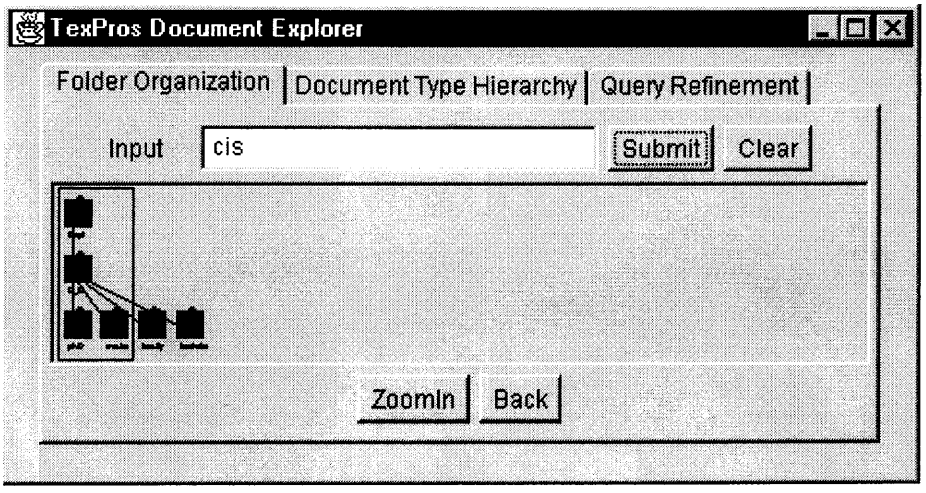

Value

2. Calculate nodes and lines: Get the nodes and lines, which will be drawn on screen. By scanning the structure we get from the Constructors (the Network Constructor or the Relationship Constructors), we establish a structure guiNode to hold all the nodes with their coordinates, and a structure guiLink to hold all lines including coordinates of starting and ending points. To get the coordinates of all nodes, we need to organize display layout, which involves how the picture is displayed on screen. This includes the row interval and column interval.
3. Zoom-in and zoom-out: We provide functions such as zoom-in and zoom-out for user. These tools make it easier for the user to look at an especially large picture. The user can use zoom-in to enlarge some specific part of the picture in order to make the specific part clearer. The user can use zoom-out to restore the original picture when he/she no longer wants to view the detailed graph. There are other tools that can enlarge the graph, such as scaling. Scaling can magnify the whole picture, but cannot enlarge only a part of the picture. When the graph is shown, usually the user is only interested in some part of it, and zoom-in and zoom-out tools can satisfy user's need.

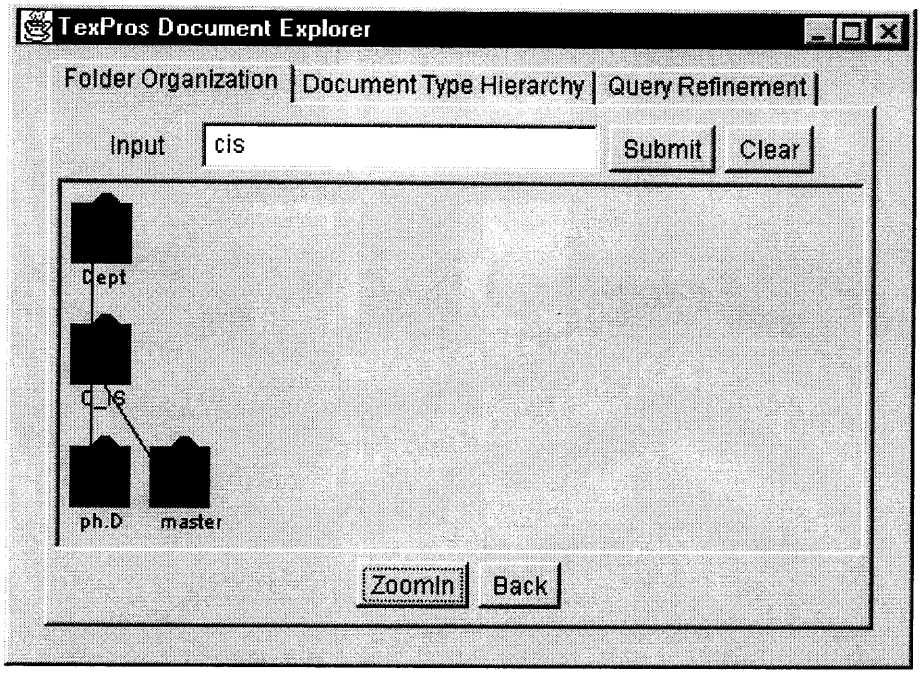
These tools provide a particularly convenient way for the user to view the part of the graph that is of interest to him/her. We give an example here. The following is the original folder organization we obtain when we input the query CIS at the folder organization process:



Then we select the part we want to enlarge as following:

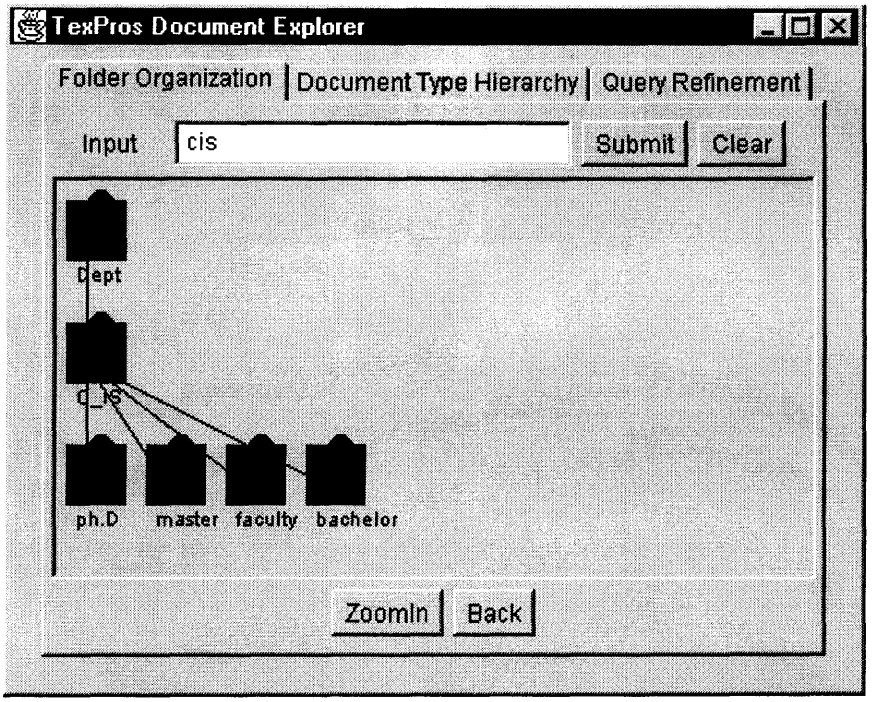


Then we select ZoomIn and we get as following:

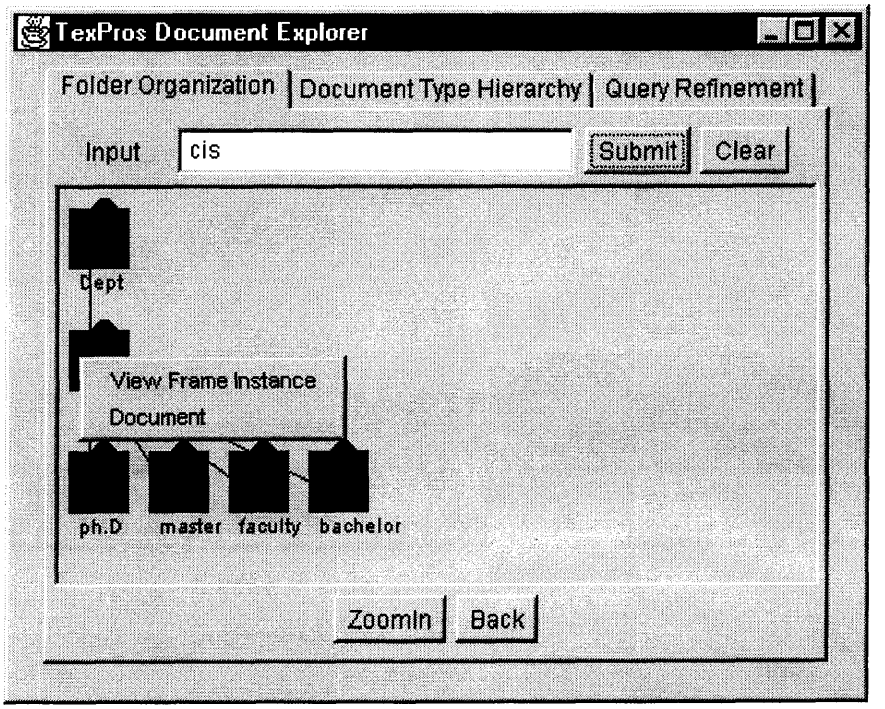


Of course we can select Back to restore the original graph.

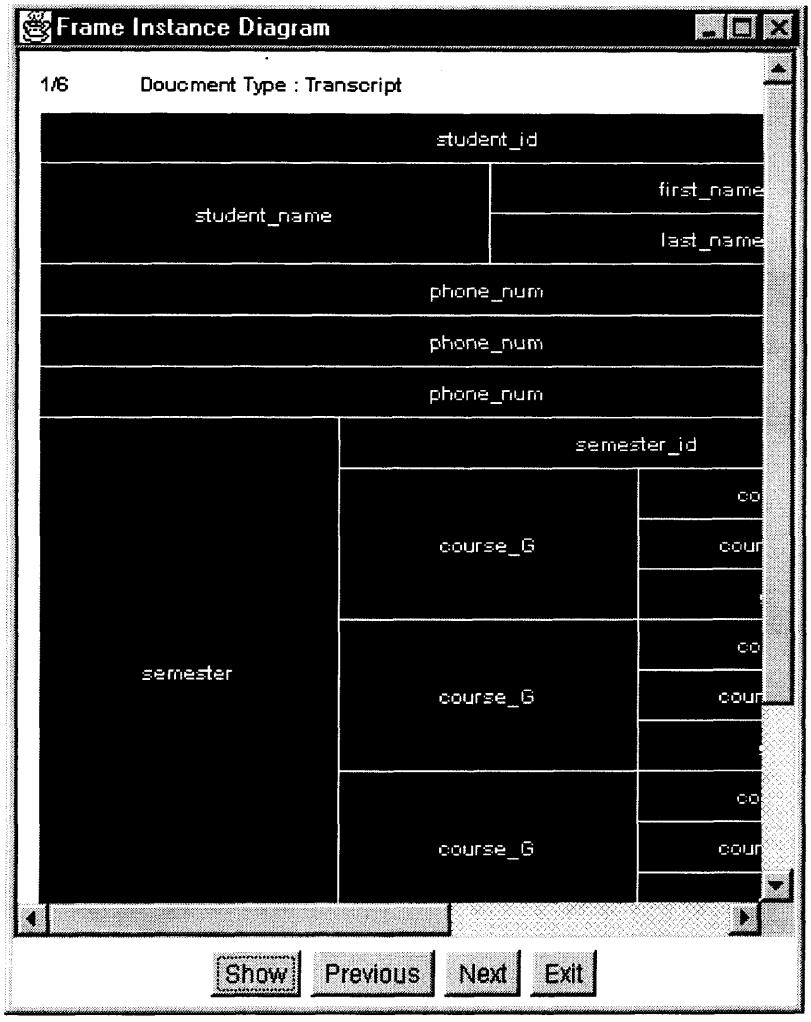
4. Display information at a node [39]: To let the users get as much information as they wish, we provide a function to display information at each node. The information available at a node includes the corresponding frame instances and original document. When the user clicks on a node, a small window is shown to let the user select between the following items: viewing corresponding frame instances, or viewing the original document. After the user makes his/her choice, the frame instances or the original document are displayed in a window. We give an example here. The following is the original folder organization we get when we input the query CIS at the folder organization process:



Then if we click on node CIS, we get the following graph:



Then if we select "view frame instance", we get the following graph:



And we can use buttons at the bottom of the graph and see all of the corresponding frame instances.

