Spring 1998

The intelligent browser for texpros

Chih-Ying Wang
New Jersey Institute of Technology

ABSTRACT

THE INTELLIGENT BROWSER FOR TEXPROS

by
Chih-Ying Wang

Browsing is a technique, which helps users to formulate their query and retrieve information in the information retrieval system. This technique provides users with capabilities of understanding their information needs and gaining system knowledge during the course of the browsing and thus it eases the users' burden when issuing queries. The basic components of the browser provides an underlying structure which allows users to navigate and a browsing process controller which provides users with the needed assistance during each browsing session.

In this dissertation, a new infrastructure (OP-Net), transformed from the existing object network is proposed. Each object in the object network is transformed into a predicate-augmented information repository. The predicate associated with each information repository governs the content of relevant documents in the depository during the browsing process and is updated continuously according to queries given by the user. The OP-Net with the relevant information repositories provides a dynamic and efficient environment for browsing.

A new ranking model is also proposed based on the signature of the documents and the user's query. The signature of a document is a document representative which utilizes the information provided by the dual model in TEXPROS (TEXt PROcessing System). With the signatures, the similarity of the document and the query can be computed, and the ranks of the documents can be derived.

This dissertation describes a three-layer architecture for the browser. At the top layer, the browsing process controller conducts and monitors the browsing process, and utilizes the services provided by the service providers. At the bottom of this architecture is the storage management system which stores the documents and then associated frame instances and responses to the requests from the service providers in the second layer. This architecture supports the principle of information hiding by allowing the change of the design of each component without changing the others. In the conclusion of this dissertation, the potential improvements and future research will be proposed.

THE INTELLIGENT BROWSER FOR TEXPROS

by
Chih-Ying Wang

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

Department of Computer and Information Science

May 1998

APPROVAL PAGE

THE INTELLIGENT BROWSER FOR TEXPROS

Chih-Ying Wang

Dr. Peter A. Ng, Dissertation Advisor — Date
Professor of Computer and Information Science, NJIT

Dr. Murat M. Tanik, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

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

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

Dr. Tina Taiming Chu, Committee Member — Date
Assistant Professor of Mechanical Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Chih-Ying Wang

Degree: Doctor of Philosophy

Place of Birth: Taiwan, Republic of China

Undergraduate and Graduate Education:

Doctor of Philosophy in Computer and Information Science,
New Jersey Institute of Technology, Newark, New Jersey, 1998

Master of Science in Computer and Information Science,
New Jersey Institute of Technology, Newark, New Jersey, 1994

Bachelor of Science in Computer Science,
Soochow University, Taipei, Taiwan, Republic of China, 1989

Major: Computer and Information Science

Publications:

C.Y. Wang, Q. Liu, and P.A. Ng. Browsing in an Information Repository. In *Proceedings of 2nd World Conference on Integrated Design and Process Technology (edited by M.M. Tanik, etc)*, IDPT-Vol. 2, pages 48-56, 1996.

C.Y. Wang, Q. Liu, and P.A. Ng. Intelligent Browser for TEXPROS. In *ISATED Proceedings of International Conference on Intelligent Information Systems (IIS' 97) (edited by H. Adeli)*, IEEE computer Society Press, pages 388-398, December 8-10, 1997.

X. Li, J. Hu, X. Fan, C.Y. Wang, and P.A. Ng. Automated Document Filing and Retrieval System: An Overview. To appear in *Proceedings of 3rd World Conference on Integrated Design and Process Technology*, Berlin, Germany, July 6-8, 1998.

This dissertation is dedicated to
my parents

ACKNOWLEDGMENT

The author wishes to express his gratitude to his advisor, Professor Peter A. Ng, who spent many sleepless nights and effort to review the drafts of the manuscripts, and also provided the valuable comments that made this final manuscript possible. Special thanks are given to Dr. Ronald S. Curtis, Dr. D.C. Douglas Hung, Dr. Murat M. Tanik and Dr. Tina T. Chu for actively participating in my committee.

TABLE OF CONTENTS

Chapter **Page**

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure **Page**

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Background

Information systems can be divided into five categories [1], namely, the management information systems, the database management systems, the decision support systems, the question-answering system, and the information retrieval systems. According to Salton's description [1], TEXPROS belongs to the information retrieval systems since it is an information system aimed to provide users with the information of potential interest based on the stored documents. TEXPROS thus has to deal with the representation, storage, and access to documents (or document representatives). The information retrieval system typically provides three facilities:

1. An interface for specifying queries,
2. A collection of algorithms for retrieving the information from stored documents; and
3. A storage system for storing the information and supporting the efficient implementation of the algorithms.

Fig 1-1 depicts the architecture of a simplified information system and the information acquisition process. The information system provides users with an interface that allows the users to specify their queries by using natural language (NL), Boolean expressions (BL), SQL, etc. Then the request will be translated into internal queries. which are sent into the storage management system, that typically is a database management system, and executed. The results will then be returned to the users. Since these three facilities (the interface, the algorithms, and the storage system) are also

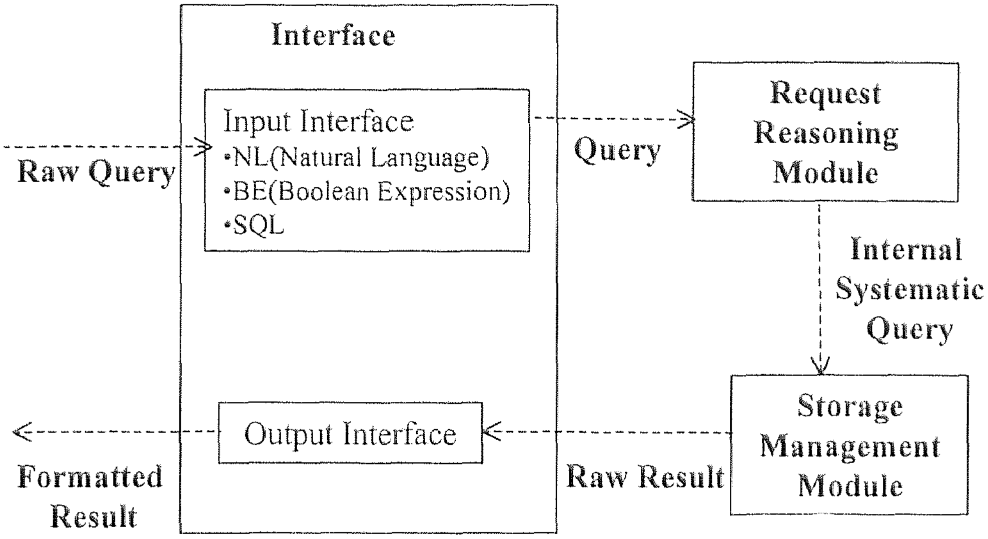


Fig 1-1 The simplified information system architecture

provided by the traditional database systems, some of the researchers treat the information system as a special case of the database system [2].

One essential difference between the database systems and the information systems is the way of handling the user's query. The traditional database systems expect that the users give the precise information about what they need. However, they always encounter many difficulties when issuing the query [3]. The reasons are as follows:

1. Most of the database systems require users to specify queries of their needs using one of the formal query languages provided by the system. Users who are not familiar with the syntax of the language cannot specify the query.
2. It is very difficult for users to precisely specify the contents of the query. The database system contains a collection of data which simulates the real world of a given problem. To specify a query, although they are familiar with the syntax of the formal language, users need to describe the contents of the query which consist of the attribute names and the values stored in the system. Users cannot

get the desired result if they don't know the correct terms for the attribute names or the values.

From the information system's point of view, the system also expects users to provide the "high quality" information need. Upon the arrival of a request, the similarity can be computed between the user's request and the information stored in the system, and the relevant information can then be retrieved. The goal of the retrieval sub-system of the information system is to retrieve possibly all the relevant information. In order to achieve this goal, the information system needs to provide a better interface than that of the database system so that users can easily and more precisely specify their needed information. The typical interfaces of information systems for specifying the query are natural language interface, Boolean expression interface and the traditional SQL or SQL-like language. With the exception of the SQL, these query languages (which will be discussed in next section) provide an intuitive way for specifying queries. The burden of issuing queries can be eased by introducing different and proper interfaces for specifying queries. With these interfaces, since the syntax is much easier than the formal query language, users need not put a lot of effort in to learning the syntax. Users also need not worry about the data model and schemas of the system before issuing the query. However, providing different interfaces complicates the internal query transformation process. In summary, instead of users, the information system tends to take full responsibilities for completing the task of formulating a query by providing users with some easy-to-use query interfaces. The information retrieval system focuses on helping users to formulate queries (more specifically, to better understand users' information needs so the relevant information or documents can be efficiently retrieved). From this aspect, the retrieval process of the information retrieval system diverts from that of the traditional database system.

Another difference is that the information system emphasizes the feedback process. The query specified by users is treated as an indication of the needed information in information retrieval system. However, the needed information may not be precisely specified in the query. The reasons are as follows:

1. The user may have the definite idea of what he/she needs, but he/she does not know how to specify it or simply forgets the exact terms. For instance, assume that a mechanic needs to look up the information of a part in order to tell the customer its exact cost. However, the only information he/she knows is that the part belongs to the suspension system. In this case, the initial query will somehow relate to the term "suspension system" according to the interface provided by the system. The mechanic can then examine the returned preliminary information and refine the original query to locate the information he/she needs.
2. The user may have a concept or an exact term in mind, without the perception of what he/she needs when first coming to the system. For instance, the user may want the information of the document extraction. When coming to the system, he/she first only has "extraction" in mind. Therefore the initial query will somehow relate to the term "extraction". Then the results returned by the system may reveal that this term has different meanings in different fields (e.g. computer information science and chemical engineering) and guide the user to refine his/her original query.

However, the system will help users to continue the retrieval process by presenting the preliminary results with the indications of how close it is to the query. The feedback process is important in the sense that it improves the quality of the result in a controlled and systematic way. On the contrary, in database system, when the result of the query (specified in formal query language) is returned, the retrieval process halts. Users find it very difficult to refine their original query based upon the returned result owing to the lack of suggestions and the guidance.

1.2 Related Work

To ease the burden of formulating the query, the early research focused on providing the traditional database system with a better language. One of the early user-friendly database languages was the example-based database language. Zloof's Query-by-example (QBE) was the earliest graphical database query language [4]. In QBE, instead of writing lengthy queries, users specified the example output by making entries into relation skeletons. There are numerous example-based languages including Summary-Table-by-Example, Time-by-Example, Generalized-Query-by-Example, Office-by-Example, Formanager, Picquery, etc [5]. These languages provide a two-dimensional, graphically aided example for formulating queries; but they have different features. For example, the Generalized-Query-by-Example supports nested relations and the Summary-Table-by-Example allows the user to produce a summary table by using two-dimensional skeletons. The advantage of using an example-based query language is that the user does not have to know the syntax of the language or the data model schema such as the attribute name. However, the major drawback of these languages is the inability to handle complicated queries.

From the information system's point of view, although it eases the burden of issuing a query, example-based query language lacks a mechanism to refine the query. To support the need of information retrieval discussed in the last section, the information system needs a mechanism for returning the result along with the guidance according to user's query. This mechanism requires an interface for users to specify the queries easily and an underlying structure which can hold the intermediate result and the suggestion provided by the system.

Browsing technique has received more attentions since 80's after the advent of the relational database management system (DBMS). The users of the DBMS have difficulties of issuing queries by using the formal query language provided by the system. Under this situation, browsing mechanisms, such as Cattell's browser for the entity-relational database [6], SDM [7], TIMBER [8], Motro's browser for loosely structured database [9], BAROQUE [10], and KIVIEW [11] are helpful for the users. However, there are some limitations among the browsers for databases [3, 10]. One of them is the scrolling boundary for browsing. The earlier browsing mechanism used for relational databases is restricted to one relation at a time. To solve this problem, Motro proposed a loosely structured database, which eliminates the difference between the schemas and values [9, 10].

In traditional text-based information retrieval system, documents are represented by collections of index terms called the representatives of documents. When a user issues a query by a method, which is more intuitive than the formal query language, it is processed and represented by a collection of index terms. The retrieval system then seeks the representatives of documents which match the representative of the query. From the database point of view, the information retrieval system is similar to the loosely structured database in the sense that the whole collection of documents is treated as a universal relation. Much of the research work has been on finding better representatives for documents and developing faster searching techniques [1, 12]. However, little effort has been made to help users formulate a better query [13, 14, 15]. That means, in the information retrieval system, users can encounter the same problems as in the relational database system but the browsing techniques for assisting query formulation can be used in information retrieval systems. CANSEARCH [16] and CoalSORT [17] are browsers which aimed to help query formulation. Some systems, including ZOG [18], I^{3}R [2], and Kabiria [19], built their browsers as general-purpose interfaces between users and the system. These browsers not only assist users to formulate their query but also provide users with an environment to explore the system knowledge and examine the documents. As an interactively searching process involving the user and the system, the browsing process will guide users to express precisely and to gain the information needed, step by step.

Almost all the browsers we mentioned here share the following common properties. They constructed a network-like structure as an underlying structure [20, 13, 21, 22, 23, 24], mostly the semantic network, which forms a browsing space. The browsing process can traverse the network. Sometimes, the browsing process cannot proceed further without the user understanding the meanings of the links of the semantic networks and selecting links for traversing across the network. The other problem is that most of the browsers provide only "short-sighted" browsing. The process can only start from a node of the network and traverse to one of its neighbors. The relevant nodes which are not neighbors of the current node cannot be examined. The final problem is the performance issue. The browsing process heavily counts on the feedback process, which could be time consuming because of the large number of the documents in the collection [2, 15].

In this dissertation, we propose a browser for TEXPROS. This browser consists of a query interface for users to issue queries, a knowledge base for resolving users' information needs, a ranking system with ranking functions for evaluating the retrieved documents, and a retrieval process controller for monitoring browsing sessions.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we briefly introduce TEXPROS and then describe the underlying network and its construction. In Chapter 3, we discuss the browsing processes provided by the system. In Chapter 4, we describe the system architecture which supports the browsing processes. Among the components in the system architecture, we give special attention to the knowledge base and ranking unit in Chapter 6 and 7, respectively. We give the conclusions and our future research work in Chapter 8.

CHAPTER 2

NETWORK TRANSFORMATION

2.1 Previous Work

In TEXPROS [25, 26], there are four kinds of objects, namely folders, frame templates, attributes and values. Documents are deposited into folders, which are organized as the *folder organization* (FO) [26, 27]. Documents are classified into different types. Those of the same type share common properties, which are characterized in terms of attributes to form a frame template. The frame templates are organized as the *document type hierarchy* (DTH) (Fig 2-3) [26, 28]. The relationships among objects are summarized in Fig 2-2. The system catalog serves as the depository of this information (called the meta-data knowledge) and the information about the database itself. It supports all the activities of the system and thus it is the central part of the system. Both the system catalog and database itself are represented uniformly.

These meta-data and information about the database itself can be organized as a semantic network called the *object network* (ON) [14, 29, 26], which captures all the relations of the objects in the system.

An object network is created to describe the view of the meta-data of TEXPROS (i.e. the document type hierarchy and the folder organization) and the documents (frame instances)[14, 25]. The characteristics of the object network include:

1. **The dual model is captured.** The dual modeling approach is used for classifying and categorizing documents in terms of the document type hierarchy and the folder organization. The system catalog contains information describing the folder organization and document type hierarchy. Both descriptions of the

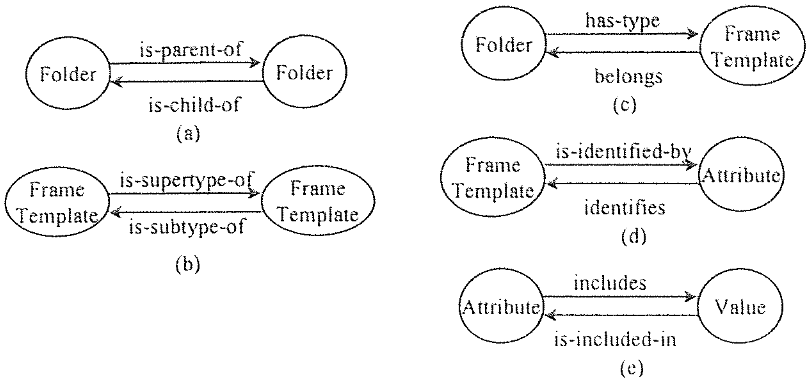


Fig 2-2 The objects and the relationships

folder organization and the document type hierarchy with their contents are unified as a single description, called an object network.

2. **The information of all stored documents is captured.** The *access by value* gives users the capability of retrieving all the occurrences of an attribute value from the database. The occurrences of an attribute value are the values of a given attribute. In order to realize the method of *access by value*, an item directory is needed to store the mapping from the values into attribute names [10].

3. **A snapshot of the system catalog is provided.** TEXPROS is a dynamic document processing system. At any time, the object network always provides a

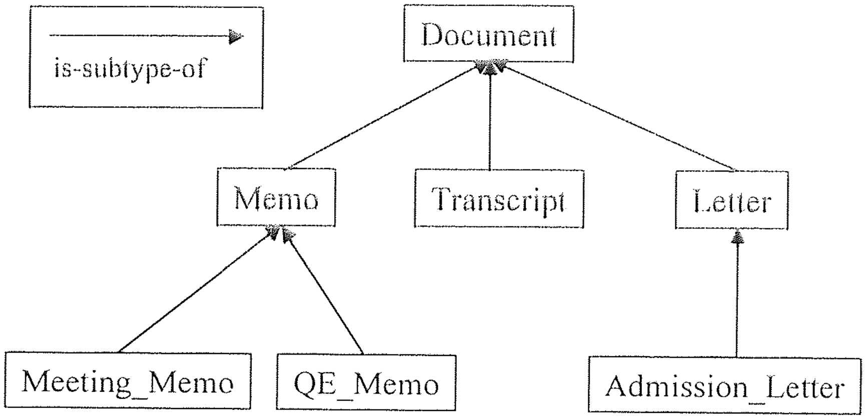


Fig 2-3 Document type hierarchy

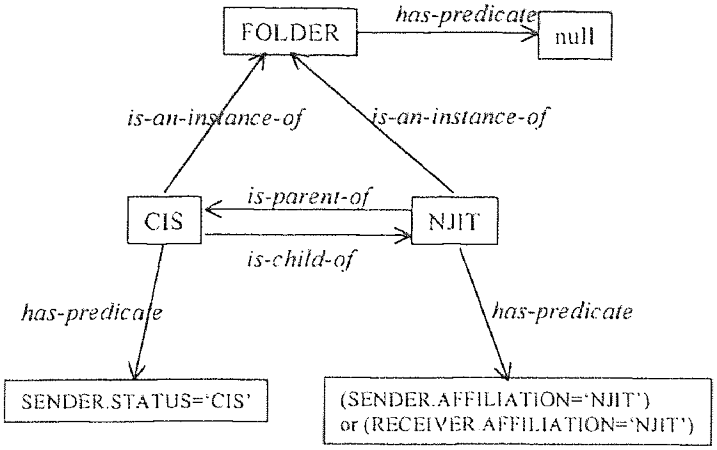


Fig 2-4 Semantic Network

consistent snapshot of the current system. Providing a snapshot could enhance the performance of the browsing process, as well as the classification and filing processes.