CHAPTER 6

EXPERIMENT RESULT

In this chapter, we present experimental results to test our Browser System. The experiments were performed to explore the impact of the Reusable Base on the efficiency of the retrieval algorithm.

6.1 Experiment Environment

The experiments were performed on a machine with a Pentium 166 processor with 32M RAM. We use Java and Oracle to implement the prototype. I will first discuss the folder organization and the document type hierarchy used in the experiments.

The following folder organization was used in the experiments:

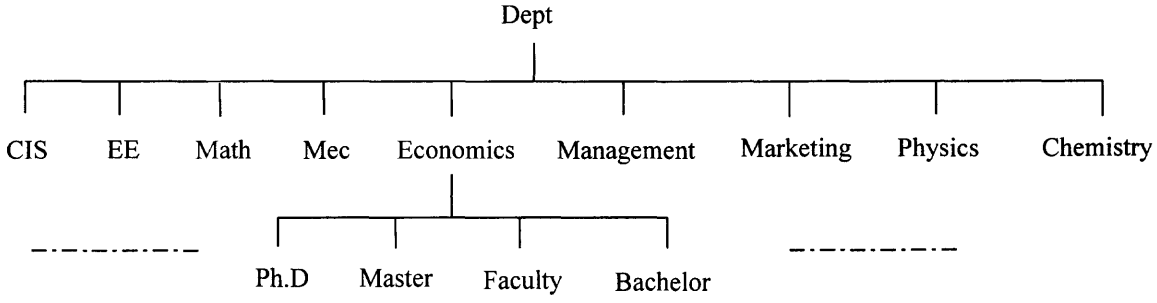


Figure 6.1 Folder Organization in the Experiments

From the Figure 6.1, we see that the folder Dept has the following folders as its children: CIS, EE, Math, Mec, Economics, Management, Marketing, Physics and Chemistry. And each of the folders (CIS, EE, Math, Mec, Economics, Management, Marketing, Physics and Chemistry) has four folders as its children: Ph.D, Master, Faculty and Bachelor.

The following document type hierarchy was used in the experiments:

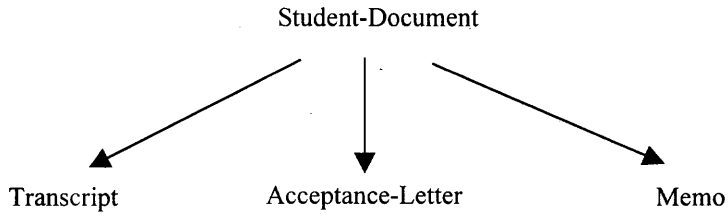


Figure 6.2 Document Type Hierarchy in the Experiments

From the Figure 6.2, we know that the document type Student-Document has three subdocument types: Transcript, Acceptance-Letter and Memo.

In the rest of this chapter, we will discuss the efficiency of the retrieval algorithm theoretically first. And then we will discuss the experiments with a limited System Catalog and a more extensive System Catalog. For each database, we will discuss the experiments using a group of fixed queries. We will discuss how the number of entries of the first and second level of the Reusable Base affects retrieval efficiency. In the experiments, we use system time to determine the amount of time taken for a particular operation to be completed. At the beginning of the retrieval process, we get the system time T1. At the end of the retrieval process, we get the system time T2. The difference between T2 and T1 is the time cost for a particular retrieval.

6.2 Theoretical Analysis

We will give an informal calculation of retrieval time through the following discussion. The symbols we use in the discussion are as following: the total number of simple query (without AND and OR operations between) involved in the retrieval process is n, the number of the entries in the first level of the Reusable Base is m1, the number of the entries in the second level of the Reusable Base is m2, the average time for accessing the simple query in the first level of the Reusable Base is T1, the average time for accessing

the simple query in the second level of the Reusable Base is T2, the average time for accessing the simple query in the System Catalog is T3, and the average time for other operations such as "swap in" and "swap out" is T4.

P1 -- the probability of finding a specific simple query in the first level of the Reusable Base is:

P1 = m1/n, when 0<=m1<=n;

P1= 1, when m1 > n.

The probability of a specific simple query in the second level of the Reusable Base is m2/n for m2<=n, and 1 for m2>n.

We consider the situation when the first level and the second level of the Reusable Base are filled:

Then if the input is a simple query, the average retrieval time T is:

$$T \sim (m1/n)*T1 + (1 - m1/n)*(m2/n)*T2 + (1 - (m1/n) - (1 - (m1/n)*(m2/n)))*T3+T4$$

To concentrate on analysis of retrieval time of a search in the first and the second level of the Reusable Base and the System Catalog, we ignore Tc. Then we get:

$$T \sim (m1/n)*T1 + (1 - m1/n)*(m2/n)*T2 + (1 - (m1/n) - (1 - (m1/n)*(m2/n)))*T3$$

We expect T1 << T2 << T3. If we can get the result from the first level of the Reusable Base, then the retrieval time will be short. That means if we can choose m1 = n, we can always get the result from the first level of the Reusable Base and the retrieval will be quick. Making m1 = n is acceptable when n is not too large and the system can distribute n memory slots to the first level of the Reusable Base. But, if n is large, it is not practical to choose m1 = n. (In most cases of use of TEXPROS, the number of the most likely queries L is usually much less than n. In this case, choose m1 = L is adequate to achieve

the minmum search). In our system, we give the user possibility to select m1. The user can adjust m1 according to his/her own needs. Hence, in many cases m1 can be selected to be equal to L, the number of most likely simple queries. In this case the system will nearly always complete a search in minimum time after the first and second level of the Reusable Base are filled.

The same analysis applies to the number of the entries in the second level of the Reusable Base. When we cannot get the result from the first level of the Reusable Base, it is better to get it from the second level of the Reusable Base than to get it from the System Catalog. We also give the user the ability to choose m2 according to his/her own needs.

Up to this point, we have discussed the case when the input is a simple query. If the input includes several simple queries, which are connected by AND or OR, we describe the query as compound. Assume K is the number of simple queries in the compound query, we consider two situations here:

1. If n >> m1, the total average retrieval time Ttotal is:

   Ttotal ~ K*T + (K-1)*Tandor

   Tandor is the time to perform the AND and OR operations between simple OP-Nets. From the experiments, we know Tandor << T. Hence, we may ignore Tandor and we get:

   Ttotal ~ K*T

2. If m1 → n, the probability for the first simple query in the first level of the Reusable Base is m1/n, the probability for the second simple query in the first level of the Reusable Base is (m1-1)/(n-1), …, and the probability for the Kth simple query in the first level of the Reusable Base is (m1-K+1)/(n-K+1). By similar arguments we can

establish the average search time for a simple query in the second level of the Reusable Base. We get all Ti for the ith simple query from the previous formular that is used to calculate T. And the total average retrieval time Ttotal is:

Ttotal ~ $\sum$Ti, i = 1 .. K

For the same reason as above, we ignore Tandor.

6.3 Experiment with Limited System Catalog

For the experiments with limited System Catalog, the data in the System Catalog contains only 1278 rows. We will discuss three cases for the limited System Catalog: 1) Fixed number of entries for the first level of the Reusable Base (for example 4) and fixed number of entries for the second level of the Reusable Base (for example 5); 2) Fixed number of entries for the second level of the Reusable Base (for example 5) and the number of entries for the first level of the Reusable Base varied from 0 to 6; 3) Fixed number of entries for the first level of the Reusable Base (for example 3) and the number of entries for the second level of the Reusable Base varied from 0 to 6. For the following experiment, we deal with a single department: the Math department.

1. **Fixed number of entries for the first and second level of the Reusable Base, 4 and 5 respectively.** We will discuss the time expended for the following queries: Math, Math^Master, Math^Ph.D, Math^Bachelor and Math^Faculty.

The representations for those strings in the following table are,

SC – System Catalog

FL – First Level of the Reusable Base

SL – Second Level of the Reusable Base

Table 6.1 Time Cost for Compound Query in Limited System Catalog

Time in Seconds / Query	SC	FL	SL	FL-SL	FL-SC	SL-SC
Math	43.2	0.1	5.1	N/A	N/A	N/A
Math^Master	72.5	0.2	11.8	10.8	36.3	40.0
Math^Ph.D	89.6	0.2	10.1	8.4	41.7	43.2
Math^Bachelor	43.7	0.2	6.0	4.8	8.6	12.2
Math^Faculty	30.9	0.2	12.1	10.8	29.1	29.9
Average	**56.0**	**0.2**	**9.0**	**8.7**	**28.9**	**31.4**

The first column in the table is the compound query. The second column is the time expended when the result is gotten from the System Catalog. The third column is the time expended when the result is gotten from the first level of the Reusable Base. The fourth column is the time expended when the result is gotten from the second level of the Reusable Base. The fifth column is the time consumed when the result is gotten from the combination of the first and the second level of the Reusable Base. The sixth column is the time expended when the result is gotten from the combination of the first level of the Reusable Base and the System Catalog. And the seventh column is the time expended when the result is gotten from the combination of the second level of the Reusable Base and the System Catalog.

From the table above, we see, without the Reusable Base, the average time cost for retrieval is 56.0. And with the Reusable Base, the time cost for retrieval in average is 22.4 (average of all columns for the bottom row). At the beginning of the retrieval, the Reusable Base is empty and results are gotten from searching the System Catalog.

Gradually the Reusable Base is populated and we do not need to go to the System Catalog to get the result because the results are found in the Reusable Base directly (we call it balance). When the balance is reached, the time cost for the retrieval is 6.0 (average of column FL, SL and FL-SL for the last row). Then we can say that the retrieval efficiency is at least two times faster with the Reusable Base than without the Reusable Base. When the balance is gotten, the retrieval efficiency is over nine times faster with the Reusabel Base than without the Reusable Base.

2. Fixed number of entries for the second level of the Reusable Base (for example 5) and the number of entries for the first level of the Reusable Base change from 0 to 6.

To test the previous theoretical analysis, we start testing when the first level and the second level of the Reusable Base are filled. To simplify testing, the selected inputs are all simply queries. The search queries are Math, Master, phD, Bachelor and Faculty.