2.2 Motivation

A semantic network consists of two major components, namely the nodes connected by the links. The semantic meanings of these links and nodes are varied depending upon the system [15, 30, 31]. Generally speaking, each node represents an object and each link represents a relationship that connects two related objects. Given a node on the semantic network, each of its outgoing links has an attribute. The labels of the links are utilized to describe their semantic meanings. However, the excessively large number of links of various types becomes a burden for the user [15], because they have to understand the meanings of these links.

In TEXPROS, the object network is a semantic network [30, 31]. It can be used to describe the folders and the relationships among the folders and the document type hierarchy with the attribute-values pairs. This helps the system to answer queries. If we provide users with this network, then they are able to find out all the meta-data knowledge in the system, which can help them, to some extent, to formulate their queries. For instance, given a simplified semantic network as shown in Fig 2-4, we are able to find out the child folder of the folder NJIT, if we follow its outgoing link *is-parent-of*. The object network of TEXPROS has four object classes FOLDER, FRAME TEMPLATE, ATTRIBUTE, or VALUE. From the object-oriented point of view, all the folders in TEXPROS can be organized as a class. They share a common property: each folder contains frame instances which satisfy its criteria specified in terms of a predicate. The relationship *is-an-instance-of* is used to describe the associations between instances and their classes. In Fig 2-4, CIS and NJIT are two instances of the class FOLDER. The FOLDER class has an attribute called *has-predicate*, and every instance of FOLDER also inherits the attribute *has-predicate*. In the example, the predicate of the folder CIS is *SENDER.STATUS='CIS'*. In Fig 2-4, the sub-folder and parent-folder relationships are represented by two attributes, *is-parent-of* and *is-child-of*. According to [31], we need to decide at what level of the knowledge that the system intends to represent when using the semantic network. In object network, the knowledge is represented at the instance level. This supports the "access by value" [10].

However, there are some drawbacks for browsing across the object network:

1. One of the issues of the semantic network is that how the property of an object described by the attribute values can be accessed. Since the relationships do not satisfying the property of transitivity, it is not a trivial task to identify precisely the related objects in the object network. Consider the simplified object network

in Fig 2-5. In this network, three different kinds of relationships are presented, namely *has-type*, *has-attribute*, and *has-value*. There are two folder objects (NJIT and CIS); each of the folders NJIT or CIS contains documents of a type referred to as a frame template object (Letter); the frame template object consists of an attribute object (Sender), which has two value objects (Roy and John). Since the folder CIS is a child of the folder NJIT, and the folder CIS contains documents of the Letter type, we conclude that both NJIT and CIS folders contain documents of the Letter type. In TEXPROS, a frame template specifies a document type in terms of attributes. Following through the relationship *has-attribute* in Fig 2-5, we can conclude that the attribute Sender is an attribute of the frame template Letter. Given the attribute Sender, following through the *has-value* relationship, we find its associated values, Roy and John. In this case, we can conclude only that there are some documents, which are sent by Roy or John. It would become more complicated if we add that the folder CIS contains documents of the Memo type, which has an attribute Sender also. Now, we cannot conclude that the CIS folder contains a letter, which is sent by Roy, although we already had some conclusion from the earlier explanation. The reason is that these relationships, *has-type*, *has-attribute*, and *has-value* do not satisfy the transitivity property. Therefore, it is not a trivial task to identify the related information from an object in the realm of the object network.

The object network (ON) is used to capture the knowledge, which contains the properties of objects and the relationships between objects. However, these captured relationships tend to explore the structural relationship between objects instead of the real semantic relations. For example, it is difficult to use ON only to answer the following query: (Q1) Find documents which were sent by Roy. Assume that the folder organization has a folder ROY, which is a child of the folder CIS. We may retrieve the documents sent by Roy from the folder Roy. However, some documents sent by Roy can also be located in the other folders. Therefore, for the worst case, we may have to search through every folder of the folder organization to find the documents sent by Roy, although these documents must satisfy "Sender = Roy". This is a time-consuming process. Let us take a look at those processes prior to the retrieval process before we reveal the reason of this problem.

3. In TEXPROS, the object network is physically stored in the system catalog [14, 26]. At the arrival of a new document into the system, it goes through the classification and extraction process. This process generates the frame instance for the document. Then the filing process stores this frame instance in the appropriate folders. Each process is responsible for updating the system catalog. However, after completing these processes, the object network does not keep the relations between the frame instance and the objects such as folders where it is kept, the frame template which specifies its type, etc. The connection between the objects in the object network and the frame instances in the frame instance base is lost. Without this connection, the browsing process cannot be performed at the frame instance level. Therefore, upon the arrival of a query like Q1, the associated frame instances cannot be found immediately.

The object network is used to help users to retrieve the documents they want. Since ON cannot fully support the associations between the frame instances and the objects, we need to transform ON into another network, which has the following characteristics:

1. **To support the browsing process**. The transformed network that supports the browsing process must be able to enlarge or reduce the scope of the search throughout the course of browsing.
2. **To answer any questions related to the system catalog**. The transformed network must be able to answer questions related to the system. The obtained network can respond to any query in the way that the ON does.
3. **To retrieve the documents**. The transformed network must be able to capture the associations between frame instances (each corresponds to a document) and the objects such as folders (where the frame instances are resided), frame templates (which classify the documents into various types), attributes (which characterize the properties of the document types).

In the next section, we shall describe the network transformation.

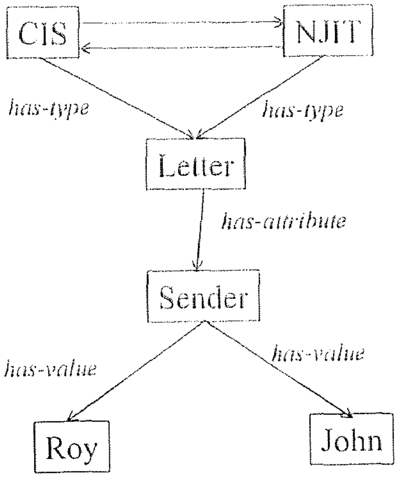
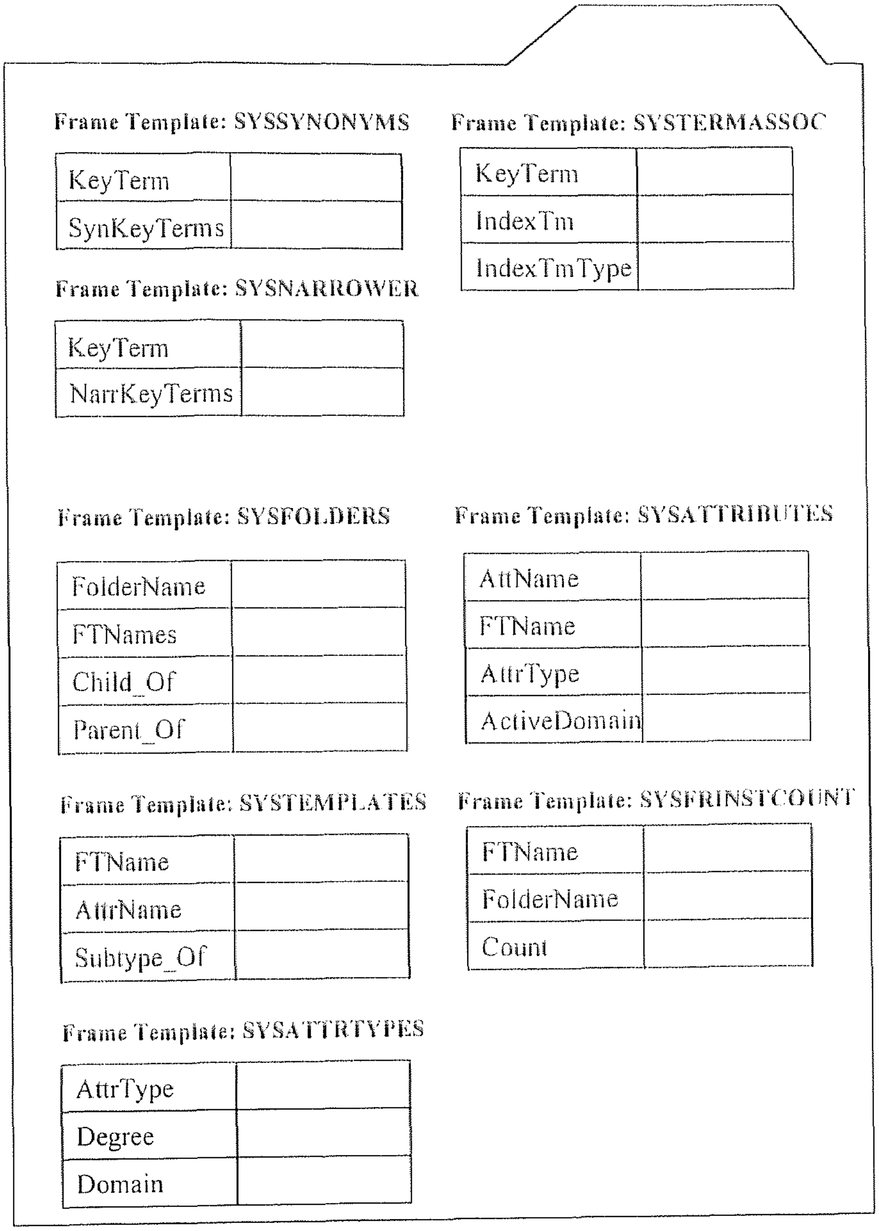


Fig 2-5 Semantic network with objects of different type

2.3 Network Transformation

The first step of the transformation is eliminating the relations among the objects of the same object type. Therefore, the information of the folder organization and the document type hierarchy will no longer exist in the transformed network (the explanation this removal will be given in the later sections). Then objects of the same type are grouped into classes, namely FOLDER, TEMPLATE, ATTRIBUTE, and VALUE. This simplifies



Frame Template: SYSSYNONYMS

KeyTerm	
SynKeyTerms	

Frame Template: SYSTERMASSOC

KeyTerm	
IndexTm	
IndexTmType	

Frame Template: SYSNARROWER

KeyTerm	
NarrKeyTerms	

Frame Template: SYSFOLDERS

FolderName	
FTNames	
Child_Of	
Parent_Of	

Frame Template: SYSATTRIBUTES

AttName	
FTName	
AttrType	
ActiveDomain	

Frame Template: SYSTEMPLATES

FTName	
AttrName	
Subtype_Of	

Frame Template: SYSFRINSTCOUNT

FTName	
FolderName	
Count	

Frame Template: SYSATTRTYPES

AttrType	
Degree	
Domain	

Fig 2-6 System Catalog

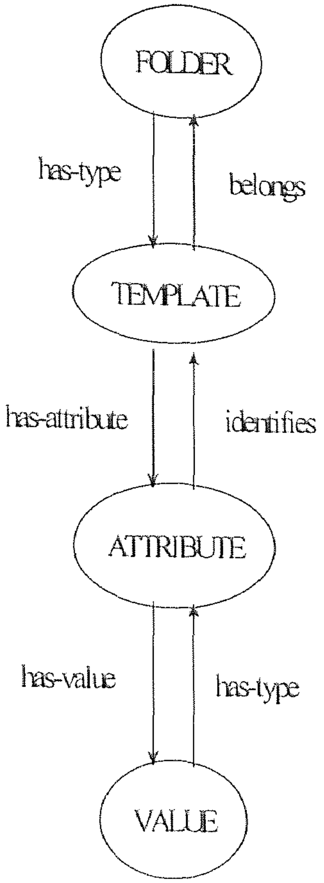


Fig 2-7 Network transformation -- Step 1

the ON (shown in Fig 2-7). By introducing the concept of classes, we can construct a semantic network at a higher abstract level by considering the behaviors at the class instead of the instance level.

The next step is to simplify the link relations among object classes as shown in Fig 2-7. A new temporary relation at a higher abstract level is created. That means all the link relations in Fig 2-7 can be generalized into a unique relation *relates* depicted in Fig 2-8.

The frame instances which are stored in the frame instance base can also be considered as a class FRAME-INSTANCE. In order to capture the associations between objects and frame instances as we have described in the previous section, we introduce eight new relations[1] between FRAME-INSTANCE and the object classes FOLDER, TEMPLATE, ATTRIBUTE, and VALUE, introduced in Step one. These relations are created by heuristics [32, 33] to capture any relation which is significant to the system. For example, relation IsInFolder and its inverse relation HasFInstance are created between the FOLDER and FRAME-INSTANCE to capture the fact that each frame instance is deposited in some folders. These relations and the facts captured by the system are shown in Table 2-1. Fig 2-9 is the transformed network after we augmented these new relations into the transformed network in Fig 2-7.

The efficiency of document retrieval performance could not be enhanced if we simply introduce the FRAME-INSTANCE class to the system. The reason is if the semantic network is employed for retrieving documents, we have to return to the instance level, in which every instance will be represented by a node. For users to retrieve the documents by going through such a huge network could be frustrated. The other problem is that during the course of the retrieval process, the browsing process has to identify a sub-network from the original network. Without reducing the size of the original network, the process for identifying a sub-network can be time consuming. Therefore, we must eliminate as many as possible objects and links from the network without downgrading the achievement of the retrieval goals. The central part of this step of transformation is to transform the object in ON into *frame instance repository* (FIR).

[1] Since the association between each object class and the frame instance class should be defined for both directions, we actually created eight new relations (four relations and their inverse relations).

2.4 Object Transformation

In the previous section, we enriched the semantics of the ON by introducing the frame instance class at the cost of enlarging the size of the network, which is not a desirable property. The goal of object transformation is to capture the associations between objects and frame instances without creating an enormous size of the network.

In Section 2.3, a set of relations between classes (as shown in Table 2-1) is defined for describing associations between an object and its associated frame instances. These relations are total relations. For instance, the IsInFolder relation between the class FOLDER and the class FRAME-INSTANCE is a total relation because all the instances of the FOLDER and FRAME-INSTANCE participate in this relation. Given an instance of the FOLDER, a set of frame instances could be identified according to the relation defined between the class FOLDER and the class FRAME-INSTANCE. In the

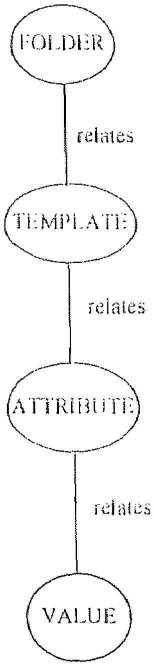


Fig 2-8 Network transformation -- Step 2

Table 2-1 New relations

Relation	Association	Fact
IsInFolder(fi, F)	FOLDER, FRAME-INSTANCE	Frame instance fi is deposited in the folder F.
IsInFolder^{-1}		Folder F contains frame instance fi.
IsOfTemplate(fi, FT)	TEMPLATE, FRAME-INSTANCE	Frame instance fi belongs to document type FT.
IsOfTemplate^{-1}		Template FT has a frame instance fi in the frame instance base.
ContainsAttr(fi, A)	ATTRIBUTE, FRAME-INSTANCE	Frame instance fi contains the attributes A.
ContainsAttr^{-1}		Attribute is in the frame instance fi.
ContainsVal(fi, V)	VALUE, FRAME-INSTANCE	Frame instance fi contains the value V.
ContainsVal^{-1}		Value V is in the frame instance fi.

identification process, the IsInFolder behaves like a predicate which instantiates all the instances. IsInFolder(fi, F1) specifies all the frame instances fi, which are kept in the folder, F1. Likewise, the other relations defined in Table 2-1 are self-explanatory. Based on this concept, each node in the original ON is transformed into a frame instance repository. In the remaining section, we shall define formally the object and FIR.

Definition 2-1: (Object)

An object is a two-tuple, $\text{Obj} = [\text{Name}, \text{Type}]$ where:

1. Name is the name of the object.
2. $\text{Type} \in \{\text{Folder}, \text{FrameTemplate}, \text{Attribute}, \text{Value}\}$

We shall use the notations Type(Name) and [Name, Type] interchangeably.

According to this definition, every node of ON is an object and each object belongs to an object Type. By using name and type for identifying an object, objects of different types are allowed to have the same name.

Definition 2-2: (Frame Instance Repository)

A frame instance repository is a four-tuple FIR = [Obj, P_O, FI_{OP}, FI_E] where:

1. Obj is an object, [name, type].

2. P_O is a predicate defined on the Obj, which is one of IsInFolder, IsOfTemplate, ContainsAttr, and ContainsVal.

3. FI_{OP} = {fi|fi is a frame instance which satisfies P_{OP}}, where P_{OP} is the predicate derived from the user's vague query.

4. FI_E = {fi|fi is a frame instance which satisfies P_E}, where P_E is the predicate derived from the topic of the exploring process.

In this dissertation, we will use FIR(Obj) to refer to the frame instance repository associated with the object Obj. We also use P_O(Obj) to refer to the predicate associated

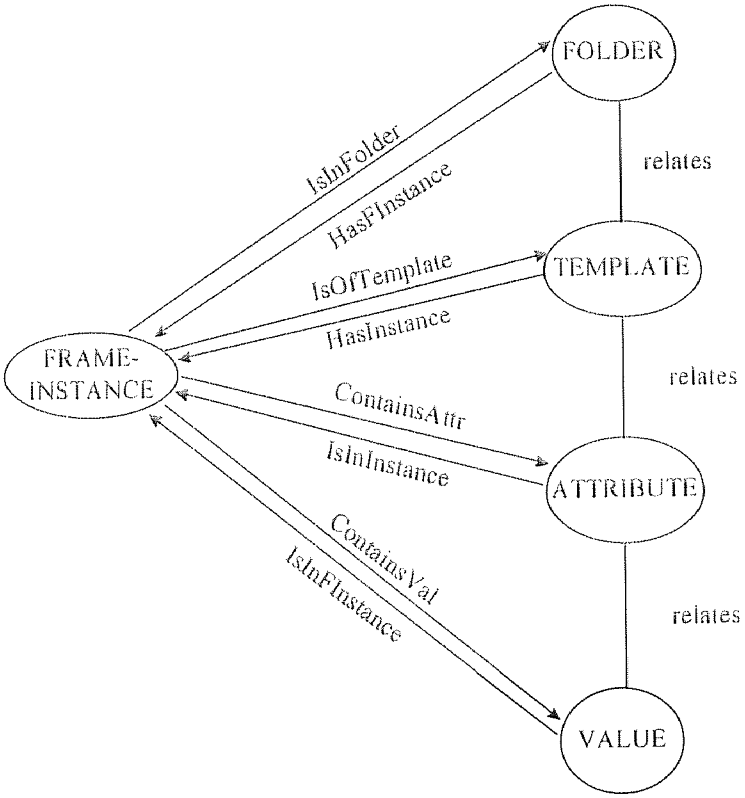


Fig 2-9 Network transformation -- Step 3

with the object Obj. This notation also applies to FI_{OP}(Obj) and F_{IE}(Obj). Note that, when the object type is trivial, the object name alone can be used to identify the Obj.

Since the associations between the object and the frame instances are embedded in the FIR, a new transformed network is obtained as shown in Fig 2-10 (in which, the double-lined circles represent the FIR class).

2.5 Operation Network

By transforming the original ON into a new network, the transformed network includes the associations among the frame instances stored in the frame instance base and the objects stored in the system catalog. It provides a better environment (in the sense of preciseness and efficiency) to users for effective browsing. Our intention is to use this

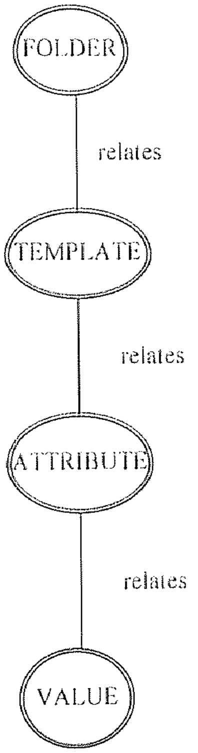


Fig 2-10 Network transformation -- Step 4

transformed network as an underlying structure for browsing. This transformed network is called an *Operation Network* (OP-Net).

Definition 2-3: (Operation Network)

An operation network (OP-Net) is a four-tuple, $OP = [T_{OP}, P_{OP}, FI_{OP}, G(V, E)]$, where:

1. T_{OP} is a topic related to the context of the browsing process;
2. P_{OP} is a predicate related to the topic T_{OP};
3. FI_{OP}={fi | fi is a frame instance satisfying P_{OP}}; and
4. G(V, E) is a graph, where
   - Each node in V(G) is a frame instance repository; and
   - Each edge (i, j) between two repositories FIR(i) and FIR(j) represents that these two repositories have at least one common frame instance.

In the definition of OP-Net, the first component of the four-tuple T_{OP} is a topic which is specified in terms of objects, connected using connectives AND, OR and NOT. Any query in the form of Boolean expression [1, 14, 25] can be transformed into a topic. For the rest of this paper, the term *topic* and *query* will be used interchangeably. The inclusion of T_{OP} in the definition captures the fact that the OP-Net is dynamically changed over topics. The browsing process can be viewed roughly as a topic refining process. Until deriving the desired result, users will keep changing from one topic to the other. The introduction of a new topic requires reconstructing the OP-Net. Initially, assume that the topic is *null*. In this case, since the OP-Net is derived by transforming the ON, before a user issues a topic, there corresponds a peer node and a link in the ON for every node and the link in the OP-Net, respectively.

The second and the third components should be considered at the same time. P_{OP} is the predicate of an OP-Net. This predicate is produced by the system based upon the interpreted query. It is specified in terms of relations between frame instances and objects connected using connectives AND, OR and NOT. For example, given a topic CIS, after topic interpretation, we find that the object CIS is a folder. If we use fi to represent a frame instance, then the predicate P_{OP} can be represented by IsInFolder(fi, CIS) which characterizes the associations between the objects and the frame instances given in Table 2-1. Any frame instances that qualify this predicate appear in the OP-Net. FI_{OP} is the set of these qualified frame instances.

The first three components defines the global property (i.e., the domain and the contents of each node) of the OP-Net in a certain state of the browsing process. The fourth component is a graph which describes the impact of a topic on each relevant frame instance repository. This graph will be the interface displayed to users. For the frame instance repository, the predicate P_O of an object is used to define all the frame instances which satisfy the predicate. In the initial state, all the qualified frame instances are associated with the object. However, during the browsing process, we may augment strict conditions to the global predicate P_{OP} and thus reduce the number of the frame instances associated with the frame instance repository. This set of frame instances is stored in FI_{OP} and is dynamically updated throughout the browsing process.

The OP-Net is defined at the instance level. The network shown in Fig 2-10 is a skeleton of OP-Nets. The underlying network for browsing is defined by representing each FIR a node in the OP-Net at the instance level.

2.6 The Relation between OP-Net and ON

An OP-Net is derived by transforming the ON. All the relations in ON fall into two categories, either horizontal or vertical relations. Generally speaking, the horizontal relations are the relations between objects of the same object type and the vertical relations are those between objects of different object types. However, the horizontal relations do not appear in the OP-Net, because the property of the links in OP-Net and ON are different. The only relation type *has-common-frame-instance* in the OP-Net is defined on frame instances. Very often, the relations of the horizontal type are difficult to convert into a frame-instance based relation. Consider the folder organization as an example. It can have more than one filing path for some specific folders. Therefore, the frame instance in the child folder which has more than one parent folders may not be in one of its parent folder. With this, it is difficult to convert the relations among folders into frame-instance based relations. These relations which are not in OP-Net, can be obtained from the ON. This will be explained in Chapter 6.

CHAPTER 3

BROWSING PROCESS

Once the ON is transformed into an OP-Net, the OP-Net will serve as an underlying structure, upon which add-on functionality will be added. A typical browsing process consists of the following phases: the topic interpretation and rewriting, the OP-Net construction, the topic refinement, the fact exploring, and the result examining. The browsing process proceeds interactively. The order of the phases is insignificant. For instance, a user may refine their original topic after examining the result. Some users may go directly to refine the original topic immediately after viewing the constructed OP-Net. Users are allowed to arrange the phases of the browsing process in order to meet their needs. The flow of the browsing process is depicted in Fig 3-11.

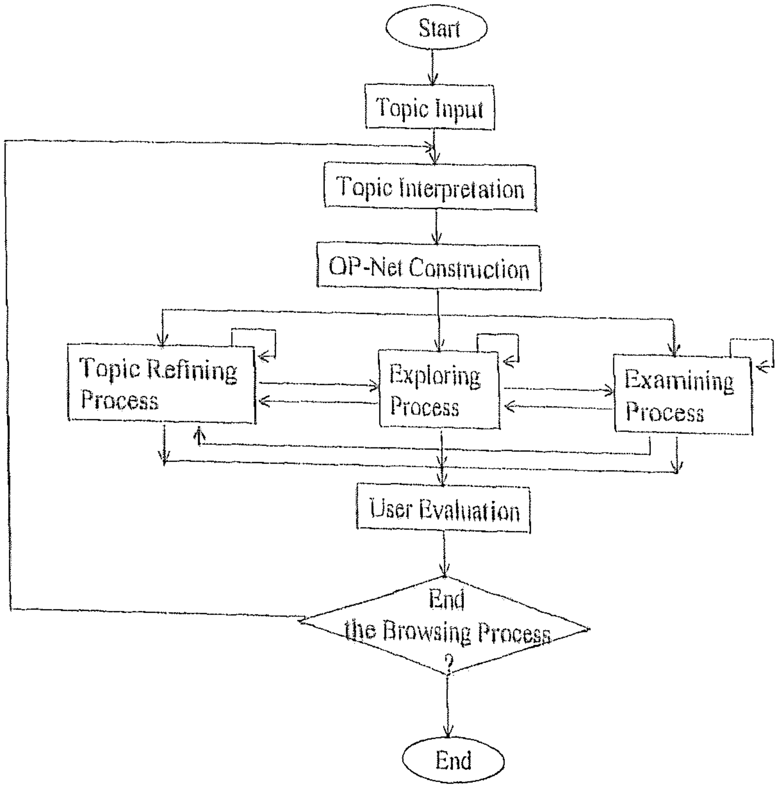


Fig 3-11 The Browsing Process Flow

In this section, we shall briefly describe the browsing process and the rationale of each sub-process. We will reveal more details of these processes when we discuss the browsing system architecture in Chapter 4 and Chapter 5.

3.1 Topic Input and Topic Interpretation

The input of this process is a query (called raw query) issued by the user. Before the system can do further processing, the raw query is required to be transformed into a topic. We use topic to represent a vague query. Our intention for using topic is to have a clear separation between the formal query and the vague query. When issuing the formal query, users are constrained to use the formal query language, which is normally SQL or SQL-like language. However, most information systems provide an easy-to-use interface when dealing with the vague query. For example, by using a Boolean expression the user needs not spend so much time becoming familiar with the syntax. Moreover, the logic of the user's thoughts can be easily expressed by using those basic logic operators such as AND and OR. Another common interface is the one which is supported by the natural language processing (NLP). NLP provides an interface that allows users to make a short statement to describe their information needs. The users have no problem using the natural language to state any expressions. However, there is an overhead for dealing with the language. The difficulty of using the NLP is given in [1].

Currently, TEXPROS supports only Boolean expressions. In the beginning of this section, we have explained briefly the rationale of the topic interpretation, and we will discuss it in detail when we present the browsing system architecture. Since users are offered to use the Boolean expression to express their information needs, without excessive usage of knowledge or inferences, most of the concepts in a user's mind could be represented by a group of key-terms connected by the logical operators. However, the system catalog in TEXPROS provides users with a lot of knowledge, which can help reach the high recall or precision. For example, a user issues a vague query CIS. If the system has no knowledge, then only the documents containing the term CIS will be retrieved. However, the term CIS that the user has in mind might be the location of the document instead of the content of the document. That means the user perceives that this intended document possibly is in a folder called CIS. In TEXPROS, the topic interpretation process will identify the CIS both as a term contained in a document and as a folder in the folder organization. In this way, both the documents containing the term CIS and the documents stored in the CIS folder will be retrieved. In this case, we improve the recall [1].

3.2 OP-Net Construction

The input of this process is derived from the topic interpretation. After the topic interpretation process successfully identified the topic, the system is able to find its relevant documents. Based upon these relevant documents, the OP-Net constructor constructs an OP-Net to include them as an environment for browsing process. All the key terms in the topic are transformed into objects which are represented in a format ObjType(ObjName). The new-formed topic contains the key terms with their types, connected by the logic operators. Therefore we combine the relevant documents and the relevant objects together to form an OP-Net. By specifying the relationships between the objects and the documents, we can provide an environment for incrementally formulating the precise query. Therefore, one of the most important tasks of this process is to find those relationships.

3.3 Topic Refining Process

As shown in Fig 3-11, after initialization, a browsing session can be proceeded by entering into the topic refining mode, the exploring mode, and the examining modes interchangeably. Each mode is conducted by the controller. The details of these controllers will be presented in the next chapter.

Consider a query as a set of conditions [34]. Assume that a user issues a null query at the beginning of the browsing. Since the null query does not have any conditions, the whole collection of documents can be the answer. The user starts the browsing session by issuing a query. In this case, the user has already performed the topic refining process

T_{OP} = Null
P_{OP} = Null
FI_{OP} = {all the frame instances in the frame instance base}

Before Topic Refining Process:
Frame Instance Repository : FR(F1)

Name	F1
Type	Folder
Po	InFolder(fi, F1)
FI_{OP}(F1)	*{1, 2, 3, 4, 5, 6}*
FI_E(F1)	Null

⇨

T_{OP} = V2
P_{OP} = ContainsVal(fi, V2)
FI_{OP} = {fi|ContainsVal(fi,V2)}
= {2, 3, 5}

After Topic Refining Process:
Frame Instance Repository : FR(F1)

Name	F1
Type	Folder
Po	InFolder(fi, F1)
FI_{OP}(F1)	*{2, 3, 5}*
FI_E(F1)	Null

FI_{OP}(F1) = {fi| InFolder(fi,F1) AND ContainsVal(fi,V2)}
= {2, 3, 5}

Fig 3-12 The changes of FIR(F1)

because s/he has refined the original null query. From this perspective, we say that every browsing session starts at the topic refining process.

The other case for performing the topic refining process is quite straightforward. During the process of browsing, after gathering some knowledge, the user is ready to refine his/her original query. When the interface controller sends the user's topic refining request to the *topic refining process controller* (which will be discussed in the next section), the process controller will arrange the procedure for performing and generating the request.

Upon the arrival of the query issued by the user, the topic refining process outputs an OP-Net corresponding to the query.

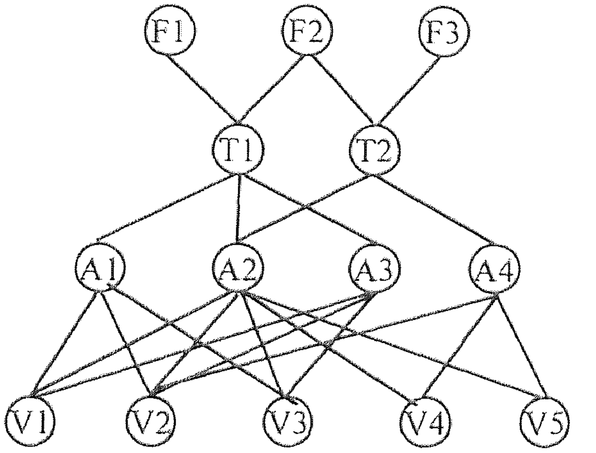
During the topic refining process, the frame instance repositories are derived from the object network. For each new topic, the OP-Net needs to be reconstructed. For simplicity, let us consider the initial state of the browsing process. In the initial state, the topic is *null*. Then the user issues a topic T_{OP}=V2. After pre-processing, the topic is rewritten as Value(V2) and the predicate P_{OP}=ContainsVal(fi,V2), and $FI_{OP} = \{fi|ContainsVal(fi, V2)\}$ are computed. FI_{OP} contains the frame instances relevant to T_{OP}, which contains the value V2. Then the graphical representation G(V,E) of the OP-Net is constructed. Assume that there is an object F1 of the OP-Net which is a folder. The left hand side of Fig 3-12 is the frame instance repository corresponding to the folder F1 before the topic refining process. FI_{OP}(F1) will be changed to reflect that the refinement of the topic from *null* to V2. Every frame instance in FI_{OP}(F1) will be checked whether it satisfies the new predicate $P_O \wedge P_{OP}$, which is $ContainsVal(fi, V2) \wedge IsInFolder(fi, F1)$ in this case. The frame instances in FI_{OP}(F1) will be removed if they are not satisfied the new predicate. If $FI_{OP}(F1) = \varnothing$, then F1 is irrelevant to the current topic and will be removed from the new OP-Net. The right hand side of the Fig 3-12 is FIR(F1) after the new T_{OP} was introduced. Fig 3-13 is the resulting OP-Net for the topic V2.

3.4 Exploring Process

For browsing, the topic refining process seems to be adequate. The state of the browsing process is changed by introducing a new query to the browser. Either issuing a new query or modifying the original query can be considered as constructing a new query. If this is the case, the topic refining process is adequate. However, our reasons of having the exploring process are as follows:

1. Users tend to modify their original queries during the browsing process. The AND operation is commonly used for modifying the queries during the browsing [1, 29], and is used to reduce the searching space.
2. At the end of the topic refining process, the process controller generates an OP-Net. The OP-Net is construct by consulting the system catalog and the frame instance base. This could be time consuming.

These two observations inspire an improvement in the topic refining process. The basic idea is that at any state of the browsing process, the intermediate result may be useful for the successive processes. For instance, consider a user's original query *(CIS AND Peter)*. After pre-processing, CIS will be identified as a folder and Peter will be identified as a value. Then the OP-Net shows the user the relations between the CIS folder and two kinds of documents: LETTER and MEMO. The user then realizes that although not sure what kind of a document is desired ,documents of the letter type are of



Top = V2
Pop = ContainVal(V2)
Flop = {1, 2, 3, 4, 5}
FI(F1) = {1, 2, 3}
FI(F2) = {2, 3, 4}
FI(F3) = {4, 5}
FI(T1) = {1, 2, 3}
FI(T2) = {4, 5}
FI(A1) = {1, 2, 3}
FI(A2) = {1, 2, 3, 4, 5}
FI(A3) = {1, 2, 3}
FI(A4) = {4, 5}
FI(V1) = {1, 2, 3}
FI(V2) = {1, 2, 3, 4, 5}
FI(V3) = {1, 2, 3}
FI(V4) = {4}
FI(V5) = {5}

Fig 3-13 A snapshot of the OP-Net

interest. In this case, the original query should be refined to *(CIS AND Peter AND LETTER)*. Without the exploring process, this refinement requires reconstructing the OP-Net. Since the user is not sure what is needed, s/he may go back to the original query and refine the query to *(CIS AND Peter AND MEMO)*. Then the system will be busy in reconstructing the OP-Net with respect to each refinement. We therefore introduce an exploring process, which allows the system to construct exploring networks (E-Net) from an existing OP-Net or E-Net, with respect to a given query.

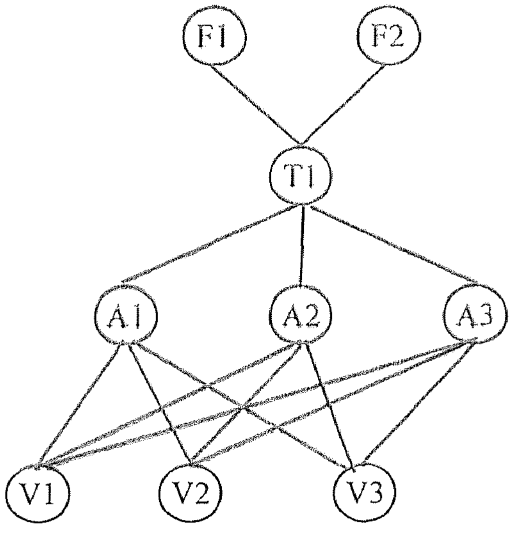
3.4.1 Exploring Network

The exploring process can be activated by users at any time during the browsing process. When a user activates the exploring process, the topic refining process will be disabled. The *exploring process controller* will then control the browsing process. During the exploring process, users are allowed to issue queries within the realm of the current OP-Net. The objects which could be used in the query to be issued are restricted to those on the current OP-Net. Given the query, the system constructs an Exploring Network (E-Net) to display the result. Given the same OP-Net, many E-Nets can be constructed, each of which reflects an input query. We treat each E-Net and the OP-Net as objects of the class *Net*. They share common properties. For example, users also can perform the examining process on E-Net in the same way that they do on the OP-Net. Some functions may be overridden to fulfill the needs. For example, on OP-Net, when a user activates the exploring process and issues a query, a new E-Net will be created. However, on E-Net, if the user performs the exploring process and also issues a query, the resulting E-Net will replace the original E-Net.