In this experiment, the total number of the simple queries involved is 5. Through experiments, we find that the average time for accessing the first level of the Reusable Base T1 is 0.2 seconds, and the average time for accessing the second level of the Reusable Base T2 is 17.5 seconds. According to the theoretical analysis, we should get the following data for the different number of entries (assume it is m1) in the first level of the Reusable Base:

If m1 = 0, the retrieval time should be 0*T1 + 1*T2 = 17.5

If m1 = 1, the retrieval time should be (1/5)*T1 + (1 - 1/5)*T2 = 14.0

If m1 = 2, the retrieval time should be (2/5)*T1 + (1 - 2/5)*T2 = 10.6

If m1 = 3, the retrieval time should be (3/5)*T1 + (1 – 3/5)*T2 = 7.1

If m1 = 4, the retrieval time should be (4/5)*T1 + (1 – 4/5)*T2 = 3.7

If m1 = 5, the retrieval time should be (5/5)*T1 + (1 – 5/5)*T2 = 0.2

If m1 = 6, the retrieval time should be 1*T1 +(1 – 1)*T2 = 0.2

In the following table, the first column is the number of entries in the first level of the Reusable Base, the second column is the average time cost for the retrieval measured in the experiments, and the third column is the time cost from the theoretical analysis.

Table 6.2 Time Cost with Changing Number of Entries in the First Level in Limited System Catalog

No. of Entries in the Fisrt Level of the Reusable Base	**Time in Seconds**	**Theoretical Time in Seconds**
0	17.5	17.5
1	15.0	14.0
2	11.8	10.6
3	7.3	7.1
4	4.2	3.7
5	0.2	0.2
6	0.2	0.2

Comparing the data gotten from the theoretical analysis and the data gotten from the experiments, we see that each data point gotten from the experiments is the same or only slightly bigger than corresponding data obtained from the theoretical analysis. Considering the fact that we ignored the swap in, swap out and some other time in the

theoretical analysis, we can say data from two methods are match within tolerable error range.

From the table above, we get the following graph:

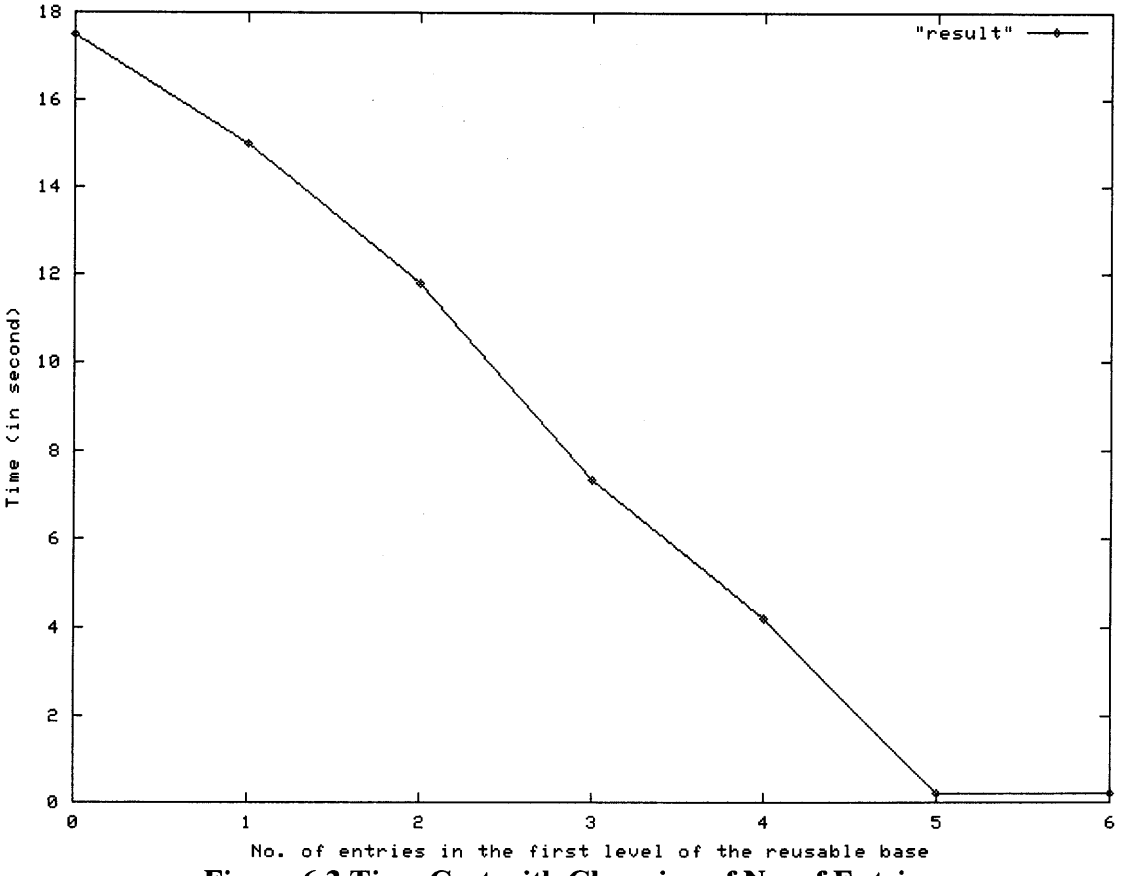


Figure 6.3 Time Cost with Changing of No. of Entries for the First Level of the Reusable Base

From Figure 6.3, we see that by incrementing the number of the entries in the first level of the Reusable Base, the time consumed for retrieval decreases significantly. When the number of entries in the first level of the Reusable Base is equal to the number of the simple queries (without AND and OR operations between) involved, the time cost becomes constant. Hence, we can say that the largest number of entries needed in the first level of the Reusable Base is the number of the simple queries involved in the retrieval.

We give the user an ability to choose the number of the entries in the first level of the Reusable Base to adjust his/her need.

3. **Fixed number of entries for the first level of the Reusable Base (for example 3) and the number of entries for the second level of the Reusable Base change from 0 to 6.**

To test the previous theoretical analysis, we start testing when the first level and the second level of the Reusable Base are filled. To simplify testing, the selected inputs are all simply queries. The same search queries are used: Math, Master, PhD, Bachelor and Faculty.

In this experiment, the total number of the simple queries involved is 5. Through experiments, we find that the average time for accessing the first level of the Reusable Base T1 is 0.2 seconds, the average time for accessing the second level of the Reusable Base T2 is 17.5 seconds, and the average time for accessing the System Catalog T3 is 50.5 seconds. According to the theoretical analysis, we should get the following data for the different number of entries (assume it is m1) in the second level of the Reusable Base:

If m1 = 0, the retrieval time should be (3/5)*T1 + 0*T2 + (1-3/5)*T3 = 20.3

If m1 = 1, the retrieval time should be (3/5)*T1 + (1 - 3/5)*(1/5)*T2 + (1-3/5-0.08)*T3 = 17.7

If m1 = 2, the retrieval time should be (3/5)*T1 + (1 - 3/5)*(2/5)*T2 +(1-3/5-0.16)*T3 = 15.0

If m1 = 3, the retrieval time should be (3/5)*T1 + (1 – 3/5)*(3/5)*T2 +(1-3/5-0.24)*T3 = 12.4

If m1 = 4, the retrieval time should be (3/5)*T1 + (1 – 3/5)*(4/5)*T2 +(1-3/5-0.32)*T3 = 9.8

If m1 = 5, the retrieval time should be (3/5)*T1 + (1 – 3/5)*(5/5)*T2 = 7.1

If m1 = 6, the retrieval time should be (3/5)*T1 +(1 – 3/5)*(5/5)*T2 = 7.1

In the following table, the first column is the number of entries in the second level of the Reusable Base, the second column is the average time cost for the retrieval obtained from the experiments, and the third column is time cost derived from the theoretical analysis.

Table 6.3 Time Cost with Changing Number of Entries in the Second Level in Limited System Catalog

No. of Entries in the Second Level of the Reusable Base	**Time in Seconds**	**Theoretical Time in Seconds**
0	20.3	20.3
1	18.5	17.7
2	15.5	15.0
3	12.5	12.4
4	10.5	9.8
5	7.2	7.1
6	7.2	7.1

Comparing the data from the theoretical analysis and the data from the experiments, each data point from the experiments is the same or slightly larger than the corresponding data point from the theoretical analysis. Considering we ignored swap in, swap out and some other time in the theoretical analysis, we can say that the theory explains the experimental observations with a tolerable margin of error.

From the table above, we get the following graph:

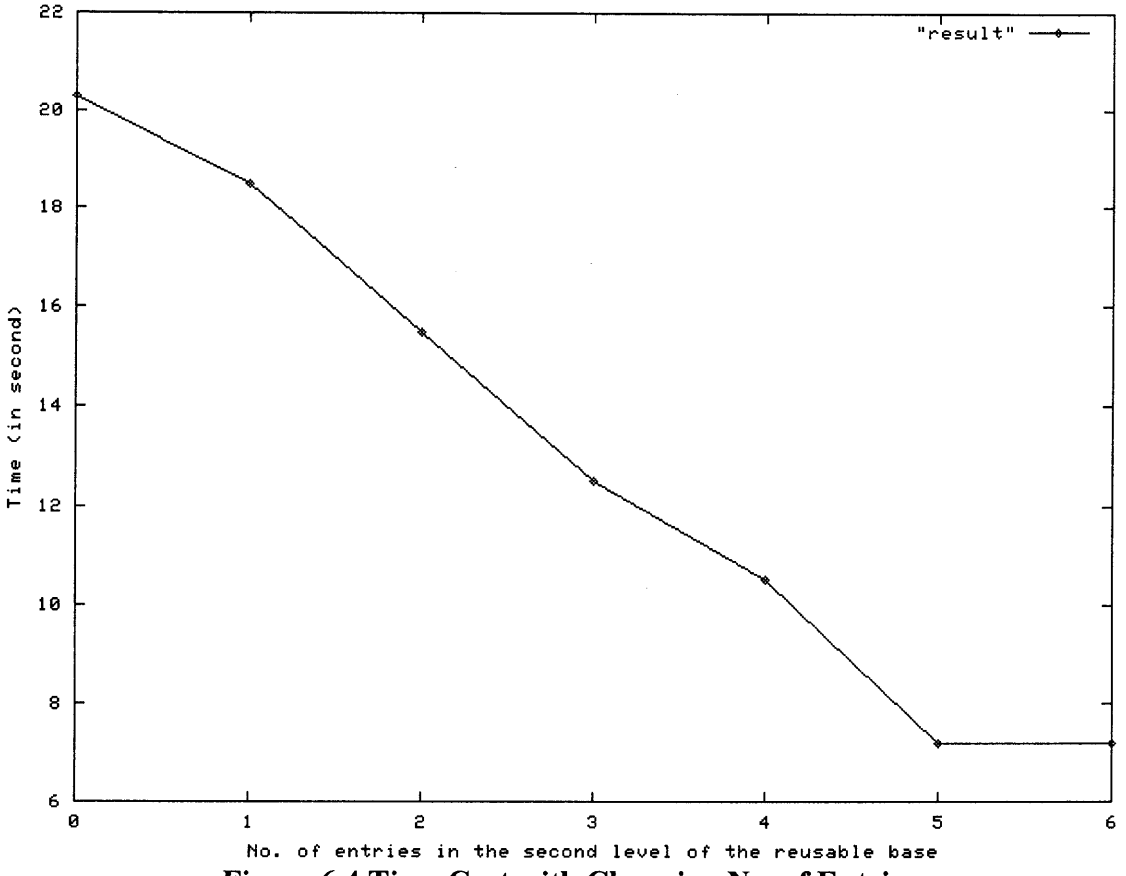


Figure 6.4 Time Cost with Changing No. of Entries in the Second Level of the Reusable Base

From Figure 6.4, we see that by incrementing the number of the entries in the second level of the Reusable Base, the time cost for retrieval decreases significantly. When the number of entries in the second level of the Reusable Base is equal to the number of the simple queries (without AND and OR) involved, the time cost for retrieval becomes constant. Then we can say the largest number of entries needed in the second level of the Reusable Base is the total number of simple queries involved.