At any time during the browsing process, the browsing process controller keeps track of the relevant frame instance set (FI_{OP}) with respect to the user's query (T_{OP}). All relevant objects of FIRs and their associated frame instances are also kept in the OP-Net. Assume that the current OP-Net is the one in Fig 3-14. The user activates the exploring process by issuing a query, say *(F1 AND T1)*. In applying the exploring process, users are restricted to constructing their queries by using those objects existing on the OP-Net. Therefore, the query needs not go through the topic interpretation process and can be directly rewritten into *(Folder(F1) AND Template(T1))*. The intersection of FI_{OP}(F1) and FI_{OP}(T1) forms the new frame instance set FI_E. After deriving the FI_E, the E-Net can be easily derived by doing the set intersection operation for each object.

The next step is to complete the graphical representation (G(V,E)) for the E-Net. The G(V,E) of E-Net is constructed from the G(V,E) of the current OP-Net. For each FIR in the OP-Net, its FI_E, which are those frame instances satisfying the predicate P_E, is computed. Basically, P_E is computed from P_O, P_{OP}, and the predicate derived from the query issued during the application of the exploring process. As explained in the topic refining process, the query issued by the user can be used to produce the predicate P_E. Then P_E can be represented by $P_O \wedge P_{OP} \wedge P_E$. For instance, let OP-Net be the current network which corresponds to a user's query *John*. At this moment, we know that John is a value. Therefore, we know that the topic of the OP-Net T = John and $P_{OP} = \mathrm{ContainsVal}(fi, John)$. Now, the user activates the exploring process and then the user issues a query *Letter*. Since the user is now performing at the phase of the exploring process, the object Letter must be on the OP-Net. Assume that there is a frame instance repository which corresponds to the folder *CIS* and we want to find out its predicate P_E. For the folder CIS, $P_O = \mathrm{IsInFolder}(fi, CIS)$. P_E can be computed as follows:

$$P_E = P_O \wedge P_{OP} \wedge P_E = \mathrm{IsInFolder}(fi,CIS) \wedge \mathrm{ContainsVal}(fi,John) \wedge \mathrm{OfType}(fi,Letter).$$



T_E = Folder(F1) AND
Template(T1)
P_E = IsInFolder(fi, F1) AND
IsOfTemplate(T1)
FI_E = {1, 2, 3}
FI(F1) = {1, 2, 3}
FI(F2) = {2, 3}
FI(T1) = {1, 2, 3}
FI(A1) = {1, 2, 3}
FI(A2) = {1, 2, 3}
FI(A3) = {1, 2, 3}
FI(V1) = {1, 2, 3}
FI(V2) = {1, 2, 3}
FI(V3) = {1, 2, 3}

Fig 3-14 The E-Net

At this particular state of the browsing process, the frame instance which is associated with the object CIS must meet the following conditions. First, it is deposited in the CIS folder during the filing process. Secondly, it must contain a value John. Finally, it has been classified as a letter. In this way, FI_E is computed for each object of the OP-Net. In the real world, for the object obj, $FI_E(obj)$ is computed from its $FI_{OP}(obj)$ as follows:

$$FI_E(obj) = FI_{OP}(obj) \bigcap FI(E)$$

For constructing the E-Net with respect to an active OP-Net, if $FI_E(obj) = \varnothing$, the object obj will not appear in the E-Net. Fig 3-14 depicts the resultant E-Net.

We now give the formal definition of the exploring network.

Definition 3-1: (Exploring Network)

An exploring network (E-Net) is a four-tuple, $E = [T_E, P_E, FI_E, G(V, E)]$, where:

1. T_E is a topic related to the context of the exploring process which can be recursively introduced using AND, OR and NOT operators;
2. P_E is a predicate related to the topic T_E;
3. $FI_E = \{fi | fi \text{ is a frame instance satisfing } P_E \wedge P_{OP}\}$; and
4. G(V, E) is a graph, where
   - Each node in V(G) is a frame instance repository
   - Each edge(i, j) between two repositories FIR(i) and FIR(j) represents that these two repositories have at least one common frame instance.

3.4.2 OP-Net and E-Net

Finally, we conclude this section by given the differences between the OP-Net and E-Net which are as follows:

1. The OP-Net and the E-Net are produced by the topic refining process and the exploring process, respectively.
2. For each OP-Net in the browsing session, it is always constructed by consulting the system catalog (object network).
3. For the E-Net in the exploring process, the first one is constructed from the current OP-Net and the succeeding E-Nets can be constructed from either the OP-Net or the previous E-Net.

3.5 Examining Process

With the functional capability of exploring the system knowledge, the *examining process controller* also has to provide the knowledge that cannot be expressed by the OP-Net or E-Net. For example, the examining process has access to the system catalog for exploring the relation between folders. The examining process controller keeps a set of pre-defined queries over the system catalog, which produces various views of the system catalog. For instance, during the course of the browsing, through the OP-Net or E-Net, the user may want to know the child folder of the folder CIS. Since the OP-Net or the E-Net does not carry the horizontal relations, the answer cannot be provided directly. Instead, the examining process controller will issue a formal query against the system catalog. We explore the request by heuristics and convert it into a set of functions, which provide the user with different views of the system catalog.

CHAPTER 4

THE SERVICE PROVIDERS AND THE STORAGE SYSTEM

In this chapter, we will present a system architecture of the browser for supporting the functionality discussed in the previous chapter. The major components of the system are shown in Fig 4-15 and divided from top to bottom into three layers.

4.1 The Controller Layer

The first layer contains interface controller, browsing controller, topic refining process controller, exploring process controller, and examining process controller. The interface controller consists of a graphic interface and a control unit. The graphic interface contains many physically graphical components: buttons, text fields, canvases, etc. When users interact with these components, events are generated. From the component where the

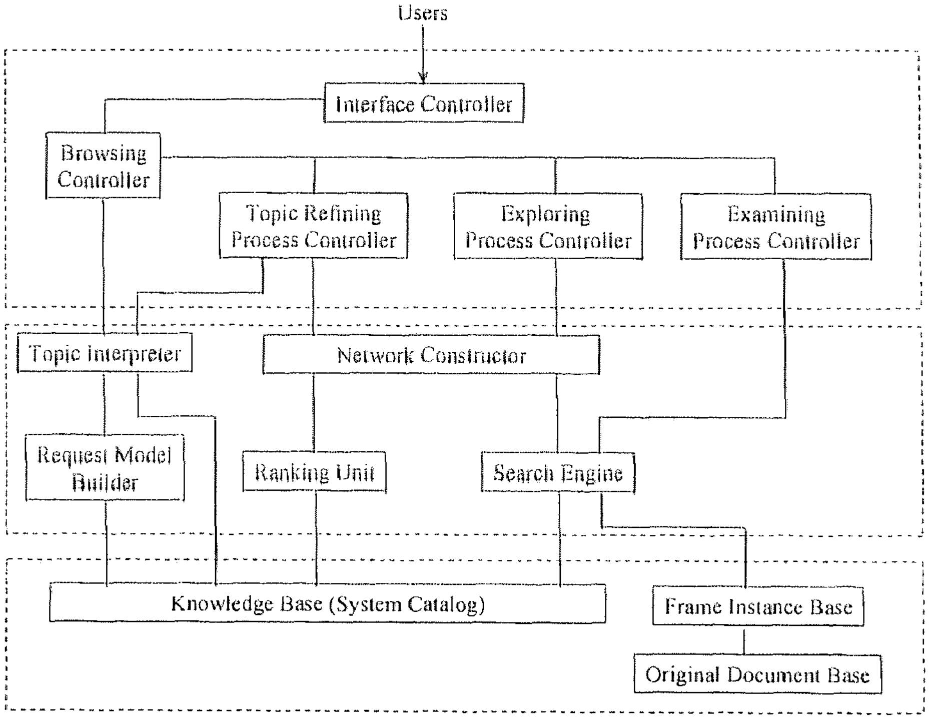


Fig 4-15 The System Architecture

event occurs, the control unit gathers two pieces of information: the name of the component and the action happened on the component. The responsibility of the control unit is to filter out those insignificant events and send the important message to the browsing process controller. For dispatching jobs, the browsing process controller analyzes the message received from the interface controller and decides the succeeded process to be taken according to the current browsing mode. In order to direct the message to the appropriate process controllers, the browsing controller keeps track of each browsing session and maintains the browsing history. The rest of the components in this layer are the browsing process controllers. These controllers receive requests from the browsing controller. There are some protocols between the browsing controller and the browsing process controllers. Each of the browsing process controllers has a set of functions that can be organized as a procedure based upon the request. Therefore upon the arrival of a request, the browsing process controller knows how to deal with it. These browsing process controllers process the request and generate additional events which are sent back to the browsing controller and wait there for processing. In the later sections of this chapter, we will take a closer look at these functions provided by these sub-process controllers.

4.2 The Service Provider Layer

The second level of the system architecture contains the topic interpreter, the request model builder, the network constructor, the ranking unit, and the search engine. These components are service providers because they provide services to the controllers at the first level.

Upon the arrival of a raw topic issued by users, the topic interpreter performs two tasks: the key-term replacement and the raw topic normalization, with the assistance of the system catalog. These two processes and their structures will be discussed in the later sections. The output of the topic interpreter will be sent to the request model builder for further processing.

The request model builder receives its input from the topic interpreter and it performs two tasks: the object identification process and the topic rewriting process. The request model builder also consults the system catalog while performing the tasks. The aim of the processes performed by the topic interpreter and the request model builder is searching for precise interpretation of users' information need.

Upon the arrival of a preprocessed topic, the network constructor builds an OP-Net or an E-Net depending on the browsing mode encountered. In the process of constructing the OP-Net (E-Net), the network constructor derives the frame instance set and searches for the relevant objects, with the assistance of the search engine.

In TEXPROS, a ranking model which ranks the frame instances related to a topic is developed. After each frame instance receives a rank, the frame instance repositories are also ranked. The ranking of the frame instance repositories can help users decide which portion of the network should be examined first. The ranking unit conducts the ranking process. The input of the ranking unit is the OP-Net or the E-Net delivered by the network constructor. During the ranking process, the ranking unit also consults the system catalog. We will give more details of the ranking model in the following sections.

The last component in this layer is the search engine. The input of the search engine comes from the network constructor or the examining process controller. The network constructor requests the search engine to provide the OP-Net (E-Net) or a particular frame instance repository with frame instance sets. The requests from the examining process are more complicated. Some requests can be directly solved by consulting the frame instance base. For example, at the examining process, the user wants to see the content of a frame instance in a specific frame instance repository. For another type of the requests, the search engine asks help from the object network. Consider the previous example. In the examining process, the user can issue a request for finding sub-folders of a specific folder in the OP-Net (or E-Net). Since the OP-Net does not contain the folder relationship of the folder organization, the search engine has to consult the object network for gathering the folder relationships. This is the reason although the horizontal relations are removed from the OP-Net, the user will not be aware of this removal because they are still in the object network.

4.3 The Storage System Layer

The third level contains three components: the knowledge base, the frame instance base and the original document base. In addition to supporting the browsing process, the components at this level also provide the service to other processes such as document classification, extraction, and filing process. In other words, these are the core components managed by the system.

The knowledge base gives the system the ability to resolve users' information needs in the topic interpretation process. The details of the knowledge base will be discussed in Chapter 6.

Each incoming document is stored in the *original document base* and its address on the physical storage device will be recorded as the document identifier. Based on the document identifier, effective index mechanisms (such as hashing technique) can be used to speed up the performance of the retrieval. However, the user seldom retrieves the original document because the important information has been extracted during the extraction process to form the frame instance.

The frame instances are stored in the *frame instance base*. The basic goal of the frame instance base is to find the needed frame instance effectively. If the retrieval of the frame instances is predictable, then grouping these frame instances together can achieve better performance. Since there is no way to predict the sequence of the retrieval, there won't be a unique grouping (clustering) that can satisfy all the users. The heuristics can solve this problem to some extent. That means, although the sequence for every user is unknown, it would help if the perception of the document or the frame instance of most users can be derived statistically. The dual model adopted by TEXPROS captures the natural perception of the document. When people receive a new document, most of them identify the type of the document immediately. If the document is important, it will be stored in the storage (such as the folder) for later reference. In our browser, two indexes (SYSFMAP and SYSTMAP) corresponding to the dual model are created to improve the performance of retrieval (which will be discussed in the next chapter). Two index maps in the system catalog, namely The SYSFMAP and the SYSFMAP, are used to help the search engine to retrieve the frame instances from the frame instance base.

In the frame instance base, the frame instances are clustered by their document type, instead of the deposited folders. The reasons are as follows:

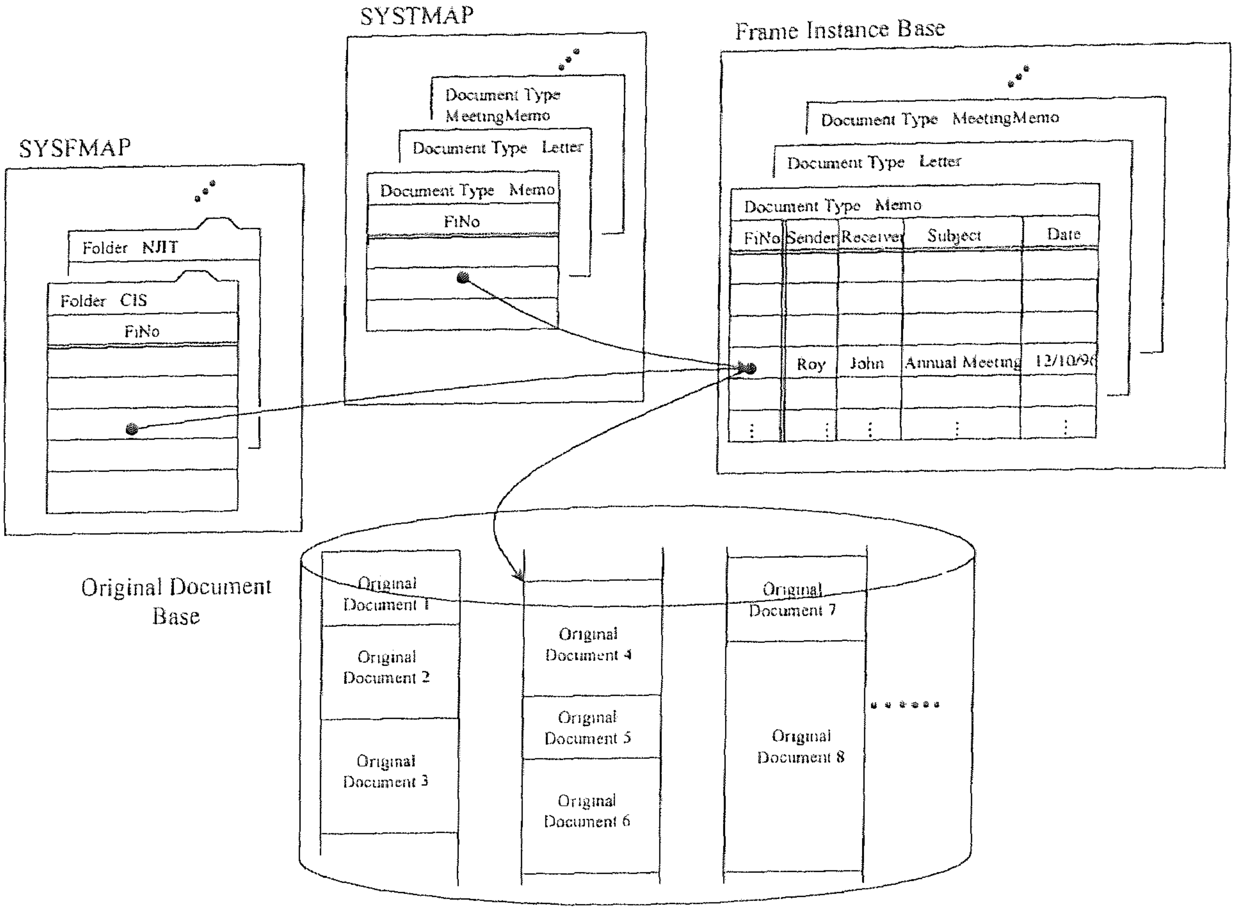


Fig 4-16 The storage system

1. The frame instances of the same document type share a common structure. Since the structure is of fixed length, it can be easily organized for better performance.
2. Since the frame instance can be deposited into more than one folder, organizing the frame instances based on the folders may cause redundancy of the occurrence of the same frame instance.

Fig 4-16 shows the storage system used by the browser. For simplicity, only one frame instance is shown in this figure. Each frame instance in the frame instance base will be assigned a unique integer Identifier (FiNo). This identifier is computed from the physical address of the original document in the original document base. Since the frame instances are used throughout the processes (except for the examining process) for the performance concern, there is no direct mapping between the folder and the original document base.

4.4 The Local Memory

Along with each browsing process session, there are lots of intermediate results produced by the components in the three layers. These intermediate results are stored in shared memory called the local memory because it is used internally by the system and hidden from users. The usage of the local memory and its interaction with other components will be explained in the next chapter.

CHAPTER 5

SYSTEM ARCHITECTURE

In this chapter, the browser which is the control unit of the browsing process is described. The browsing process monitors four major processes in any browsing sessions, namely the topic interpretation process, the topic refining process, the exploring process and the examining process. The topic interpreter transforms a user query into a normalized topic (which is in disjunctive normal form), in which the values of the standard term (called key terms) are used. For each normalized topic, more information, such as the properties of a key term (e.g., the key term could be a value type or a folder name) can be obtained through the request model builder which rewrites the topic to include this additional information. In supporting the refining, exploring or examining process, the OP-Net and the E-Net for a given topic are needed. Given this normalized topic, the network constructor creates and OP-Net or an E-Net, using the search engine for finding all the frame instances from the frame instance base, which are pertinent to the given topic. If necessarily, the search engine consults with the knowledge base for additional information.

5.1 Topic Interpreter

The topic interpretation process of the topic interpreter consists of the key term replacement and the raw topic transformation. Upon the arrival of a raw topic (which is entered by the user), the topic interpreter receives the raw topic from the browsing controller, and then transforms it into a normalized topic.

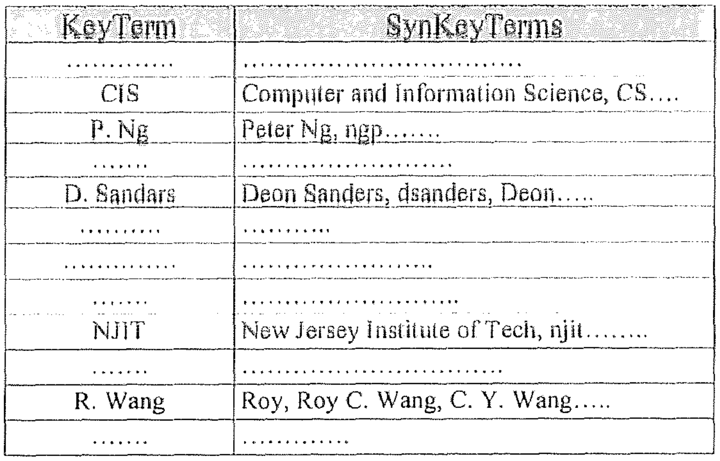
The topic interpreter has two major tasks, namely the key term replacement and the raw topic transformation.

1. Key term replacement

   Normally, a user begins a browsing session by entering a vague query, which comprises terms and logical operators such as AND, OR, and NOT. This vague query is called a *raw topic*. The terms appearing in the raw topic are uncontrolled in the sense that these terms may not be the standard terms used by the system. For consistency, the uncontrolled terms are replaced by the key terms which are used internally by the system. To identify the key terms for the uncontrolled terms, the topic interpreter consults the system catalog. The thesaurus in the system catalog provides the ability of resolving users' uncontrolled terms. Currently, the thesaurus which consists of two system frame templates, SYSSYNONYMS and SYSNARROWER, is part of the system catalog. Fig 5- 17 is an example of the SYSNARROWER frame template.

2. Raw topic transformation

   After uncontrolled terms have been replaced by the key terms, an intermediate



KeyTerm	SynKeyTerms
....	....
CIS	Computer and Information Science, CS....
P. Ng	Peter Ng, ngp.......
....	....
D. Sandars	Deon Sanders, dsanders, Deon.....
....	....
....	....
....	....
NJIT	New Jersey Institute of Tech, njit........
....	....
R. Wang	Roy, Roy C. Wang, C. Y. Wang.....
....	....

Fig 5- 17 System Template SYSSYNONYMS query is obtained. The next step is to transform this intermediate query into a topic, which is in disjunctive normal form and will be used by the succeeding processes.

To conclude this section, the following example is given to explain these two processes. Given a raw topic **Q1: (*dsanders* OR *png*) AND *CIS***, the topic interpreter first identifies three uncontrolled terms, *dsanders*, *png*, and *CIS*. After consulting the system catalog, these three terms are replaced by the key terms *D. Sanders*, *P. Ng* and *CIS* respectively. Therefore, we obtain an intermediate query **Q2: (*D. Sanders* OR *P. Ng*) AND *CIS***. The next step is to transform **Q2** into a topic of disjunctive normal form. During this process, after checking the correction of the syntax, **Q2** is transformed into a normalized topic according to the priority of the operators and the parentheses. The topic **Q3: (*D. Sanders* AND *CIS*) OR (*P. Ng* AND *CIS*)** is then obtained. The data flow of the process of topic interpretation is given in Fig 5-18. Dot lines, double dot lines, arrows, rectangles, and round rectangles are used to represent the data flows, multiple data flows, the directions of the flow, the components, and the processes inside the components, respectively.

5.2 Request Model Builder

Based on the normalized topic produced by the topic interpreter, the request model builder [2] is used to gather, construct and disseminate detailed representation of the user's information needs. The output is a rewritten query, which reveals more information.

As in most the full-text search information systems, queries that contain only key terms fail to capture users' information needs, such as the types of key terms. The request model builder consists of two processes, namely, the object identification process and the query rewritten process.

1. Object identification process

   Given a normalized topic, the object identification process searches through the system catalog to find out the type for each term in the topic. Applying this process, all the key terms in the topic are transformed into objects which are represented in a format *ObjType*(*ObjName*) where ObjName is a key term and ObjType is key term type. In this way, the full-text system becomes a special case of our system in the sense that it contains only one object type *VALUE*. The output of this process will be a new topic containing the key terms along with their type of key tems (and is expressed in terms of *ObjType*(*ObjName*)), and the logic operators. When a key term is related to more than one type (as in our later example), the operator OR will be used between types, e.g. ObjType$_1$(ObjName) OR ObjType$_2$(ObjName) OR ....

2. Query rewriting process

   After the end of the object identification process, the topic could no longer be in disjunctive normal form. The query rewriting process is applied for normalized the topic and outputs the final topic which is in disjunctive normal form.

Consider the example from the previous section. In **Q3**, there are three key terms, namely *D. Sanders*, *P. Ng*, and *CIS*. After consulting the system catalog, the object types of these terms can be identified. Assume that *D. Sanders* and *P. Ng* are values and *CIS*

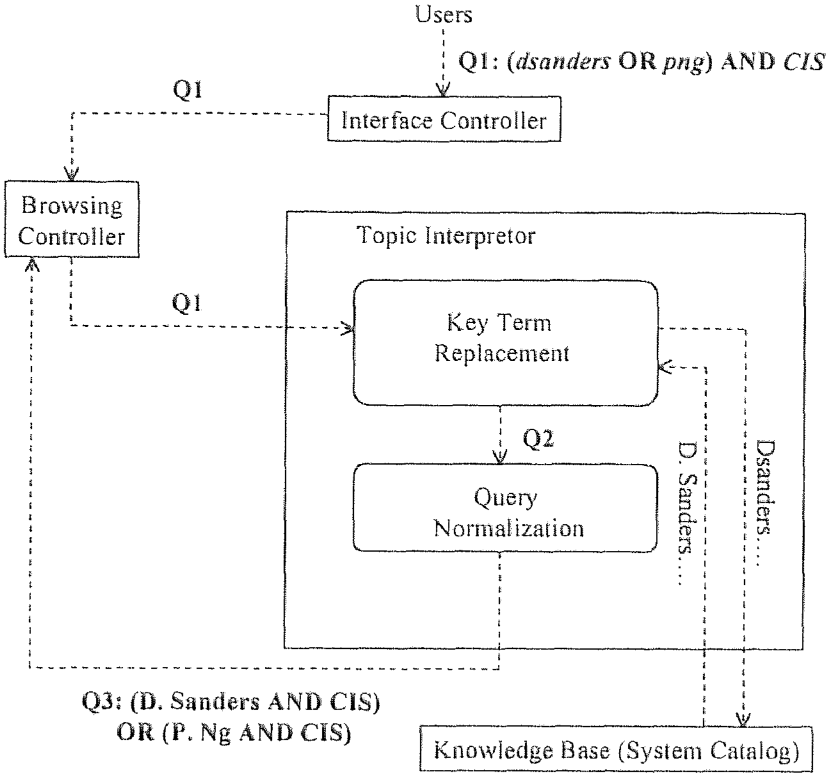


Fig 5-18 The Data flow of the Topic Interpretation Process

can be a value and also a folder. These terms will be rewritten as *VALUE*(*D. Sanders*), *VALUE*(*P. Ng*), *VALUE*(*CIS*), and *FOLDER*(*CIS*). **Q3** is then converted into

Q4: (VALUE(*D. Sanders*) AND (VALUE(*CIS*) OR FOLDER(*CIS*)) OR

(VALUE(*P. Ng*) AND (VALUE(*CIS*) OR FOLDER(*CIS*)).

Since **Q4** now is not in disjunctive normal form, it will be normalized to

Q5: (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*)).

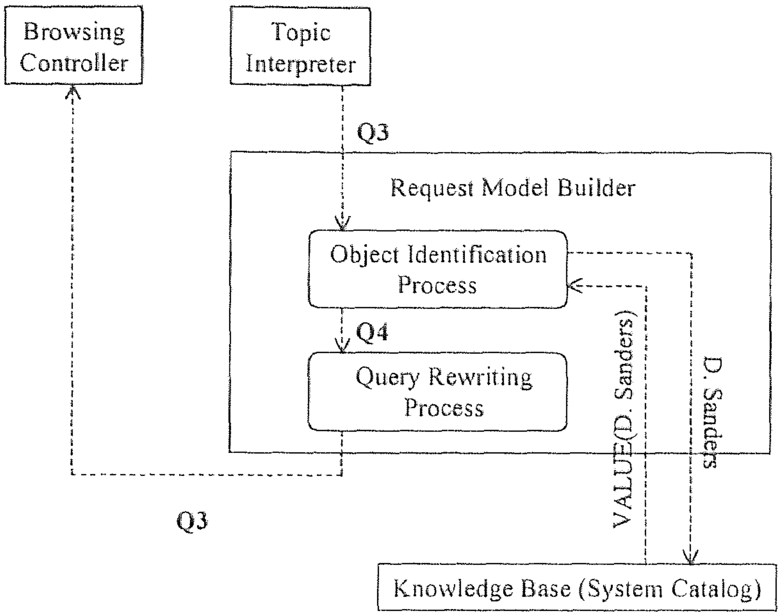


Fig 5-19 The Data Flow of the Request Model Builder

The final topic **Q5** is returned to the Browsing Controller for later use. The data flow of these processes for the request model builder is shown in Fig 5-19.

5.3 Network Constructor

Given a normalized topic produced by the topic interpreter and the request model builder, the network constructor creates an OP-Net or an E-Net.

The network constructor has two main processes.

1. Deriving the frame instance set

According to the definition of the OP-Net (E-Net), each OP-Net (E-Net) is corresponding to a topic T_{OP} and an associated predicate P_{OP} is derived from T_{OP}. The P_{OP} is a predicate that specifies conditions for all the frame instances. For any document related to the T_{OP}, its corresponding frame instance fi must satisfy the predicate P_{OP}. The first task of the network constructor is to find the set FI_{OP} of those qualified frame instances.

2. Finding the relevant object

In the initial state, which is the state before users bring up the browsing process session, conceptually the object network is considered to be the *virtual* OP-Net. When a user issues a new query or modifies the original query, the network constructor modifies the original network to represent the query. As the user changes the topic, the predicate changes, and the criteria of selecting relevant frame instances is also changed. Therefore, the modified network captures the changes, such as adding or removing frame instances from the contents of nodes in the network, or adding or removing nodes from the network.

According to the definition of the frame instance repository, given a frame instance repository FIR(f) of an object f, $FI_{OP}(f)$ represents the frame instance set containing all the frame instances associated with the frame instance repository of f that satisfy P_{OP}. The frame instance set $FI_{OP}(f)$ associated with each frame instance repository FIR(f) will be changed dynamically throughout the browsing session. Objects can be classified into two categories, namely relevant and irrelevant objects.

Definition 5-1: (Relevant and Irrelevant Objects)

Given a frame instance repository FIR(f) where f is an object, if $FI_{OP}(f) = \varnothing$, then f is an irrelevant object. Otherwise, it is a relevant object.

Definition 5-2: (Active and Passive Objects)

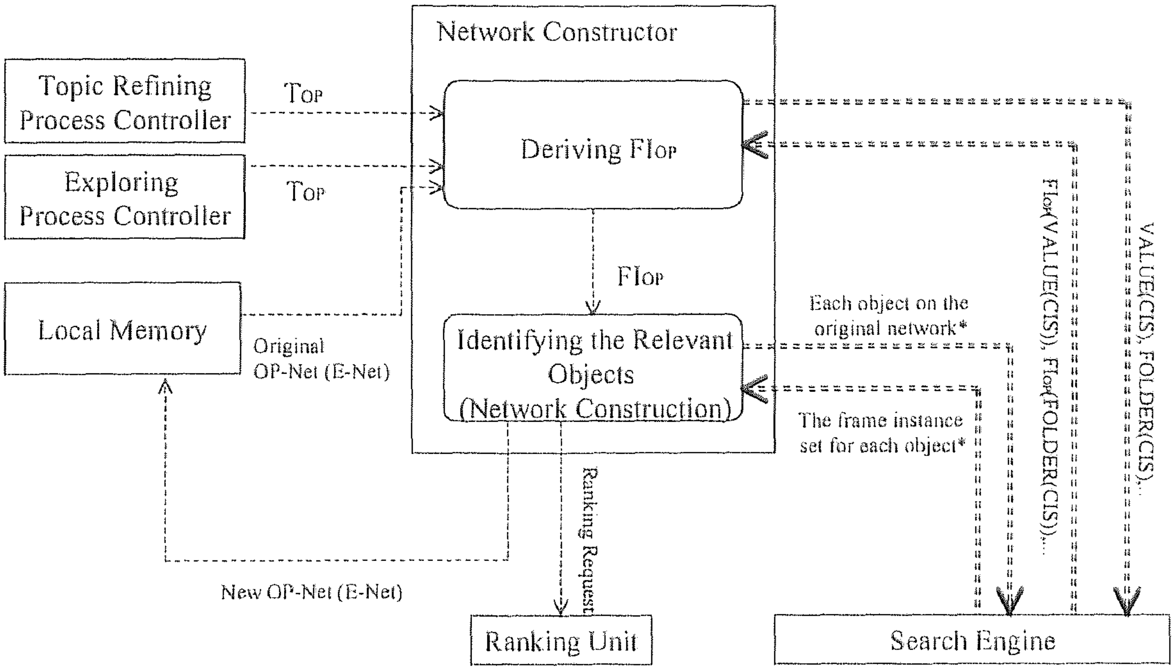
Given a frame instance repository FIR(f), where f is an object, if f is a relevant object and appears in a topic T_{OP}, then f is an active object. Otherwise f is a passive object.

The frame instance repository of an irrelevant object will be removed from the original network. The active objects are those identified by the request model builder. For example, *D. Sanders*, *P. Ng*, and *CIS* in the earlier example are active objects. These objects are important because they set up the conditions for qualifying frame instances; and we compute the FI_{OP} or FI_E based on these active objects. The objects of the original OP-Net containing frame instances in FI_{OP} or FI_E while are not active will be considered as passive objects. Any remaining objects which do not contain any frame instance in FI_{OP} or FI_E are considered as irrelevant objects. Intuitively, the OP-Net construction is similar to a graph traversal process in which every object of the original network is examined to check whether it is a relevant object.

During the network construction, the network constructor requires a search engine for finding the frame instance set for a final topic and the frame instance repositories for each object in the topic. Since the OP-Net (E-Net) serves as the underlying structure, the network will be stored in the public local memory which allows it to be accessed by other processes. Fig 5-20 shows the data flow of the network constructor.

5.4 Search Engine

The search engine serves as a general interface of accessing the components in the third layer. Since the components in the third layer count on the physical implementation structure, the components that are not in this layer can only see the public methods provided by this layer.


Network Constructor
Topic Refining Process Controller
Top
Exploring Process Controller
Top
Deriving Flop
Flop
Local Memory
Original OP-Net (E-Net)
Identifying the Relevant Objects (Network Construction)
Each object on the original network*
The frame instance set for each object*
Flop(VALUE(CIS)), Flop(FOLDER(CIS))...
VALUE(CIS), FOLDER(CIS)...
Ranking Request
New OP-Net (E-Net)
Ranking Unit
Search Engine

* This happens only when constructing the OP-Net.

Fig 5-20 The Data Flow of the Network Constructor

When the search engine receives a request from the network constructor or the examining process controller, a procedure, which utilizes the primitive methods provided by the storage system, achieves the request. The outputs of the search engine are varied depending on the type of the requests.

Based on the components involved, the requests can be divided into two types. For the first type of the requests, the search engine has to consult the knowledge base. For the second type of the requests, it needs the services provided by the frame instance base and the original document base.

5.4.1 First Type Request

In Chapter 2, we described the transformation of an object network into OP-Net, by removing the horizontal relationships among the objects of the same object type from the object network. The information of the folder organization and the document type hierarchy is no longer in the transformed network. The removal of these relationships does not reduce the power of responding to requests. Its explanation and the mechanism of how it works will be given fully in the later sections of this chapter. The search engine is the only component that supports the examining process and has access to the storage system. Therefore, this is the appropriate unit to support those horizontal relationships which are removed during the network transformation. Consider the following scenario as an example. Assume that, during the examining process, a user is interested in finding sub-folders of a particular folder CIS. This function will be integrated in the user interface and an intuitive name of the function is given so that users can easily understand the meaning of this function. When a user activates this function, the necessary information will be directed to the search engine. The search engine is able to find out that this is a first type request which requires information from the knowledge base. The search engine then calls an internal procedure that makes a request to the system catalog in the knowledge base and derives the answer from the system template SYSFOLDER.

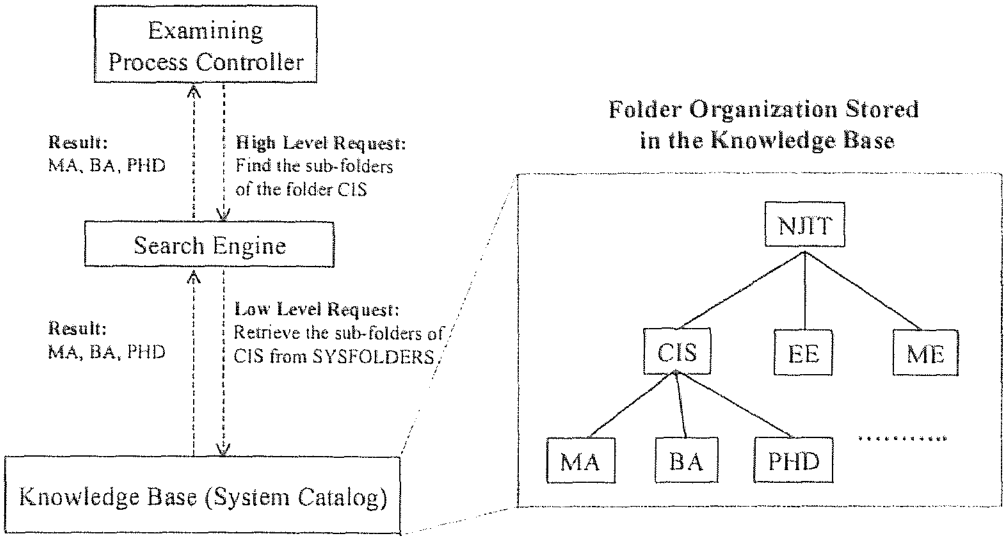

Examining Process Controller
Result:
MA, BA, PHD
High Level Request:
Find the sub-folders of the folder CIS
Search Engine
Result:
MA, BA, PHD
Low Level Request:
Retrieve the sub-folders of CIS from SYSFOLDERS
Knowledge Base (System Catalog)
Folder Organization Stored in the Knowledge Base
NJIT
CIS
EE
ME
MA
BA
PHD
....

Fig 5-21 The Data Flow of the Search Engine

The Fig 5-21 depicts an example of a folder organization stored in the knowledge base and the data flow of the process for handle the finding of the subfolders of the folder CIS.

5.4.2 Second Type Request

The second type requests mainly inquire the frame instances, the original documents, or their associated information. All the different cases of the second type request can be covered by using some primitive public methods provided by the frame instance base and the knowledge base. The functionality of the primitive public methods provided by the frame instance base are as follows:

1. Given a frame template, the frame instance base returns the frame instance set, and

2. Given a frame instance ID, the frame instance base returns the frame instance.

The frame instance base will be explained in the next section. Similarly, the knowledge base also provides a set of primitive methods for the public use.