6.4 Experiment with a More Extensive System Catalog

For the experiment with a more extensive System Catalog, the data in the System Catalog contains 6209 rows. As in the case of the limited System Catalog, we will discuss three cases for the more extensive System Catalog: 1. Fixed entries for the first level of the Reusable Base (for example 8) and fixed entries for the second level of the Reusable Base (for example 10); 2. Fixed number of entries for the second level of the Reusable Base (for example 8) and the number of entries for the first level of the Reusable Base varies from 0 to 9; 3. Fixed number of entries for the first level of the Reusable Base (for example 4) and the number of entries for the second level of the Reusable Base varies from 0 to 9. In the following experiments, as in the previous case, we choose only one department: the Math department.

1. **Fixed number of entries for the first and second level of the Reusable Base, 8 and 10 respectively.**

We will discuss the time expended for the following queries: Math, Memo, Math^Master, Math^Transcript, Acceptance_Letter, Math^Memo, PhD^Memo, Math^Bachelor^Transcript, Math^Faculty^Memo, and Math^Acceptance_Letter.

The representations of the strings used in the following table is,

SC – System Catalog

FL – First Level of the Reusable Base

SL – Second Level of the Reusable Base

Table 6.4 Time Cost with Compound Query in Extended System Catalog

Time in Seconds / Query	SC	FL	SL	FL-SL	FL-SC	SL-SC
Math	412.7	0.2	73.9	N/A	N/A	N/A
Memo	300.8	0.4	138.3	N/A	N/A	N/A
Math^Master	654.8	1.4	241.5	96.4	316.2	413.5
Math^Transcript	635.6	1.2	223.7	163.3	244.9	305.4
Acception_Letter	393.9	0.4	174.9	N/A	N/A	N/A
Math^Memo	582.4	1.6	204.8	149.0	209.9	285.6
PhD^Memo	743.6	3.2	298.4	148.6	453.8	508.4
Math^Bachelor^Transcript	680.9	1.6	270.5	203.7	354.6	420.4
Math^Faculty^Memo	633.6	1.9	392.1	186.5	281.3	467.7
Math^Acception_Letter	546.2	1.4	261.3	188.8	282.3	301.7
Average	**558.5**	**1.3**	**227.9**	**162.3**	**306.1**	**386.1**

The meaning of each column is similar as that in table 1. The first column in the table is query. The second column is the time cost when the result is fetched from the System Catalog. The third column is the retrieval time when the result is obtained from the first level of the Reusable Base. The fourth column is the retrieval time when the result is found at the second level of the Reusable Base. The fifth column is the time cost when the result is from a combination of the first and the second level of the Reusable Base. The sixth column is the time cost when the result is from the combination of the first level of the Reusable Base and the System Catalog. And the seventh column is the time cost when the result is from the combination of the second level of the Reusable Base and

the System Catalog. From table 4, we see, without the Reusable Base, the time cost for retrieval is 558.5. And with the Reusable Base, the time cost for retrieval in average is 273.7 (average of all columns for the bottom row). At the beginning of the retrieval process, the Reusable Base is empty and results are obtained by searching the System Catalog. Gradually the Reusable Base is populated and we do not need to go to the System Catalog to get the result because we can get the result from the Reusable Base directly (we call it balance). When the balance is achieved, the time cost for the retrieval is 130.5 (average of column FL, SL and FL-SL for the last row). Then we can say that the retrieval efficiency is at least two times faster with the Reusable Base than without the Reusable Base. When the balance is gotten, the retrieval time is reduced to one quarter of time with the Reusable Base than without the Reusable Base.

2. **Fixed number of entries for the second level of the Reusable Base (for example 8) and the number of entries for the first level of the Reusable Base varies from 0 to 9.** To test the previous theoretical analysis, we start testing when the first level and the second level of the Reusable Base are filled. To simplify testing, the selected inputs are all simply queries. The search queries are Math, Memo, Master, Transcript, Acceptance_Letter, PhD, Bachelor, and Faculty. In the following table, the first column is the number of entries in the first level of the Reusable Base and the second column is the average time cost for the retrieval from the experiments.

Table 6.5 Time Cost with Changing Number of Entries in the First Level in Extended System Catalog

No. Of Entries for the First Level of the Reusable Base	Time in Seconds
0	134.7
1	121.5
2	102.2
3	88.5
4	68.0
5	55.7
6	35.0
7	19.4
8	0.4
9	0.4

In this experiment, the total number of the simple queries involved is 8. Through experiments, we found the average time for accessing the first level of the Reusable Base T1 is 0.4 seconds, and the average time for accessing the second level of the Reusable Base T2 is 134.7 seconds, and the average time for accessing the System Catalog T3 is 273.2 seconds. If we do the same theoretical analysis as for the limited System Catalog, we can get the same solution: each data point in the two groups is the same within a tolerate error range.

From the table above, we get the following graph:

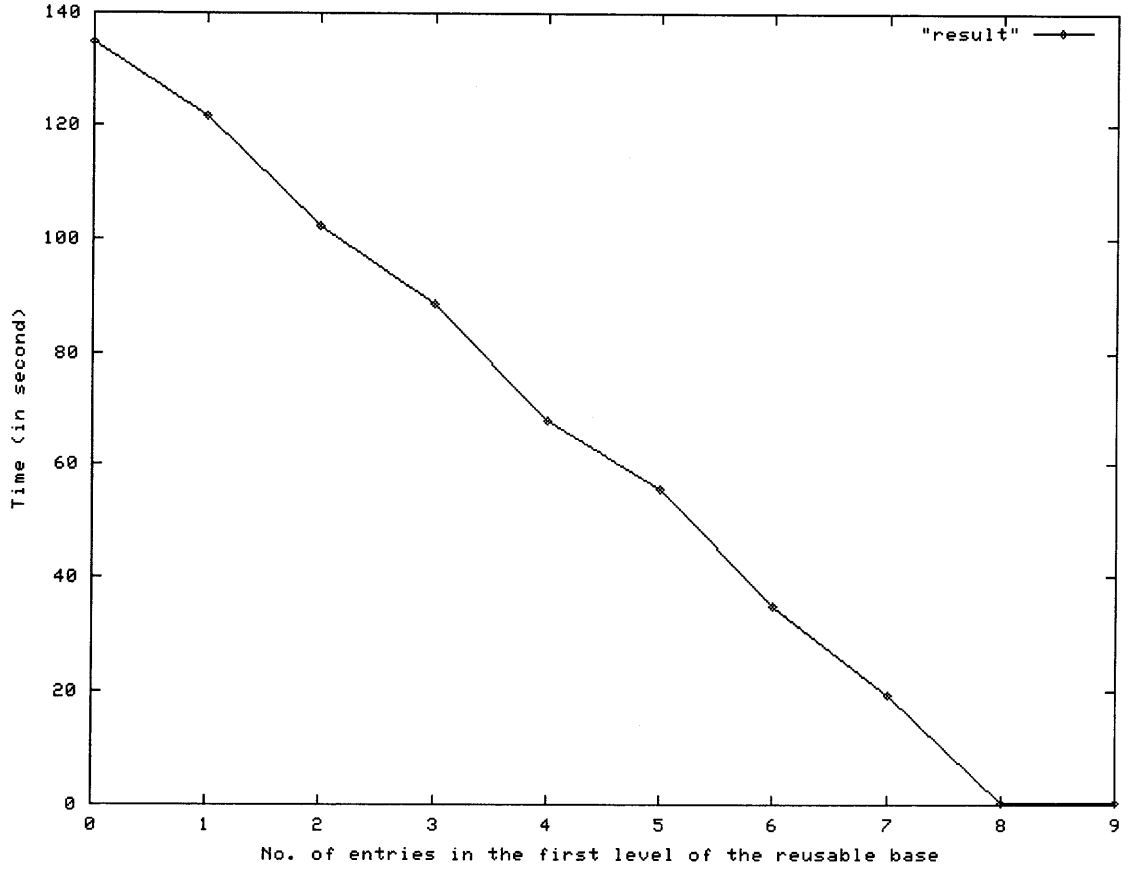


Figure 6.5 Time Cost with Changing No. of Entries in the first level of the Reusable Base

From Figure 6.5, we see that as the number of the entries in the first level of the Reusable Base is incremented, the retrieval time decreases significantly. When the number of entries in the first level of the Reusable Base is equal to the number of the simple queries (without AND and OR operations between) involved, the time cost becomes constant. Hence, we can say the largest number of entries in the first level of the Reusable Base needed is the number of the simple queries involved in the retrieval. As mentioned for the

limited System Catalog, our system gives the user the ability to adjust the number of entries in the second level of the Reusable Base.

3. Fixed number of entries for the first level of the Reusable Base (for example 4) and the number of entries for the second level of the Reusable Base change from 0 to 9. To test the previous theorical analysis, we start testing when the first level and the second level of the Reusable Base are filled. To simplify testing, the selected inputs are all simply queries. The search queries are Math, Memo, Master, Transcript, Acceptance_Letter, PhD, Bachelor, and Faculty. In the following table, the first column is the number of entries in the second level of the Reusable Base and the second column is the average retrieval time obstained from the experiments.

Table 6.6 Time Cost with Changing Number of Entries in the Second Level in Extended System Catalog

No. of Entries for the Second Level of the Reusable Base	Time in Seconds
0	136.8
1	129.5
2	119.1
3	111.9
4	102.2
5	94.6
6	83.7
7	76.5
8	67.7
9	67.7

In this experiment, the total number of the simple queries involved is 8. Through experiments, we choose the average time for accessing the first level of the Reusable Base T1 is 0.4 seconds, the average time for accessing the second level of the Reusable Base T2 is 134.7 seconds, and the average time for accessing the System Catalog T3 is 273.2 seconds. If we do the same theoretical analysis as for the limited System Catalog, we can get the same solution: each data point in two groups is the same within a very small range of error.

From the table above, we get the following graph:

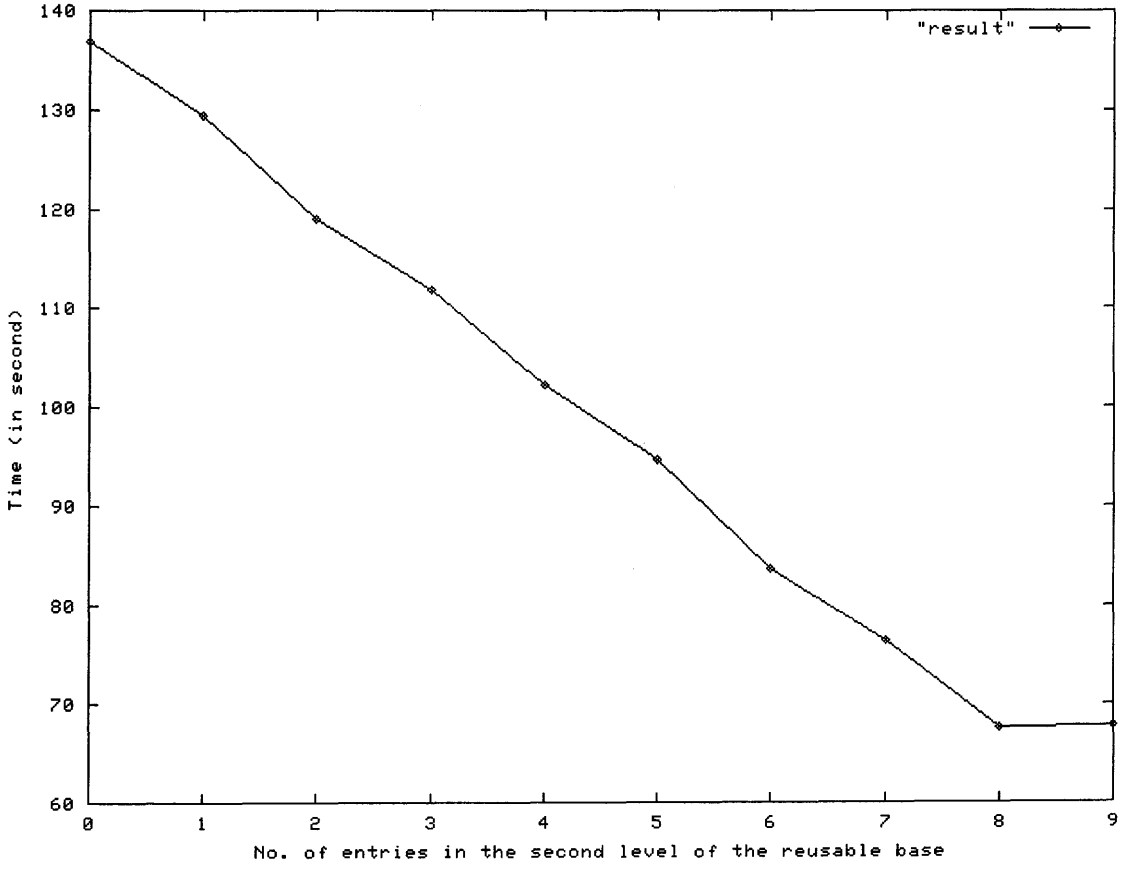


Figure 6.6 Time Cost with Changing of No. of Entries in the Second Level of the Reusable Base

From Figure 6.6, we see that as the number of the entries in the second level of the Reusable Base is incremented, the time cost for retrieval decreases until the number of entries in the second level of the Reusable Base is equal to the number of the simple queries (without AND and OR operations between) involved, then the retrieval time becomes constant. Hence we can say that the largest number of entries needed in the second level of the Reusable Base is the number of the simple queries involved.

6.5 Conclusion

From the experiments above, we draw the following conclusions:

1. Retrieval is at least two times more efficient with the Reusable Base than without it. If the number of the entries for the first and the second level of the Reusable Base is large enough, we do not need to go to the System Catalog to get the result after the first time since we can get the result from the Reusable Base directly. In that case, the retrieval will be an order of magnitude faster than a search that does not find the simple queries in the first level.
2. As the number of entries for the first level of the Reusable Base is incremented, the retrieval time decreases significantly until the number of entries in the first level of the Reusable Base is equal to the number of the simple queries (without AND and OR operations between) involved. Hence we can say that the largest number of entries needed in the first level of the Reusable Base is the number of simple queries involved. Typically, the number of the most likely queries L is much smaller than total number of possible simple queries involved. Hence in most cases, the users should choose the number of entries in the first level of the Reusable Base to be L,

the most likely number of queries used by that user. Our system gives the user an opportunity to change the number of the entries in the first level of the Reusable Base. For example we set the original number of entries in the first level of the Reusable Base to 5 and the user wants faster retrieval, he/she can change the number of the entries in the first level to 8 or whatever.

3. As the number of entries for the second level of the Reusable Base is incremented, the retrieval time decreases significantly until the number of entries for the second level of the Reusable Base is equal to the number of the simple queries (without AND and OR between) involved. Hence we can say that the largest number of entries needed in the second level of the Reusable Base is the number of simple queries involved. As in the case of the first level of the Reusable Base, our system gives an opportunity for the user to change the number of the entries in the second level of the Reusable Base. For example we set the original number of entries in the second level of the Reusable Base to 15, and the number of the simple queries the user is interested in is bigger than 15, and he/she wants the second level of the Reusable Base to hold more than 15, he/she can change the number of the entries in the second level to 20 or whatever.
4. Our system gets shorter retrieval time by making use of more spaces including memory and hard disk space. As mentioned earlier, the user is given an ability to adjust the number of entries in the first level of the Reusable Base. As long as the system has enough memory, the user can increase the number of entries in the first level of the Reusable Base. For the number of entries in the second level of the Reusable Base, the user is still given an ability to adjust it. But if the second level of

the Reusable Base becomes too large, it is useless to use the Reusable Base since the access time becomes too long and approaches the time to access the System Catalog. Hence, finding the optimal number of entries for the first and second level of the Reusable Base is very important.

From the experiments, we can find that the browser we discussed in this dissertation is very fit in the library environment. The System Catalog holds all materials in variety fields. Every user concerns only a small part of it with one (mostly) or two aspects and that will be stored in this user's Reusable Base. When the user wants to search, his/her Reusable Base will be searched first. When the Reusable Base is established, most of the time, the user can get the result directly from the Reusable Base. From the experiments, we already see how fast for retrieval with the Reusable Base than without it.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this dissertation, we provide an expanded range of browser tools to improve a user's ability to retrieve a desired document and an efficient algorithm for document retrieval. First we expand the interface functions. The new functions (such as zoom in and zoom out, i.e. automatic scaling of a portion of a graph that is of interest to a user) are provided to make it easier for the user to look at an especially large graph. Help functions are also provided to give the user help during the retrieval process. Two types of help functions (i.e. terminological help and strategic help) are provided. These functions give users an easier way to view a large graph in one window, to obtain a clearer picture of a particular part, and to get help through the retrieval process. In future research, we will consider the use of a ranking mechanism. We will consider classifying retrieved information into categories and let the most relevant or most useful information or document to be viewed first. This will save time from the user to find the exact information.

Second we provide an algorithm that makes retrieval more efficient by using a Reusable Base. The Reusable Base is used to hold information that is most related to the user previous desires and the information stored in the Reusable Base is more easily used to form the OP-Net than that in the System Catalog. We do not need to go to the System Catalog to find a result if we can find it in the Reusable Base. This significantly speeds up the retrieval process. Currently, when the new search topic enters the system, we simply store it in the Reusable Base. In the future work, we will consider rules for the

inclusion of a component in the Reusable Base. If the component satisfies these conditions, it will be stored in the Reusable Base. Otherwise, the new search topic may just be ignored. Currently, we give users an opportunity to adjust the number of entries in the first and second level of the Reusable Base. In the future, we will consider helping users to adjust number of entries automatically according to the performance of the system. If the retrieval time is too long, the system will delete or insert new space for entries.

Third, we deal with a wider variety of situations compared to the previous browser. Information about the Folder Organization and the Document Type Hierarchy are provided in addition to the information of OP-Net. Hence, if users know what kind of documents is desired, or which folder they are interested in, they can go to the particular document type or the particular folder directly.

Currently, Boolean expressions are used as input. In the future, we will consider using natural language as input in ways that benefit novice users. To accomplish this, we must transfer the natural language input into several Boolean expressions and do search using the Boolean expressions. Currently, we use a simplified version of a thesaurus that only considers synonyms. In the future research, we intend to extend the thesaurus to include others relations: NT—narrow term, BT—broader term, RT—related term, and so on. And future research will also focus on the document-based information retrieval instead of document retrieval. We will work on information mining in the original document when result cannot be found at the System Catalog.

APPENDIX A

STORAGE LAYER – KNOWLEDGE BASE

As discussed in great detail in Reference [46], TEXPROS is an intelligent knowledge-based document processing system. It employs knowledge-based techniques and methods to provide the system with automatic document classification and extraction, intelligent document filing, and cooperative document browsing. Hence, knowledge management becomes an essential part of TEXPROS. Knowledge management deals with ways of representing knowledge and how it is organized. We focus on the following concerns:

- Developing representation and organization of knowledge adequate to improve system performance and maintain system consistency.
- Developing the representation and organization of the knowledge to fit in general working domain.

The storage layer includes the System Catalog, the Frame Instance Base, a Thesaurus and the Original Document Base. These are the core parts of TEXPROS. They support all other parts of TEXPROS: classification, extraction, filing and browsing. A powerful knowledge base can strengthen the capability of TEXPROS.

TEXPROS adopts dual models: document type hierarchy, which is used to classify documents, and folder organization, which represents the user's real-world document filing system. The System Catalog represents both of these models. It is used to reflect the actual meta-data of the document type hierarchy and the folder organization. It makes it possible for the system to resolve the ambiguity in a user's query and obtain the needed information.

Whenever a document comes into the system, it is stored in the Original Document Base. The document identifier is used to record the document's address on the physical storage device. The Original Document Base gives users a chance to access original documents. However, accessing to the System Catalog can often satisfy most needs. Hence, it is not necessary to access the document base to browse and retrieve a document most time. In cases where we need to access the document base, the document identifier can be used as an index for retrieval. Using the document identifier to access to the document base improves the search performance.

The frame instances are stored in the Frame Instance Base. Users can find frame instances efficiently through the Frame Instance Base, and often do not need to access the Original Document Base.

The Thesaurus is composed of a collection of items and the relationships between them. The technology of thesaurus is widely used in the document retrieval to improve system performance.

From the view of the browser subsystem, the storage layer provided by TEXPROS is organized as figure A.1.

In the remainder of this chapter, we discuss each part of the storage layer separately.