The search engine has to provide some high level functions by utilizing those methods that provided by the third layer. In order to explain how this mechanism works, consider an example of using the function FindFiNoSet provided by the search engine. The functionality of the function can be described as follows. The FindFiNoSet takes an object as the argument and returns a set of associated frame instances based on the association defined in Chapter 1. Let us use an object of the ATTRIBUTE type as an argument. Upon receiving a request FindFiNoSet((ATTRIBUTE, SENDER)), the search engine has to come up with a procedure to find the set of the frame instances. The procedure is described as follows:

1. Consult the system catalog to find all the frame templates that contain the attribute SENDER. Assume that the two document types, namely the Memo and the E-Mail, contain the attribute SENDER.
2. For each document type derived in the step 1, the search engine issues a request to the frame instance base and returns the result of finding all the frame instances of the types containing the attribute SENDER.

In this procedure, step one will use a method provided by the system catalog to find the templates. In step two, the search engine will use the method provided by the frame instance base to find the frame instances for each document type containing the attribute SENDER. In this procedure, the search engine provides the logic to call these methods and organizes the returned results. The data flow for this example is given in Fig 5-22. Since there are too many cases of the second type requests, we cannot describe them all.

From the system architecture described in Chapter 4, obviously the second type requests can be directed from either the network constructor or the examining process controller. The search engine in our system provides an interface for communicating with the core components namely, the knowledge base and the frame instance base. Although the search engine is a small part, it is a crucial component for enhancing the performance of the system. One of the basic functions of the search engine is to find the related frame instances for a given object. Based on this basic function, the search engine can support the following processes.

1. Find the frame instance set for the OP-Net or E-Net.
2. Find the associated frame instance set for each frame instance repository on the OP-Net or E-Net.

Another important function of the search engine is to support the examining process. Users and the system use the search engine to get information contained in the system catalog. This is the reason that the system can provide the information about the horizontal relations, even though they are not included in the OP-Net.

5.5 Frame Instance Base and Original Document Base

Through the search engine, the frame instance base gets two types of requests from the network constructor or the examining process controller. The first one is to return a set of frame instance of a given frame template type and the other one is to return a frame instance if the frame instance id is given.

After the classification and the extraction processes, a frame instance of a given original document is formed. Each frame instance will be assigned a unique ID, FiNO, and stored in the frame instance base. The frame instances are organized by their document types (frame templates). That is, all the frame instances of the same document type will be clustered. Therefore, the search engine needs to identify the document type before issuing the request to the frame instance base. For speeding up this process, two system templates which will serve as the indexes are created in the system catalog. One is created over frame instances and folders (SYSFMAP). The other one is created over frame instances and frame templates (SYSTMAP).

5.6 Browser

Browser is the control unit of the browsing process. It monitors browsing sessions

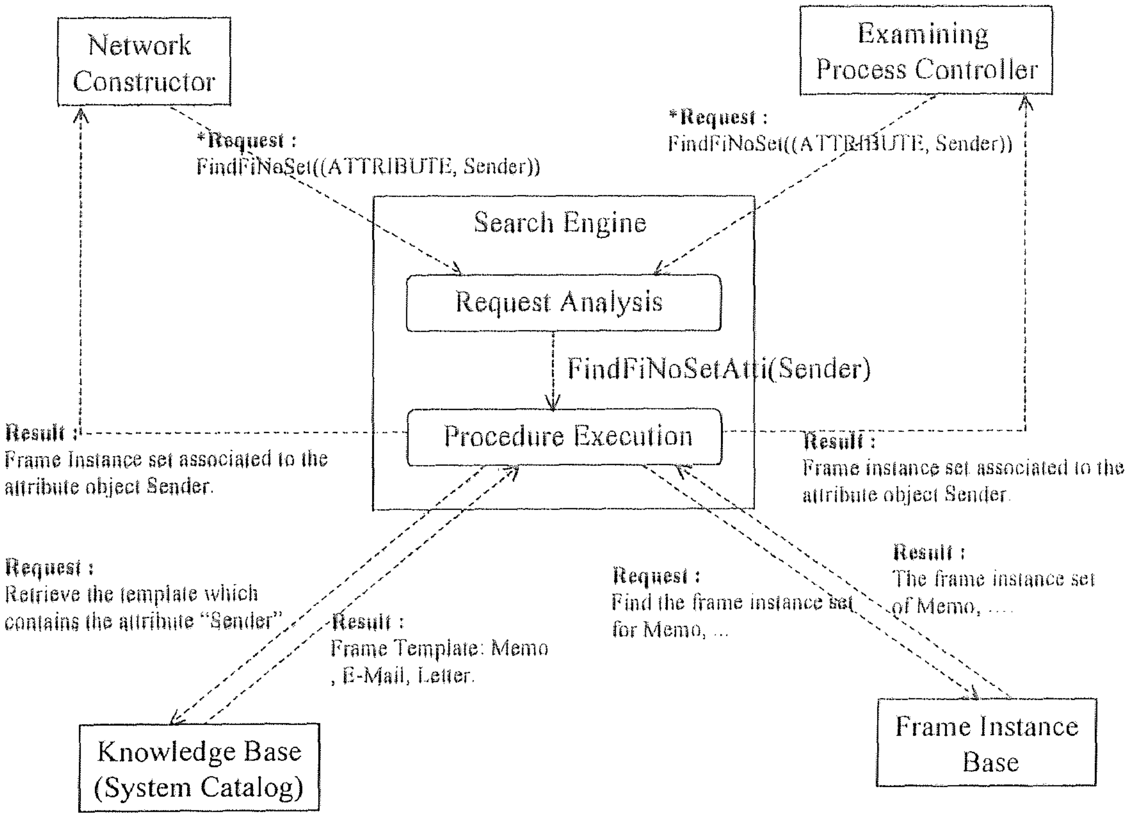

Network Constructor
Examining Process Controller
*Request :
FindFiNoSet((ATTRIBUTE, Sender))
*Request :
FindFiNoSet((ATTRIBUTE, Sender))
Search Engine
Request Analysis
FindFiNoSetAtti(Sender)
Procedure Execution
Result :
Frame Instance set associated to the attribute object Sender.
Result :
Frame instance set associated to the attribute object Sender.
Request :
Retrieve the template which contains the attribute "Sender"
Result :
Frame Template: Memo , E-Mail, Letter.
Request :
Find the frame instance set for Memo, ...
Result :
The frame instance set of Memo, ...
Knowledge Base (System Catalog)
Frame Instance Base

Fig 5-22 The data flow of the second type request

between the system and the user. The main function of the browser is to extensively utilize the functions provided by the other components to fulfill the user's needs. The browser consists of five parts: the browsing controller, three sub-process controllers, and the interface controller.

5.6.1 Browsing Controller

The browsing controller is like a task dispatcher. It receives the request from the interface controller, then analyzes and assigns it to a proper sub-process controller to continue the process of finding the answer of the user's request. The sub-process controllers are the topic refining process controller (TRPC), the exploring process controller (EXPC), and the examining process controller (EPC). Browsing process controller monitors three major processes in the browsing session, namely, the topic refining process, the exploring process, and the examining process. Therefore the browsing session proceeds by switching back and forth between these three process modes. The precise task descriptions assist the browsing process controller to dispatch the task correctly. Therefore, the responsibility of the browsing process controller is to conduct the task identification and dispatching process to identify the task description for each user's request and dispatch these tasks to the proper controller (The details of dispatching tasks will be given in the next sections). The task identification and dispatching process is described in next section.

5.6.1.1 Task Identification and Dispatching Process: The objective of this process is to find the task ID (TaskID) and the controller which can achieve that task. This browsing process controller receives the user's request and needed information such as the name of the function selected by the user during the browsing process, the browsing process mode changed by the selected function, and possibly the name of the object in the OP-Net or E-Net, from the interface controller. The browsing process controller keeps this important information in the local memory. Upon receiving the new request from the interface controller, if necessary, the browsing process controller updates the information in the local memory. One thing needs to be pointed out is that in the graphical interface, intuitive names are used for the provided functions. From the implementation point of view, each function name is considered as a tag of the physically graphical components. For instance, there is a function "Find the child folders" which can find the child folders for a folder object. The name "Find the child folders" can be considered as a tag for the graphical component (the menu item). Each graphical component has its unique ID. When a user clicks on this menu item, instead of sending "Find the child folders", the interface controller sends the ID of the menu item to the browsing process controller.

In the local memory, there is a *task description table*. Compared with the information dynamically received from the interface controller, this map is static. By utilizing the dynamic information and the task description table, the browsing process controller is able to identify the task ID corresponding to a user's request and the suitable controller. The details of the task description table and how it works will be explained shortly in an example.

5.6.1.2 Object-Originated and Non-Object-Originated Task: There are two kinds of tasks to be achieved, namely object-originated and non-object-originated task. In

TEXPROS, the requests of users are made through the use of the graphical interface. The graphical interface consists of graphical components for interacting with the user and accepting user's input. The difference between two kinds of tasks depends on the graphical components in which the users' requests are made. Two examples are given in the next sections to explain the difference between these two types of tasks.

Object-Originated Task Fig 5-23 is the data flow of processing an object-originated task. Assume that the user examines the OP-Net displayed in the graphical interface and wants to find out the child folders of a folder CIS. In this case, the user will click on the node representing the folder CIS. The node responses with displaying a menu consisting of a list of provided functions. The request is made and sent to the system as the user selects one of the functions from the displayed menu. In this example, the user selects a predefined functions associated with the object (e.g. FOLDER(CIS)). Therefore, it is an object-originated task. The browsing process controller receives the request and information from the interface controller. To identify object-originated task, the browsing process controller has to know the current browsing mode, the object in which the request is originated, the function that the user asks to perform and its corresponding graphical component ID. In Fig 5-23, the browsing mode is the examining mode, the object is the CIS folder, and the function to be performed is FindChildFolder and the ID of it's corresponding graphical component is EX_Menu_Item2. The task identification and dispatching process takes these information and look up in the task description table. The returned result of this process are the TaskID (in our example is 5) and the controller (EXPC) which is responsible for achieving this task.

Non-Object-Originated Task An example of the non-object-originated task is the topic refining process. When a user wants to refine the current topic, s/he needs to find the text field in the graphical interface and to enter in the modified topic. In this case, the request has nothing to do with any particular object in the OP-Net or E-Net. Therefore, this is an non-object-originated task.

Fig 5-24 is an example of the topic refining process. The user inputs the new topic in the text field provided by the graphical interface. The graphical component ID (TE_TextField1), the new topic (CIS AND Roy), and the switched browsing mode (topic refining mode) are sent to browsing process controller. Upon receiving the needed information, the browsing controller conducts the tasking identification and dispatching process to look up the task description table. The result which consists of the TaskID (2) and controller (TRPC) will be returned to the browsing controller.

5.6.2 Topic Refining Process Controller

The topic refining process controller (TRPC) conducts and controls the topic refining process. The whole topic refining process can be initiated by the interface controller. Upon the arrival of a user's query, the interface controller sends it to the browsing controller. The browsing controller performs two tasks. First it checks the current browsing mode. There could be several cases. Here we assume that the current browsing mode is in the topic refining mode. We will explain how the browsing controller decides which mode should be in the later section. After knowing the current browsing mode, the browser directs the query to go through the topic interpretation process. The output of the topic interpretation process is stored in the local memory for later use.

From the user's point of view, after providing a query (topic), the system will display an OP-Net. Throughout the browsing session, the user could perform some operations on this network. From the TRPC's point of view, in order to provide an OP-Net, it needs a procedure for constructing this network. After the OP-Net is constructed, the TRPC keeps track of all the processes and operations performed on the OP-Net. In brief, TRPC has two main tasks: namely, construct the OP-Net and monitor the topic refining process.

Constructing an OP-Net is straightforward. From the discussion in section 5.3, once the necessary information is provided in the local memory, all TRPC needs to do is to issue a command to network constructor and the network constructor will take care of the rest.

Let us take a close look at the second task of TRPC. The goal of the topic refining process is to provide an environment with a set of primitive functions in which a user can continue issuing new topics or modifying the current topic. By using these functions, the topic refining process can provide some procedures, which are meaningful to the users.

The topic refining process provides the following functions:

- **NewTopic( )** This function is called when the user issued a new raw topic through the interface controller. The topic interpretation process generates the normalized query and related information which are stored in the local memory. In this case, the TRPC notifies the network constructor to construct an OP-Net by examining the normalized topic produced by the topic interpretation process and consulting the system catalog.
- **NEW(Object)** This function takes an object as its argument and constructs a new

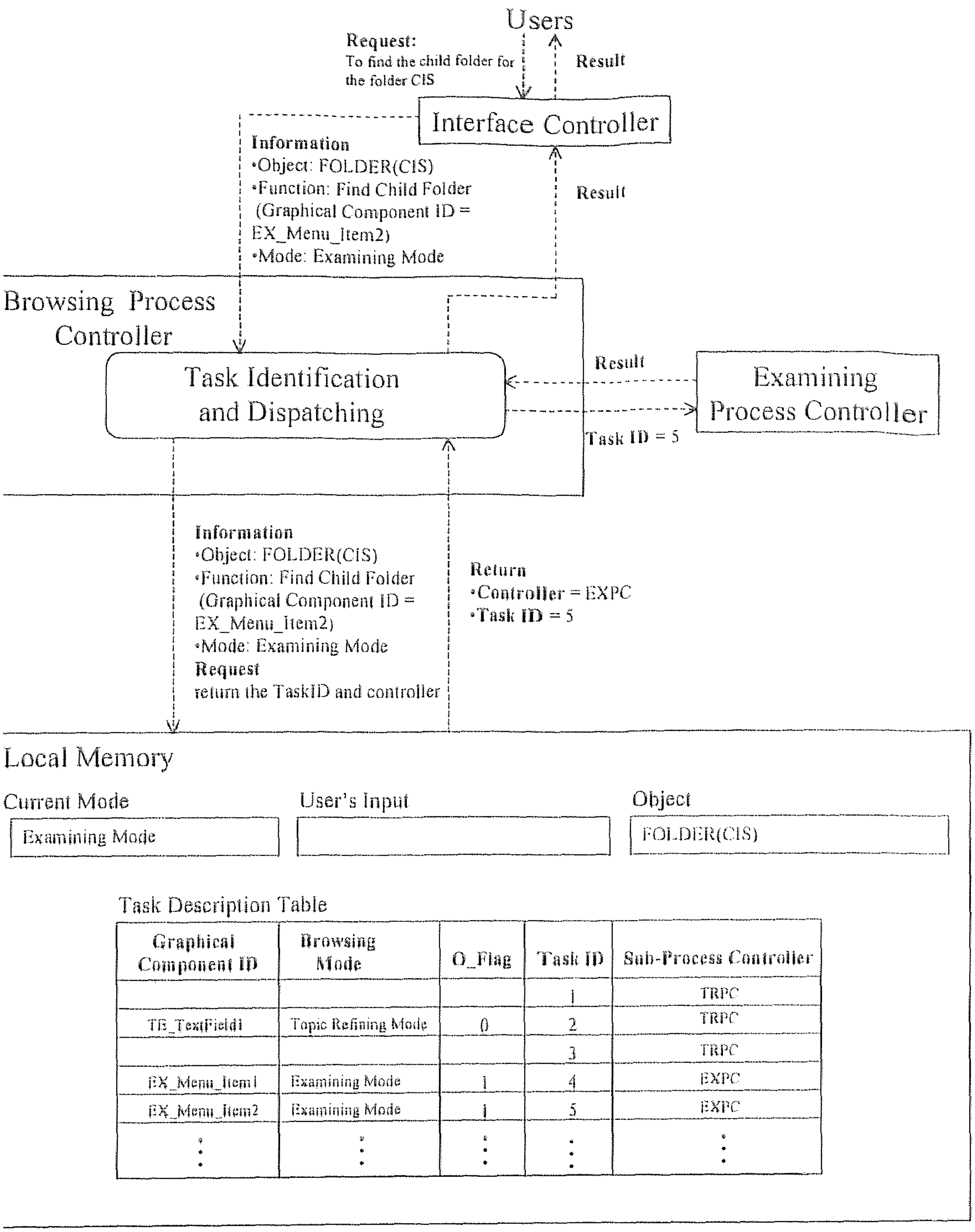
OP-Net. Intuitively, this function is activated when a user issues a query which contains only a term, which is represented by the argument *Object*. Therefore, TRPC schedules for constructing an OP-Net based on the *Object*. The difference between this function and NewTopic( ) arises because of the way they have been activated. In the interface controller, the input of the NewTopic( ) is derived from a text field provided by the interface. However, the argument Object of NEW( ) is derived by clicking on a node of the current OP-Net. Since the input for the NewTopic( ) is a raw topic, it needs to go though the topic interpretation process. The input for NEW( ) is a controlled object in the OP-Net, which does not need to go though the topic interpretation process.

- **AND(Object)** This function is activated when the user wants to modify the current topic from the existing OP-Net. There are two cases according to the inputs of the function.

  1. **The Object is in the current OP-Net** The user inputs this argument by clicking on a node on the current OP-Net. In this case, the argument is a controlled object and needs not to go through the topic interpretation process.

  2. **The Object is not in the current OP-Net** The user enters this argument by typing it in a text field provided by the interface controller. In this case, the argument is an uncontrolled object and needs to go through the topic interpretation process.

  Normally, this function will reduce the size of the current OP-Net by removing nodes or sub-networks, which are not qualified by the AND condition, from the OP-Net.