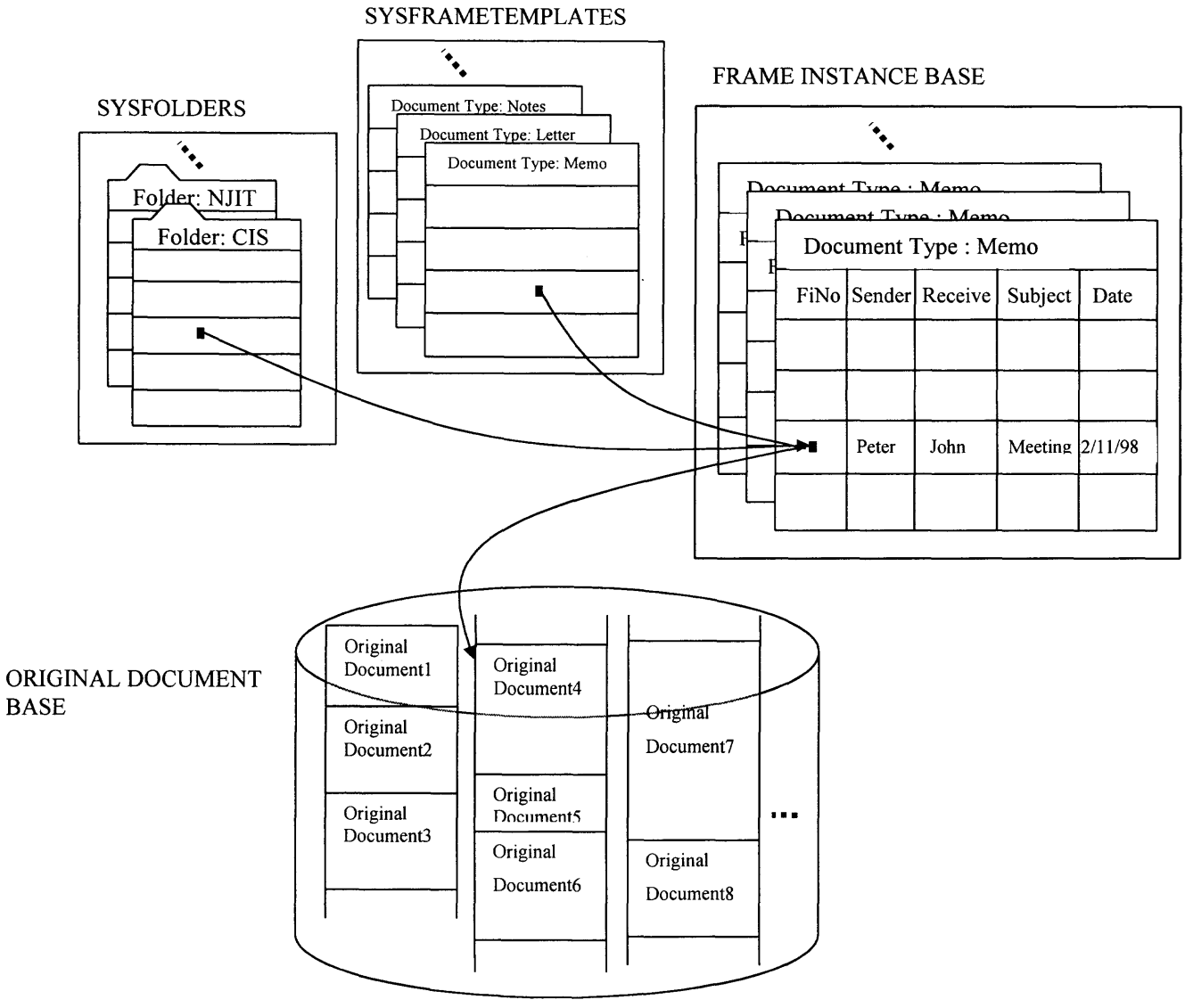


Fig A.1 The Storage System

A.1 System Catalog

The TEXPROS System Catalog, which is an important part of the knowledge base, is used to describe the meta-data of the folder organization and the document type hierarchy. The folder organization and the document type hierarchy are combined to form the dual model of TEXPROS. In TEXPROS, the concepts of frame templates and frame

instances are used both at the operational level and the system level. At the operational level, the concept of a frame template is used to establish the document type hierarchy; and the concept of frame instances is used to describe the key information of a particular document according to its document type, which is defined by a frame template. At the system level, the concept of frame template is used to classify the information stored in the System Catalog; and the concept of the frame instance is used to contain the information about dual model – folder organization and document type hierarchy. This approach, which represents operational knowledge and system knowledge consistently, allows the System Catalog to be stored directly into the storage base in the same way as the frame instances are stored into the frame instance base.

In TEXPROS the System Catalog can be viewed as a collection of sets of frame instances whose types are the corresponding frame templates. In the System Catalog, those sets of frame instances are used to store the information of dual models – folder organization and document type hierarchy. The following is the structure of the System Catalog:

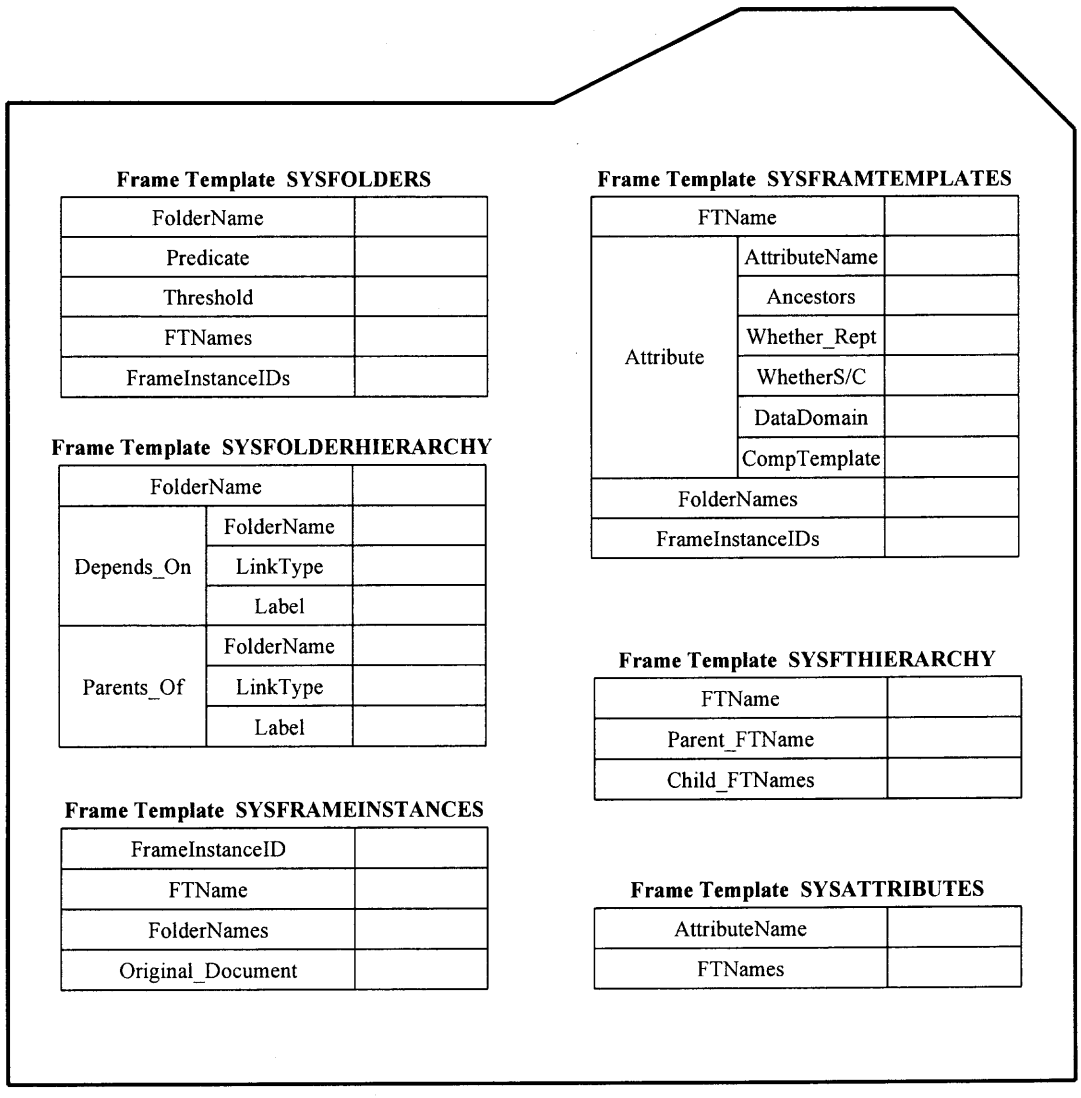


Figure A.2 System Catalog Structure

A set of SYSFOLDERS frame instances describes the information contained in each folder. It includes the following information: the name of the folder, the predicate defined in the folder identifier, the threshold, the name of frame template that describes the type of the frame instances of the folder, and frame instances ID number that is specified in terms of attributes of the corresponding frame template and is repeatable.

A set of SYSFOLDERHIERARCHY frame instances describes the architecture of folder organization. It contains the following information: the name of the folder, the repeatable and composite attributes Depends_On that describes the parents folder of the current folder, and repeatable and composite attribute Parents_Of that describes the children folder of the current folder. Through frame template SYSFOLDERHIERARCHY, the folder organization is established.

A set of SYSFRAMEINSTANCES frame instances describes the information of every frame instance. It includes the following information: the frame instance ID number, the name of the frame template that describes the type of the current frame instance, the name of the folder that includes the current frame instance, and the ID of original document from which the current frame instance is extracted.

A set of SYSFRAMETEMPLATES frame instances describes the information of every frame template. It contains the following information: the name of the frame template, the attributes that are used to define the frame template, the name of the folder that describes the information of the folder corresponding to the current frame template, and the frame instance ID number that describes the frame instances whose type is the current frame template.

A set of SYSFTHIERARCHY frame instances describes the architecture of the document type hierarchy. It includes the following information: the name of the frame template, the attribute Parent_FTName that describes the name of the parent frame template of the current frame template, and the attribute Child_FTName that describes the name of the child frame template of the current frame template. Through SYSFTHIERARCHY, we see that the document type hierarchy is established as a tree structure.

A set of SYSATTRIBUTES frame instance describes the information contained in the attributes. It includes the following information: the name of the attribute, and the name of the frame template to which the current attribute belongs.

From what was described above, we can see that the System Catalog represents the information of dual model of TEXPROS – the folder organization and the document type hierarchy. It includes the information about every folder and its corresponding frame instance, every frame template and its corresponding attributes, and, of course, architecture of the folder organization and the document type hierarchy. It establishes the basis for the later retrieval and browsing processes. It also makes retrieval more efficient.

A.2 Frame Template Base

In TEXPROS the concepts of frame instances and frame templates are used both at the operational level and system level. TEXPROS unifies the approach of handling operational knowledge and system knowledge. We next discuss the construction of the frame template base at the storage level.

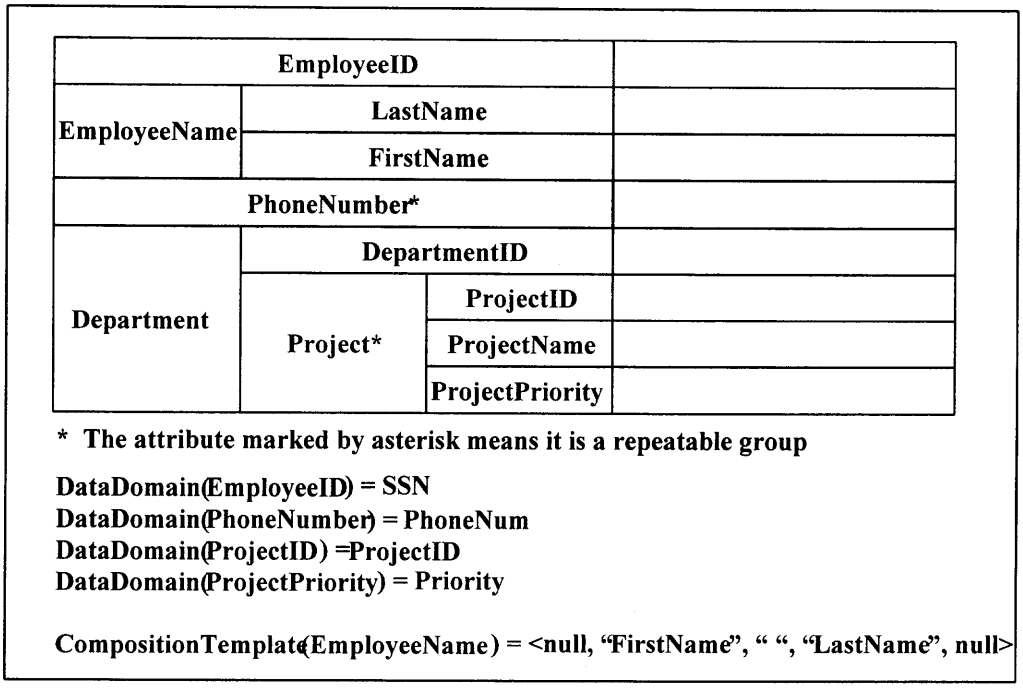
We adopt a model for the frame template base based on relational databases. However, it extents the capability of the traditional relational database. Because a frame template is composed of a set of attributes and those attributes describe the characteristics of one kind of documents, it seems natural to construct a frame template base as a relational database. But relational databases are limited in their representation. First, a traditional relational database does not support the concept of composite attributes. Secondly, a traditional relational database does not support the concept of multi-valued attributes. Thirdly, a traditional database does not support the concept of Data Domain and

Composition Template. In addition, we need to keep the sequences of the attributes, which makes identifying the position of the attribute in the frame template easier.

Based on the consideration above, we can not map a frame template directly to a table of a relational database. Hence, it is necessary to extend the definition of a traditional relational database by defining the schema for the frame template base as following:

(FT_Name, ATT_Name, Ancestors, WhetherRep, WhetherS/C, Seq, DataDomain, CompTemplate)

This schema contains the complete knowledge of the frame template. According to this schema, each record under the same frame template is an attribute of the frame template. In the schema of the frame template, FT_Name indicates the name of the frame template; Att_Name means the corresponding name of the attribute in the frame template; Ancestors defines which composite attribute block the attribute is in; WhetherRep is a Boolean value that shows whether the attribute is repeatable or not; WhetherS/C is a Boolean value that is used to indicate whether the attribute is a simple one or composite one; Seq shows the position of the attribute in the corresponding composite attribute block; DataDomain defines the domain for the attribute; and CompTemplate stores the user-defined composition template for that composite attribute. The following is an example of frame template and its representation in the frame template base. In the example, the frame template "transcript" has a composite attribute "Student Name" and multi-valued attribute "PhoneNumber". From the way a frame template is represented in the frame template base, we can see how important it is to keep the attributes in order.



EmployeeID			
EmployeeName	LastName		
	FirstName		
PhoneNumber*			
Department	DepartmentID		
	Project*	ProjectID	
		ProjectName	
		ProjectPriority	

* The attribute marked by asterisk means it is a repeatable group

DataDomain(EmployeeID) = SSN
DataDomain(PhoneNumber) = PhoneNum
DataDomain(ProjectID) =ProjectID
DataDomain(ProjectPriority) = Priority

CompositionTemplate(EmployeeName) = <null, "FirstName", " ", "LastName", null>

Figure A.3 Frame Template "EmployeeRecord"

FT_Name	Att_Name	Ancestors	WhetherRep	WhetherS/C	Seq	DataDomain	CompTemplate
EmployeeRecord	EmployeeID		N	Y	1	SSN	
EmployeeRecord	EmployeeName		N	N	2		<null, Firstname, LastName, null>
EmployeeRecord	LastName	EmployeeName	N	Y	1		
EmployeeRecord	FirstName	EmployeeName	N	Y	2		
EmployeeRecord	PhoneNumber		Y	Y	3	PhoneNum	
EmployeeRecord	Department		Y	N	4		
EmployeeRecord	DepartmentID	Department	N	Y	1	ProjectID	
EmployeeRecord	Project	Department	Y	N	2		
EmployeeRecord	ProjectID	Department###Project	N	Y	1		
EmployeeRecord	ProjectName	Department###Project	N	Y	2		
EmployeeRecord	ProjectPriority	Department###Project	N	Y	3	Priority	

Figure A.4 Frame Template "EmployeeRecord" in Frame Template Base

A.3 Frame Instance Base

After the document's classification and extraction, a frame instance of the document is formed. All frame instances are kept in the Frame Instance Base. Every frame instance has a unique ID – FI_ID that is used to distinguish it from others. The frame instances are

organized according to their corresponding document type, which means that all frame instances of the same type will be clustered. We create two types of frame template – SYSFOLDER and SYSFRAMETEMPLATE, to connect frame instances to their corresponding folders and frame templates separately. Through SYSFOLDER, we can find the frame instances in the same folder. And through SYSFRAMETEMPLATE, we can find corresponding frame instances with the same document type. The following is the sample of the frame instance of the type "Transcript" that we discussed in a previous paragraph.

ATTRIBUTE			VALUE
EmployeeID			777987777
EmployeeName	LastName		Hu
	FirstName		Jason
PhoneNumber			973-5961111
PhoneNumber			9735961112
PhoneNumber			9735961113
Department	DepartmentID		Engineering
	Project	ProjectID	ED-S-01
		ProjectName	Management System
		ProjectPriority	3
	Project	ProjectID	ED-C-09
		ProjectName	Global Control System
		ProjectPriority	6
	Project	ProjectID	ED-F-20
		ProjectName	Y2K Compatability
		ProjectPriority	9
Department	DepartmentID		Service and Support
	Project	ProjectID	SSD-C-01
		ProjectName	Electronic Tranasction System
		ProjectPriority	6

Figure A.5 A Sample Frame Instance of Type "Transcript"

Based on the similar consideration used to form a relational database for the frame template base, to represent all frame instances, we define the schema for frame instance base as follows:

(FI_ID, ATT_Name, Ancestors, Orig_Value, Key_Value, Seq, End_Seq)

In the schema of frame instance base, FI_ID indicates the ID number for the frame instance, which is unique; ATT_Name indicates the name of the attribute; Ancestor

indicates the composite attribute block to which the attribute belongs; Orig_Value indicates the value extracted directly from the document; Key_Value indicates the internal representation for the original value; And Seq and End_Seq are used to store the value sequence and the value block information.

From the above schema for the Frame Instance Base, we see besides the original value extracted from the document, the frame instance keeps the key value that is the internal representation of the original value. This later can speed up the retrieval process. Because in the retrieval system first change the query into a form that consists of only key terms, this will narrow the range of the search. The following is the sample of the frame instance "Transcript" in frame instance base.

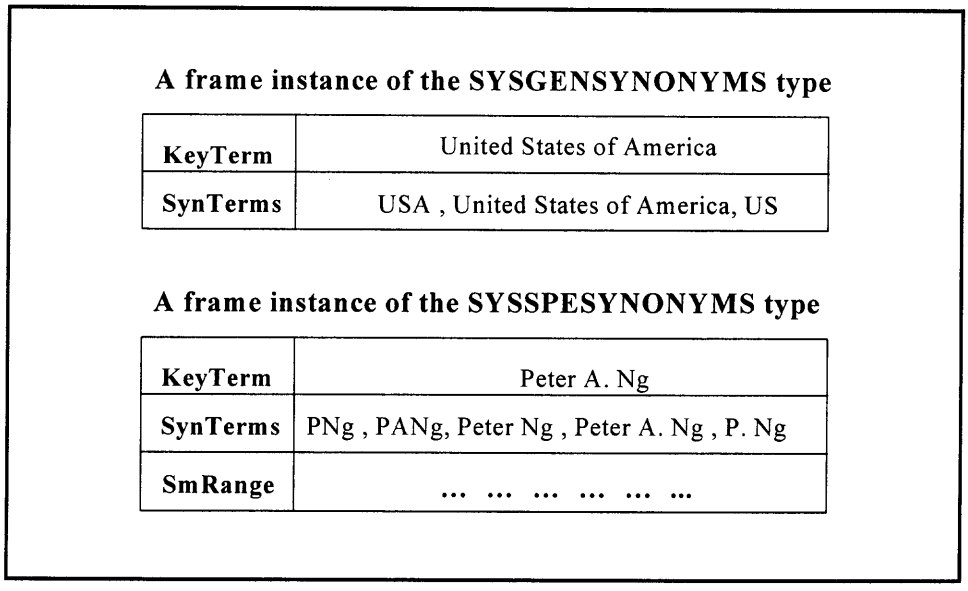
FI_ID	Att_Name	Ancestor	Orig_Value	Key_Value	Seq	End_Seq
10001	EmployeeID		77798777	777-98-7777	1	-1
10001	EmployeeName		Jason Hu	Jianshun Hu	2	3
10001	LastName	EmployeeName	Hu	Hu	2	-1
10001	FirstName	EmployeeName	Jason	Jason	3	-1
10001	PhoneNumber		973-5961111	973-5961111	4	-1
10001	PhoneNumber		9735961112	973-5961112	5	-1
10001	PhoneNumber		9735961113	973-5961113	6	-1
10001	Department				7	16
10001	DepartmentID	Department	Engineering	Engineering	7	-1
10001	Project	Department			8	10
10001	ProjectID	Department##Project	ED-S-01	ED-S-01	8	-1
10001	ProjectName	Department##Project	Management System	Management System	9	-1
10001	ProjectPriority	Department##Project	3	3	10	-1
10001	Project	Department			11	13
10001	ProjectID	Department##Project	ED-C-09	ED-C-09	11	-1
10001	ProjectName	Department##Project	Global Control System	Global Control System	12	-1
10001	ProjectPriority	Department##Project	6	6	13	-1
10001	Project	Department			14	16
10001	ProjectID	Department##Project	ED-F-20	ED-F-20	14	-1
10001	ProjectName	Department##Project	Y2K Compatability	Y2K Compatability	15	-1
10001	ProjectPriority	Department##Project	9	9	16	-1
10001	Department				17	20
10001	DepartmentID	Department	Service and Support	Service and Support	17	-1
10001	Project	Department			18	20
10001	ProjectID	Department##Project	SSD-C-01	SSD-C-01	18	-1
10001	ProjectName	Department##Project	Electronic Transaction System	Electronic Transaction System	19	-1
10001	ProjectPriority	Department##Project	6	6	20	-1

Figure A.6 A Sample "EmployeeRecord" Frame Instance in Frame Instance Base

A.4 Thesaurus

Thesaurus technology is widely used in information retrieval to improve system performance. The basic idea of the thesaurus is to set one key term to represent a group of synonyms. Later this "key term" is used internally by the system in the retrieval process. These key terms act as a filter directed towards the choice, or descriptors of contextual documents that are believed to be useful in the later retrieval phase. Based on the fact that

there are two kinds of synonyms – one for the general application domain (for example USA represents United States of America everywhere) and the other for special application domain (for example RU represents Rutgers—State University of New Jersey, CS represents department of computer science, etc.), two-level thesaurus model is adopted in TEXPROS. The following is the example of the frame instances for the Thesaurus model.