Users
Request:
To find the child folder for the folder CIS
Result
Interface Controller
Information
•Object: FOLDER(CIS)
•Function: Find Child Folder (Graphical Component ID = EX_Menu_Item2)
•Mode: Examining Mode
Result
Browsing Process Controller
Task Identification and Dispatching
Result
Examining Process Controller
Task ID = 5
Information
•Object: FOLDER(CIS)
•Function: Find Child Folder (Graphical Component ID = EX_Menu_Item2)
•Mode: Examining Mode
Request
return the TaskID and controller
Return
•Controller = EXPC
•Task ID = 5
Local Memory
Current Mode
Examining Mode
User's Input
Object
FOLDER(CIS)
Task Description Table

Graphical Component ID	Browsing Mode	O_Flag	Task ID	Sub-Process Controller
			1	TRPC
TE_TextField1	Topic Refining Mode	0	2	TRPC
			3	TRPC
EX_Menu_Item1	Examining Mode	1	4	EXPC
EX_Menu_Item2	Examining Mode	1	5	EXPC
⋮	⋮	⋮	⋮	⋮

Fig 5-23 The data flow of an object-originated task

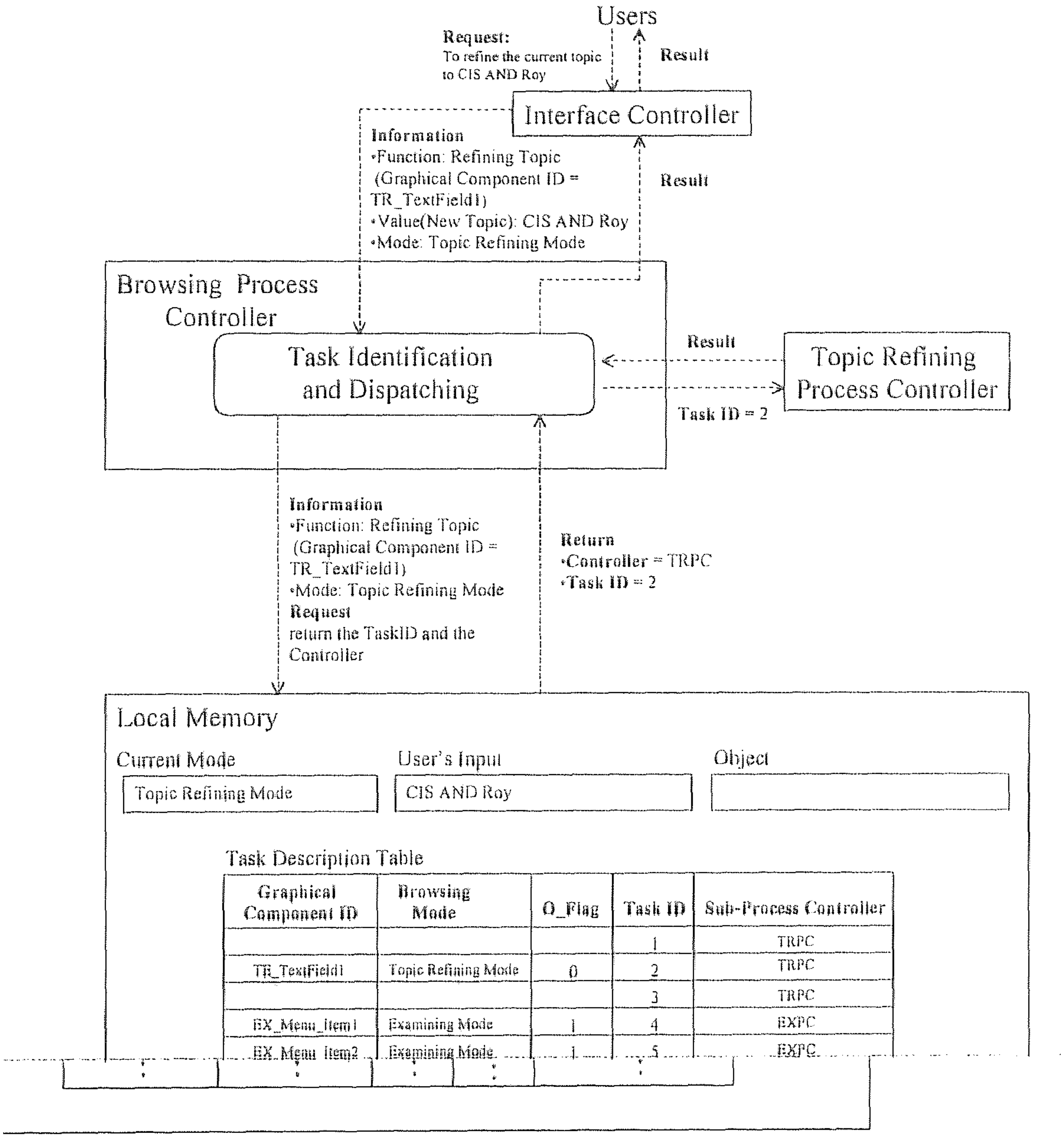
- **OR(Object)** This is another function for modifying the current topic. Two cases will arise which are the same as the cases for the **AND( )** function. Normally, this function will enlarge the size of the current OP-Net by adding nodes or sub-networks, which are satisfied by the OR condition, into the OP-Net.
- **NOT(Object)** When this function is activated, the frame instance set associated with the argument Object will be excluded from the frame instance set associated with the current OP-Net's FI_{OP}. If the argument Object is in the current OP-Net, then it will become irrelevant object in the new OP-Net. When this function is used with other functions, its effect is equivalent to **AND NOT**.
- **REDO( )** This function provides a way to go back to the original OP-Net. To support this function, the TPRC needs to maintain a modification history during the browsing session.

These primitive functions, according to the perception of users and the application domain, will be assigned a friendly label and organized in the user interface. For instance, the OP-Net can be displayed in a window and a button for the function REDO( ) should be arranged close to the OP-Net. However, instead of using REDO as the label of the button, the label BACK is used because of its popularity among various browsers.

5.6.3 Exploring Process Controller

The exploring process controller (EPC) controls the exploring process. Before performing the exploring process, the user needs to activate some particular functions, which force the system to switch to the exploring mode from the current mode. The interface controller receives this request and makes the necessary changes to the interface in order to support the exploring process, and informs the browsing controller that the browsing mode has been changed. The browsing controller will then give control to EPC. From now on the EPC is ready to take any request from users.

All the functions provided for the exploring process are similar to those provided by the topic refining process. Each function provided by the EPC produces a new E-Net. As


Users
Request:
To refine the current topic to CIS AND Roy
Result
Interface Controller
Information
•Function: Refining Topic (Graphical Component ID = TR_TextField1)
•Value(New Topic): CIS AND Roy
•Mode: Topic Refining Mode
Result
Browsing Process Controller
Task Identification and Dispatching
Result
Topic Refining Process Controller
Task ID = 2
Information
•Function: Refining Topic (Graphical Component ID = TR_TextField1)
•Mode: Topic Refining Mode
Request
return the TaskID and the Controller
Return
•Controller = TRPC
•Task ID = 2
Local Memory
Current Mode
Topic Refining Mode
User's Input
CIS AND Roy
Object

Task Description Table

Graphical Component ID	Browsing Mode	O_Flag	Task ID	Sub-Process Controller
			1	TRPC
TR_TextField1	Topic Refining Mode	0	2	TRPC
			3	TRPC
EX_Menu_Item1	Examining Mode	1	4	EXPC
EX_Menu_Item2	Examining Mode	1	5	EXPC
:	:	:	:	:

Fig 5-24 The data flow of an non-object-originated task

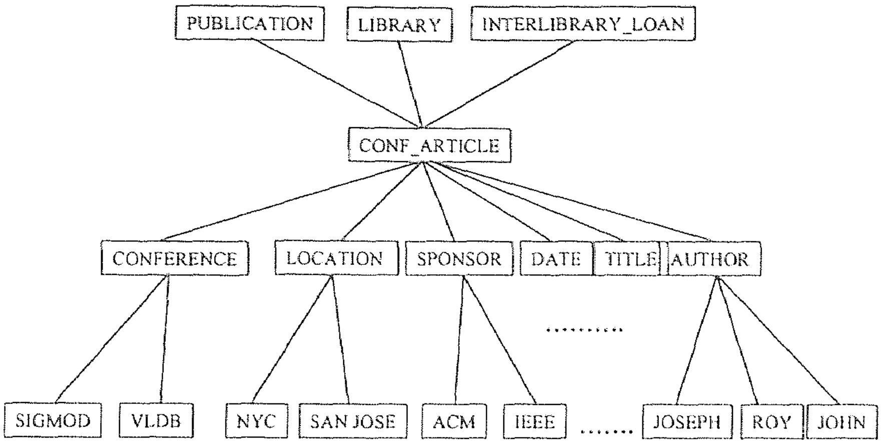


Fig 5-25 The OP-Net for CONF_ARTICLE

we mentioned in Section 3.2, the OP-Net and E-Net are structurally identical but they originate from different networks. The E-Net focuses on exploring the inter-relations between objects in the OP-Net. However, to perform these functions in exploring mode the user needs to follow the following steps.

1. Select a node on the OP-Net.
2. Select a function to be performed.

From the user's point of view, they can only modify the current topic by using the objects in the OP-Net, since the range of browsing process in exploring mode is restricted in the current OP-Net. Moreover, since all the information about the objects in the OP-Net is stored in the corresponding frame instance repository, it could be easy for the network constructor to construct the E-Net. The topic of the exploring process combines the topic of the OP-Net and those functions performed by the user. For example, assume that the topic is T_{OP} before switching to the exploring mode. Then the user switches to the exploring mode and performs a function **AND(obj1)**. The new topic T_E became T_{OP} **AND** *obj1*.

The following functions controlled by the EPC support the exploring process.

- **NEW(Object)** This function has an object as its argument and construct a new E-Net. This function can be performed only when the OP-Net has been constructed and the browsing mode has been switched to exploring mode. The following example is used to explain this function. Consider an OP-Net for a frame template CONF_ARTICLE in Fig 5-25. If the user switches to exploring mode, selects the PUBLICATION and requests to perform NEW( ) function, then the resulting E-Net is shown in Fig 5-26 (Assume that there is no common frame instance in the PUBLICATION, the LIBRARY, and the INTERLIBRARY_LOAN). This E-Net has the documents of the document type CONF_ARTICLE, which are stored in the folder PUBLICATION. This E-Net allows the user to explore the information, such as "find all conference articles from the PUBLICATION folder authored solely by John".
- **AND(Object)** In the exploring mode with an E-Net displayed, this function is called to modify the current topic. Not like the topic refining process, users can only perform this function by clicking on an object. The object does not have to go through the topic interpretation process for it is a controlled object. This function reduces the size of the current E-Net.
- **OR(Object)** This function can be used to enlarge the browsing range of the exploring process. When a user switches the browsing mode to the exploring mode, the underlying network becomes an E-Net, but the OP-Net is still available. This function makes sense only when the Object is in the OP-Net but not in the current E-Net. Since the E-Net is a subset of the underlying OP-Net, it can grow

only as large as the OP-Net, regardless of the number of times the OR( ) function is applied.

- **NOT(Object)** This function is similar to the NOT(Object) in the topic refining process. The frame instance set associated with the selected object will be excluded from the current frame instance set FI_E of the E-Net.
- **RESET( )** The function allows users to go from the current network to the previous one.

For the topic refining process or the exploring process, The TRPC and EPC provide users with a set of functions. The EPC is only responsible for preparing the information for the network constructor. This information contains the result from the topic interpretation process if it is an uncontrolled argument, and the corresponding Boolean operator if one of the AND, OR, or NOT functions is performed. Based on this information, the network constructor is able to construct the network.

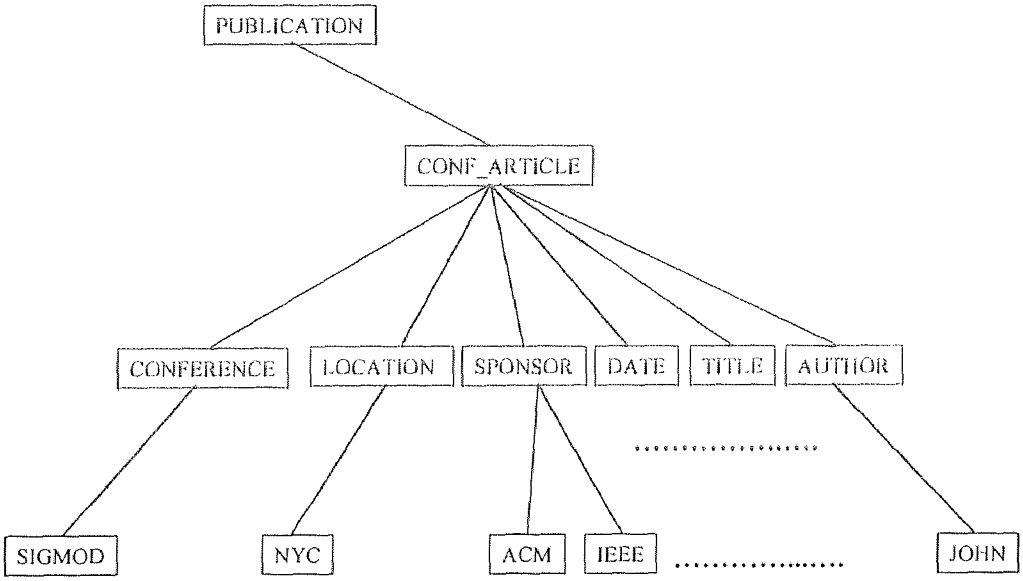


Fig 5-26 The E-Net for PUBLICATION

5.6.4 Examining Process Controller

The examining process controller provides functions for exploring the system knowledge provided by the knowledge base documents, the frame instances stored in the frame instance base or the original documents stored in the original document base. These functions are well organized in the user interface and monitored by the interface controller. In this way, users can switch to the examining mode at any time during the browsing session. For instance, in the topic refining process, after the OP-Net has been constructed, the user can select a node in the OP-Net and ask to view all the associated frame instances or the original documents. The user can also ask to view all the frame instances related to the query. Some examining functions which are associated with objects, are bound to objects according to their object types. For instance, for the objects of the FOLDER type, the examining process may provide users with the functions allowing users to examine the child or the parent folder of a particular folder. But for an object of the frame template type, the functions would be changed to allow users to examine the super type or the sub-type of the frame template.

5.6.5 Interface Controller

The interface controller controls the interface between the system and the users. Each sub-process controller provides users with a set of functions. From the system point of view, these functions can be activated at certain time in certain browsing mode. From the user point of view, they don't care about the current browsing mode; instead, they only care which functions they can perform. Under this consideration, we want to create an environment that is easy to use. To achieve this, the interface must take charge of the system logic. That means, if a certain function should not be performed at a certain time, then the interface should disable this function so that the user will not have any chance to perform it.

The interface controller consists of two main components namely the interface unit and the control unit. The interface has graphical components such as buttons, text fields, and the menu bar. Behind these graphical components, the functions are waiting for the events to happen. These functions make up the control unit. For instance, the graphical component *text field* is designed for users to enter the topic. When the user enters the topic, an event will be produced. The event triggers the function behind the text field. This function collects the topic issued by the user, gathers all the information when the event is produced, and then sends it to the browsing controller. Among this information, the most important one probably is the task ID. As being discussed in Section 5.9.1, the browsing controller needs the task ID to figure out the suitable sub-process controller to take care of the request.

These functions are organized in such a way that the browsing modes are almost transparent to the user. Consider an object in the OP-Net as an example. The functions provided by the browsing sub-processes would be organized for operating upon objects. If the object is a folder, users can see the function "modify the current topic", "Examine the associated frame instances", "Explore the folder organization" and many others. Executing the function "Examine the associated frame instances" will trigger the examining process. Executing the function "modify the current topic" will trigger the topic refining process or the exploring process. However, these changes are transparent to the user.

CHAPTER 6

KNOWLEDGE BASE

6.1 Information Retrieval

In [12, 35], van Rjisbergen has pointed out the difference between data retrieval and information retrieval. The most important differences are as follows:

1. **Matching mechanism:** In information retrieval, it's not always possible to have information which is a perfect match against the user's request, because the information can be unstructured and unorganized. Information may be searched within documents. The partial match between information and the user's request is a better and effective approach. In data retrieval system, data are well organized and structured and therefore the exactly matching items can be obtained.
2. **Inference mechanism:** In data retrieval systems (e.g. database system), the inference is more like the simple deduction among relations. For example, given two facts ***a is-a b*** and ***b is-a c***, then we can conclude that ***a is-a c*** also. However, information retrieval systems use inductive inference to specify the degree of certainty or uncertainty. With this degree, our confidence in the inference is variable. This difference also describes that data retrieval system is deterministic but the information retrieval system is probabilistic.
3. **Query mechanism:** In data retrieval systems, an artificial query language (e.g. structured query language) is commonly used, the query needs to be complete for retrieving data items. On the contrary, the use of natural language is a better way for information retrieval.
4. **Classification:** From the classification point of view, the data retrieval system is

interested in the monothetic classification instead of polythetic classification which is a better approach for the information retrieval system.

When compared with the data retrieval system, the information retrieval system provides an environment which is closer to the human's perception for issuing a query and getting the result. Usually a user will begin his/her search for the needed information by starting the search process with something that impressed him/her the most; however, most of the time the query is vague and incomplete because this impression is not complete, well structured and well organized data. In data retrieval systems, there is no way to get the answer if the query is not complete.

TEXPROS aims to be an intelligent document processing system. Therefore, as a retrieval sub-system, the browser needs to provide the functionality of the information retrieval system.

6.2 Knowledge Based System for the Browser

Sometimes, a knowledge-based system is called a rule-based system or an expert system. A typical knowledge-based system contains a knowledge base and an inference engine [36, 37, 38, 39, 40, 41, 42, 43, 44]. The knowledge base contains knowledge and the inference engine is built to operate on the knowledge. Before storing the knowledge into the knowledge base, we have to decide the knowledge representation. There are several ways to represent the knowledge. In TEXPROS, a semantic network called the object network is used to represent the knowledge. During the browsing process, the object network is transformed into OP-Net; since these networks are structural identical, the

Frame template : Memo

Super-type	Document
Sub-type	MeetingMemo, QE_Memo, VenderMemo
Has-attributes	Sender, Receiver, Subject, Date, CC
Contain-in-folder	CIS, NJIT

Fig 6-27 The frame of the object FrameTemplate (Memo)

same inference engine can be used for all the networks. In other words, during the browsing session, the OP-Net (E-Net) also becomes a part of the knowledge base.

In addition to the knowledge base and the inference engine, a knowledge-based system needs to provide tools to help users maintain or use the system. In TEXPROS, during the processes (such as the classification process, the extraction process, the filing process, and the browsing process), the controller, which conducts these processes, provides a knowledge base with the information they gathered from each of the processes.

The current system catalog serves as the knowledge base of TEXPROS and supports many of these processes. We shall explain this component in the next section.

6.3 Object Network

6.3.1 Knowledge Representation

The object network can be considered as a semantic network [14, 25], which is a way to represent the knowledge with data structures for responding to user's requests efficiently. The information stored in the object network contains the folder organization, the document type hierarchy, and the relationships between the same type (e.g. the relationship between attributes of a frame template) and the different type of objects (e.g. the relationship between the folder objects and frame template objects). These relationships provide the underlying facts for the knowledge base. Based on these facts, upon the arrival of a request from the user or an internal component, the object network can do the reasoning and return the result by spreading activation from node to node through the links.

Since the object Network is a semantic network, there are various ways of implementing it. One of them is to consider it as a collection of frames [31]. A frame is a collection of attributes and values associated with a particular object, which is represented as a node in the network. Fig 6-27 is an example for representing the frame template. By using *Contain-in-folder(FrameTemplate(Memo))*, the value of the attribute contain-in-folder can be retrieved. The following equation explains this procedure.

contain-in-folder(FrameTemplate(Memo)) = (Folder(CIS), Folder(NJIT))

These primitive functions provided by the frame can be organized into routines and augment the power of the semantic network. In the next section, we take the document type hierarchy as an example to explain how the semantic network mechanism works.