A frame instance of the SYSGENSYNONYMS type

KeyTerm	United States of America
SynTerms	USA , United States of America, US

A frame instance of the SYSSPESYNONYMS type

KeyTerm	Peter A. Ng
SynTerms	PNg , PANg, Peter Ng , Peter A. Ng , P. Ng
SmRange	... ...

Figure A.7 Examples of Frame Instances for Thesaurus Model.

From Figure A.7, we see a two level thesaurus model: The first level is for the general domain and the second level is for the special domain. The set of frame instances of type SYSGENSYNONYMS describes the synonyms in the general domain; and the set of frame instances of type SYSSPESYNONYMS describes the synonyms in the specific domain and where the attribute SmRange defines the specific working domain. The first level of the Thesaurus in the general domain is created before the system is delivered to the users because it includes a set of both common and unique terms. The second level of the Thesaurus in specific working domain involves different knowledge in the different

user's application domain. It is created by the user according to his/her knowledge of the domain. Hence, the user may add, update, and delete the terms from the second level Thesaurus. To avoid the possible conflict between the two level of the Thesaurus – same words occurs in both the first level and the second level because of the user's action to the second level of the Thesaurus, the system gives higher priority to the second level. Therefore, when searching the thesaurus, the second level for the specific domain is always checked first. If the key term can not be found, the first level for the general domain will be checked.

In the browser subsystem of TEXPROS, we use the Thesaurus in the topic parsing. Hence, the Thesaurus helps to find the key terms in the raw topic that were inputted by the user. It allows the retrieval process to be conducted more efficiently.

APPENDIX B

SOURCE CODE

Source code of the prototype is attached.

REFERENCES

1. Augusto Celentano, Maria Grazia Fugini, and Silvano Pozzi, "Knowledge-Based Document Retrieval in Office Environments: The Kabiria System", *ACM Transactions on Information Systems*, Vol. 13, No. 3, pp. 237-268, July 1995.

2. Bookstein, "A. Probability and Fuzzy-Set Applications to Information Retrieval", *Ann. Rev. Inf. Sci. Tech.* 20 (1985), pp.117 – 151.

3. Wong, S. K. M., Ziarko, W., Raghavan, V. V., and Wong, "P.C.N. On Extending the Vector Space Model for Boolean Query Processing", *Processing of the ACM Conference on Research and Development in Information Retrieval* (Pisa, 1986). F. Rabitti. Ed., pp.175 – 185.

4. D. V. Rama and Padmini Srinivasan, "An Investigation of Content Representation Using Text Grammars", *ACM Transactions on Information Systems*, Volume 11, No. 1, Pages 51-75, Jan. 1993.

5. Dario Lucarella and Antonella Zanzi, "A Visual Retrieval Environment for Hypermedia Information Systems", *ACM Transactions on Information Systems*, Vol. 14, No. 1, pp. 3-29, Jan. 1996.

6. Alistair Moffat and Justin Zobel, "Self-Indexing Inverted Files for Fast Text Retrieval", *ACM Transactions on Information Systems*, Vol. 14, No. 4, pp. 349-379, Oct. 1996.

7. Bertino, E., Negri, M., Pelagatti, G., and Sbattella, L., "Object-Oriented Query Languages: The Notion and the Issues", *IEEE Data Knowl. Eng*, 4, 3, pp. 223-237, 1992.

8. Straube, D. D. and Ozsu, M. T., "Queries and Query Processing in Object-Oriented Database Systems", *ACM Transactions on Information System*, Vol. 8, No. 4, pp. 387-430, 1990.

9. Gordon, M., "Probabilistic and Genetic Algorithms in Document Retrieval", *Commun. ACM* 31, 10, 1988.

10. Watters, C R., "Logic Framework for Information Retrieval", *J. Am. Soc. Inf. Sci* 40, 5, 1989.

11. Salton, G., *Automatic Text Processing*, Addison-Wesley, Reading, Mass., 1989.

12. Jide B Odubiyi, David J Kocur and Stuart M Weinstein, *Saire - A Scalable Agent-Based Information Retrieval Engine*, 1997.

13. Chanda Dharap and Martin Freeman, *Information Agents for Automated Browsing*, 1996.

14. Croft, W. B. "Approaches to Intelligent Information Retrieval", *Inf. Process. Manage*. 23, 4, 1987.

15. Giorgio Brajnik, Stefano Mizzaro, and Carlo Tasso, "Evaluating User Interfaces to Information Retrieval Systems: A Case Study on User Support ", *SIGIR'96*, Zurich, Switzerland, pp. 128-136.

16. Bienvenido Velez, Ron Weiss, Mark A.Sheldon, David K. Gifford, "Fast and Effective Query Refinement ", *SIGIR 97,* Philadelphia PA, pp. 6-10.

17. Wendlandt, E.B. and Driscoll, J. R. "Incorporating a Semantic Analysis into a Document Retrieval Strategy", *SIGIR 91, Proceedings of the 14th Conference on Research and Development in Information Retrieval*, Chicago.

18. William C. Wake and Edward A. Fox, "SortTables: A Browser for a Digital Library", *CIKM 95,* Baltimore, pp. 175-181.

19. Goldberg, A., *Smalltalk - the Interactive Programming Environment*, Addison-Wesley, Reading, Mass, 1989.

20. Motro, A., "Flex A Tolerant and Cooperative User Interface to Databases", *IEEE Trans. Knowl. Data Eng.* 2, 2, 1990.

21. Scheabe, D. and Mizutani, E. E., " A Knowledge Based Browser to Access Complex Databases", *Proceedings of the International Conference on Extending Database Technology*, 1990.

22. Trigg, R. H., " Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment", *ACM Trans. Off. Inf. Systems*, 6, 4, 1988

23. X. Hao, *Automatic Office Document Classification and Information Extraction*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, August 1995.

24. X. Hao, J.T.L. Wang, M.P.Bieber and P.A. Ng, "Heuristic Classification of Office Documents", *International Journal of Artificial Intelligence Tools*, 3(2): 233-265, 1994.

25. X. Hao, J.T.L. Wang and P.A. Ng, "Information Extraction from the Structured Part of Office Documents", *Information Science*, 91(3/4): 245-274, 1996.

26. X.Hao, J.T.L. Wang, and P.A. Ng, "Nested Segmentation: An Approach for Layout Analysis in Document Classification", *Proceeding of Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, pp. 319-322, Oct. 1993.

27. C. Wei, *Knowledge Discovering for Document Classification Using Tree Matching in TEXPROS*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, May 1996.

28. C. Wei, J.T.L. Wang, X. Hao, and P.A. Ng, "In Inductive Learning and Knowledge Representation for Document Classification: The TEXPROS Approach", *Proceedings of 3rd International Conference on Systems Integration*, pp. 1166-1175, Sao Paulo, SP, Brazil, Aug. 1994.

29. F. Mhlanga, *D_Model and D_Algebra: A Data Model and Algebra for Office Documents*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, May 1993.

30. F. Mhlanga, Z. Zhu, J.T.L. Wang, and P.A. Ng, "A New Approach to Modeling Personal Office Documents", *Data and Knowledge Engineering*, 17(2): 127-158, Nov. 1995.

31. Z. Zhu, Q. Liu, J.A. Mchugh, and P.A. Ng, "A Predicate Driven Document Filing System", *Journal of Systems Integration*, 6(3): 373-403, Sep. 1996.

32. Z. Zhu, J.A.Mchugh, J.T.L. Wang, and P.A. Ng, " A Formal Approach to Modeling Office Information Systems", *Journal of Systems Integration*, 4(4): 373-403, Dec. 1994.

33. Q. Liu and P.A. Ng, *Document Processing and Retrieval: Text Processing*, Kluwer Academic Publishers, Norwell, 1996.

34. J.T.L. Wang, F.S. Mhlanga, Q. Liu, W.C. Shang, and P.A. Ng, "An Intelligent Documentation Support Environment", *Proceedings of the fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, pp. 429-436, June 1993.

35. Z. Zhu, *On Document Filing Based upon Predicates*, Ph.D. Dissertation Proposal, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, Oct. 1994.

36. J.T.L. Wang and P.A. Ng, "TEXPROS: An Intelligent Document Processing System", *International Journal of Software Engineering and Knowledge Engineering*, 15(4): 171-196, Apr. 1992.

37. Q. Liu and P.A. Ng, " A Browser of Supporting Vague Query Processing in an Office Document System", *Journal of Systems Integration*, 5(1): 61-82, 1995.

38. C.Y. Wang, Q. Liu and P.A. Ng, "Browsing in an Information Repository", *2nd World Conference on Integrated Design and Process Technology*, Dec. 1996.

39. C.Y. Wang, *The Intelligent Browser for TEXPROS*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, 1998.

40. Q. Liu, *An Office Document System With the Capability of Processing Incomplete and Vague Queries*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, Aug. 1994.

41. Q. Liu, J.T.L. Wang, and P.A. Ng, " An Office Document Retrieval System with the Capability of Processing Incomplete and Vague Queries", *Proceedings of the Fifth Intl. Conf. On Software Engineering and Knowledge*, San Francisco, CA, pp. 11-17, June 1993.

42. Marti A.Hearst and Chandu Karadi, "Cat-a-Cone: An Interactive Interface for Specifying Searches and Viewing Retrieval Results using a Large Category hierarchy", *SIGIR 97,* Philadelphia PA, USA, pp. 246-255.

43. Helena Ahonen, Oskari Heinonen, Mika Klemettinen, and A. Inkeri Verkamo, "Applying Data Mining Techniques for Descriptive Phrase Extraction in Digital Document Collections", *Advanced in Digital Libraries Conference*, Santa Barbara, California , Apr. 22-24, 1998

44. Michel Jaczynski, "A framework for the Management of Past Experiences with Time-Extended Situations", *CIKM 97*, Las Vegas Nevada, USA, pp. 32-39.

45. Aravindan Veerasamy and Russell Heikes, " Effectiveness of a Graphical Display of Retrieval Results", *ACM 1997 Proceedings – Information Retrieval*, Page 236 – 244.

46. Jason Hu, *Knowledge Management for TexPros*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, 1999.

47. Xien Fan, *An Automated Document Filing System*, Ph.D. dissertation, Department of Computer and information science, New Jersey Institute of Technology, Newark, New Jersey, 1998.

48. C. Thanos, *Multimedia Office Filing: The MULTOS Approach*, Elsevier Science Publishing Co., Inc., New York, 1990.

49. T.W.Malone, K.R.Grant, K.Y.Lai, R.Rao, and D.Rosenblitt, "Semistructured Messages are Surprisingly Useful for Computer-Supported Coordination", *ACM Transactions on Office Information Systems*, Vol. 5, No. 2, pp. 115-131, 1987.