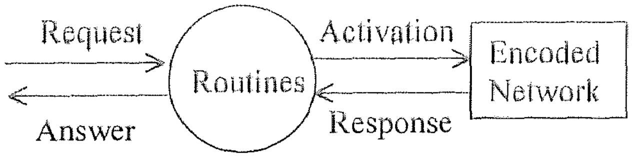


Fig 6-28 Semantic Network Architecture

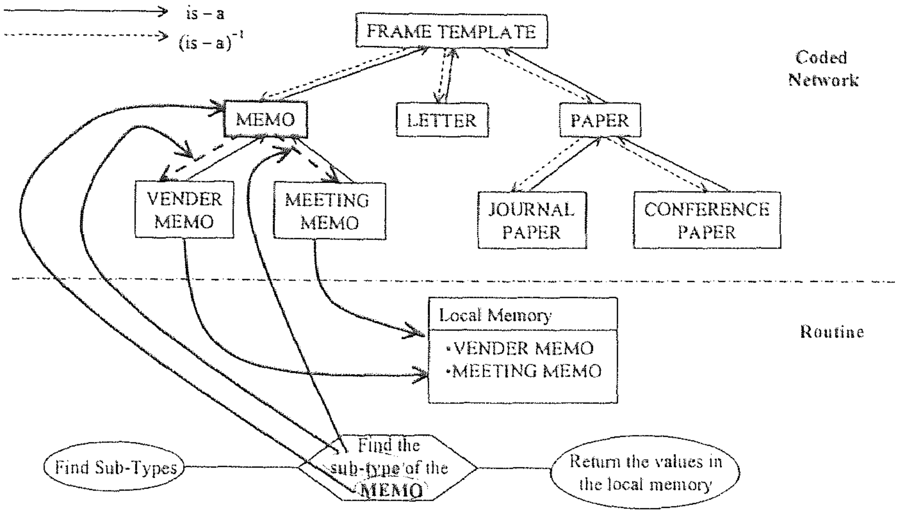


Fig 6-29 Example of the Semantic Network

6.3.2 Example

As we stated in the previous section, a semantic network contains a way of representing the knowledge with an efficient data structure for reasoning [45]. In TEXPROS, since the users are not allowed to interact with the knowledge base directly, the knowledge base is an internal service provider. From the implementation point of view, although the semantic network is a large and fully connected network, it needs to be broken down into smaller pieces with interconnections. We called this an encoded network. To access this encoded network, a set of routines needs to be defined. A simplified semantic network architecture (in Fig 6-28) has been shown in [45].

We now start with a simple example.

Example 6-1: (To find the sub-type of the MEMO)

Let us consider the document type hierarchy (DTH) as shown in Fig 6-29, which is a portion of the object network. DTH is a simple example in the sense that it is organized by using a typical relationship is-a. In this example, in order to make it clearer, we also removed the properties and their values associated to the document types in the hierarchy in Fig 6-29.

We shall describe the notation we used in Fig 6-29. In this example, we adopted the notation used in [45]. There are only two kinds of links that appear in the DTH, namely is-a and the inverse of the is-a which are represented by the solid arrow and the dotted arrow, respectively. The inverse of the is-a relation can also be denoted as is-subtype-of. However, we use the inverse to show that it originated from the is-a relation. There are two kinds of arcs that connect the coded network and the routines. First, the arcs leaving the nodes in the routines to the nodes in the network represent that the nodes in the network receive the activation from the routines. Secondly, the arcs that come out of the network activate the answers. The oval nodes in the routines represent the action steps and the hexagonal nodes represent the queries.

In semantic network, it is assumed that the default relation among the nodes in the network is the is-a relation. This example expresses a typical routine in the semantic network. This routine activates the node MEMO and from the MEMO node it follows $(\text{is-a})^{-1}$ links (represented by a highlighted dotted arc). The *action* will activate a set of nodes and links. Each activated node will be considered as a starting point. By following the activated links, a set of nodes will be identified. These identified objects will be directed back to the routine as alternate answers. In our example, the VENDER MEMO and the MEETING MEMO will be identified and sent back to the routine.

Next, we shall introduce another example to show why we use the semantic network in our system.

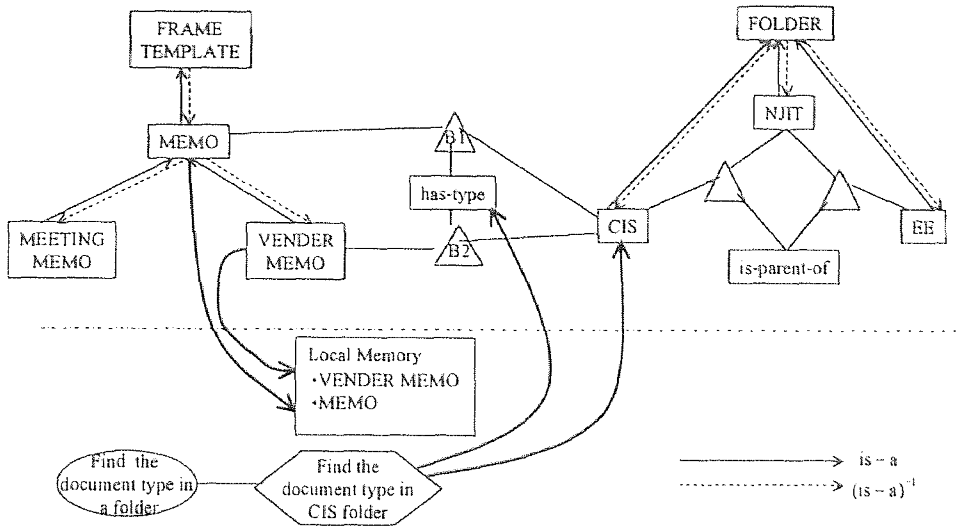


Fig 6-30 Another Example of the Semantic Netowrk

Example 6-2: (To find the document types of the documents stored in the folder CIS)

This example is more complicated than the previous one in the sense that we have to utilize more than one type of relation in the network for solving this query. The semantic network in Fig 6-30, unlike the one in Example 6-1, contains the typical elements of a semantic network which are concepts (objects), their properties, and the superclass/subclass relationships. The triangle symbols (called the triangle binder nodes) in the semantic network are used to associate objects with properties, and property values. Therefore, each triangle binder node has three links. When two nodes propagate the activation through links, the binder node is activated and by following the only inactivated link of the activated binder node, the alternate answers can be found.

In this example to find the document types in the folder CIS, the routine has to look up the property *has-type* of the folder CIS. The routine activates the regular node CIS and then activates the property node has-type. Consequently, the activation propagates to reach the binder node B1 and B2, which are, in turn, activated and then the activation reaches the property value nodes MEMO and VENDOR MEMO which will be returned as alternate answers.

These two examples are used to explain the architecture of the semantic network. In the following section, how this architecture of the semantic network fits into our three-layer browser architecture will be discussed.

6.3.3 Object Network in the Browser Architecture

As part of the knowledge base, the object network has routines to provide portions of the services. In the previous section, before finding the answers of the requests, the corresponding routines need to be called first. However, can we predict what are the routines we need? Before construct the semantic network, the designer derives a small set of routines by analyzing the problem domain where the system will be used, and consulting with the users of the system. In TEXPROS, these routines become the public methods for other components in the three-layer architecture. These basic routines are summarized but not limited to those given below:

1. Given a folder, find its parent folder.
2. Given a folder, find its child folders.
3. Given a folder, find its sibling folders.
4. Given a folder, find the document types of its contained documents.
5. Given a template, find its associated attributes.
6. Given a template, find all folders which contain at least one document of the template type.

7. Given an attribute, find all the frame templates which have the attribute.

8. Given an attribute, find its associated values.

9. Given a value, find all the attributes which contain the value.

These basic routines can further be assembled to form another set of routines. For instance, for finding out the folders which contain documents sent by John, a procedure consisting of a set of routines that can respond to this query is needed. In TEXPROS, the procedure for dealing with the query is taken care of by the search engine. The new routines could be composed by the basic routines provided by the knowledge base.

6.4 Thesaurus

Another important component of the knowledge base is the thesaurus [1, 46]. Currently, the thesaurus only provides service to the topic interpreter. The basic idea of the thesaurus is to form the word groups, each group having its common properties, and assign the group to a representative called *index term*. The index terms are thus used internally by the system for every process. From the retrieval point of view, the thesaurus improves the matching capability by broadening the interface for users' input. To design a thesaurus, we first need to consider how to group words, and we also need an efficient data structure to support fast access.

6.4.1 Main Component

In a traditional information system, the indexing approach deeply affects the way that the system organizes the thesaurus. However, in an office information system this may not necessarily be true. In TEXPROS, the extraction process is more complicated than the traditional full text information system. The frame instance is used as the document representative instead of using a vector of key terms. By using frame instances, we incorporate not only the vector composed by key terms but also the information about the metadata. Since the vector representation is a subset of the frame instance, all the mechanism developed for the key term vector can still be used in our system to a certain extent.

In TEXPROS, the thesaurus contains two parts, which are stored in the system folder as system frame templates, namely SYSSYNONYMS and SYSNARROWER. The SYSSYNONYMS provides a mapping between the key term and its synonymous terms. The SYSSNARROWER provides a mapping between the key term and its narrow terms. For instance, *Teaching Assistant* can be a narrow term of *Student Assistant*. From the implementation point of view, besides these two system frame templates, we still need some subsidiary structures to augment the capability of the thesaurus.

6.4.2 Subsidiary Component

Most of the thesaurus provides stem resolution utility to improve the matching capability [1]. Consider the following example. A user enters a query containing a term "happiness". It's possible that in our thesaurus there is no such key term. However since we have stem resolution utility, we can reduce the word from the thesaurus entry. For example, to reduce the word "happiness", the word stem utility needs to perform the following procedure.

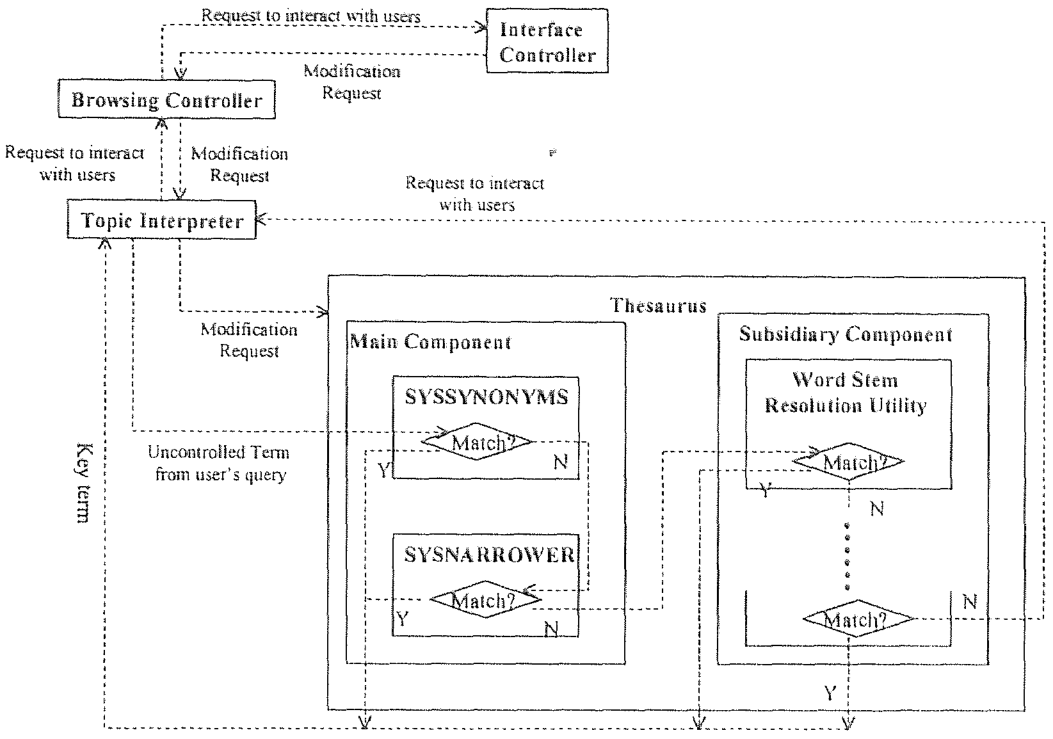


Fig 6-31 The Data Flow of the Thesaurus

- Reduce the "happiness" to "happi" by removing the "ness".
- Use a final 'y' for an 'i'. As a result, the "happi" will be replaced by "happy".

We then take "happy" as a thesaurus entry for finding its key term.

Obviously, in order to provide the word stem resolution, we also need a dictionary of suffixes to assist the word form reducing process.

6.4.3 Thesaurus in the Browser Architecture

In TEXPROS, currently the thesaurus only provides its services for the topic interpretation process. During this process, the thesaurus will receive the requests from the topic interpreter. The thesaurus checks each term in the user's topic and replaces it by the key term used by the system. When the thesaurus cannot find the key term for the term in the query, it reports to the browsing controller. The browsing controller will then request the interface controller for interacting with the user. At this point, the user joins the resolution process and gets a chance to update the thesaurus. The process is shown in Fig 6-31.

6.5 Summary

It is of utmost important to have better and precise understanding of the needed information for the retrieval system. From the system point of view, it's a tradeoff between the preciseness and the broadness of acquiring the information based on a user's vague query. Therefore, to understand a user's information need is crucial for the retrieval system. The knowledge base is an important component that can help the system to achieve the reasonable compromise between preciseness and broadness. However, currently our knowledge base provides mostly the facts but not rules. This needs to be strengthened especially for the thesaurus. Synonyms, narrow terms, or word stems only focus on the structural similarity of terms. For example, assume that Beth is the chairperson of EE department. When a user wants to find a technical paper authored by Beth, then it is possible that the paper can be retrieved if the user's query includes the term *Beth* but cannot be retrieved if the user only knows that the paper is written by the chairperson of EE department. That means it's very difficult for the system to identify that Beth and the chairperson of the EE department are identical without using the thesaurus. In our future research, we should investigate the solution for this problem.

CHAPTER 7

DOCUMENT RANKING

The browsing process is activated by users' query. Since the initial query normally does not precisely describe users' intentions because of the inadequate or the incorrect information provided by users [14, 25,47], the result returned by the browser may not always be satisfactory. Therefore, there is an evaluation process which allows users to review the result, gather some information about the system, and decide what's the next step to be taken. From the system point of view, during the evaluation process, the browser needs to provide all the information (such as the child folders and the parent folders of a particular folder, the sub-type and super-type of a particular document type, or the attribute type of an attribute) that might be useful for users. These can be done by incorporating the semantic network architecture into the graphical user interface, along with the functions offered by the browsing process controllers. However, in the initial steps of the browsing session, the size of the network could be large and the number of retrieved documents could be huge. Users could easily get lost in such a big information jungle. We try to provide mechanisms for reducing the size of the network and the volume of the retrieved documents.

The evaluation process has the functional capabilities of providing users with the system knowledge and an environment for reviewing the relevant documents with respect to the current topic. In TEXPROS, the system knowledge can be obtained from the knowledge base. Every request for getting any system knowledge could be precisely specified and therefore can be answered directly by the semantic networks. In this chapter, we shall investigate the other problems, how to provide an environment for users to examine the relevant documents, and then continue the browsing process.

In TEXPROS, various semantic networks (OP-Net, E-Net and O-Net) are the main components in the graphical user interface. We organize the returned documents by categorizing them and associating them with the objects. In [10], Motro proposed an important concept -- "access by values". People tend to remember the content of the document, but a user can not query in a formal retrieval system unless he/she also knows some attribute-value pairs. By providing "access by values", the system gives users an intuitive way of retrieving the information. In our browser, we extend this concept by considering its applications in an office automation environment. The values in "access by values" are not restricted to those values referred to in the traditional database system. In TEXPROS, these values can also refer to the name of a folder, a template, and an attribute. This is a way to cover more relevant documents and improve the recall. High recall also means that more documents are retrieved at the same time. This may create a problem for users during the evaluation phase. Among these documents, users don't even know where to start the evaluation process. A solution for this problem can be provided by ranking the documents.

In TEXPROS, we created a new ranking model by taking into account the traditional ranking model and the situation we encountered in the office automation environment. In the following sections we shall introduce our ranking model.

7.1 Ranking Unit

Ranking unit is responsible for producing the ranking list for relevant frame instances (documents) with respect to a given query in FI_{OP} or FI_{E}. After the OP-Net or E-Net is constructed, the network constructor allows the user to make a request to the ranking unit. In this section, we first describe our ranking model and then we introduce the ranking function.

7.2 Ranking Model

Various models of the information retrieval system [1, 48, 49, 50, 51] have been proposed. The Boolean model compares Boolean query statements with the terms, which represent the document. The probabilistic model uses the probabilities of relevance for the documents of the collection. The vector-space model represents both user queries and the documents by a set of controlled terms (e.g. index terms) and computes the similarities between them. Most of these models do not support the ranking ability. We take the traditional Boolean model as an example [1]. There are only two valid similarity values, 0 and 1. In this model, both the document and users query are also represented by a set of index terms. A document is retrieved if the term set of the document matches the term set of the query. Therefore, this model cannot produce the ranking list for the relevant documents. The probabilistic model [34] does not improve the retrieval effectiveness, since it is difficult to obtain the values for the term-occurrence parameters (e.g. term dependencies) [1], the probabilistic model did not improve the retrieval effectiveness. In the vector-space model, the document and the query are represented by the vectors, which allow the system to compute the distance between them. The similarity between documents and queries can then be defined by these distances. The vector-space model is easy to use and productive. A shortcoming of this model is that any user has no way to express the dependencies between terms that appeared in the document or the query. In our system, we extended the vector-space model which allows us to express the dependency of terms to some extent.

7.3 Ranking Model – TEXPROS Approach

In TEXPROS, a frame instance is a synopsis of an original document and is defined as a set of attribute-value pairs [25, 26]. Each document will be deposited into folders of a folder organization and be classified as a document type according to the document type hierarchy. The folder organization and document type hierarchy can be treated as the property of a document. We thus define the *signature* for documents.

Definition 7-1: (Signature for a document)

The signature of a document D, Sig(D)=[F_D,T_D,Fi], where

1. F_D is the set of folders where D is deposited;
2. T_D is the document type of D, and
3. Fi is the frame instance corresponding to D.

For example, let Sig(D)=[(CIS, PHD), MEMO, {(Sender, John),(Receiver, Tom), (Subject, TA meeting), (Date, 9/11/96), (CC, CIS)}]. Then there is a document D of the memo type which is deposited in the folders, CIS and PHD. The information about the content of a document is revealed by the frame instance in the signature.

We also need to define the signature for user's query. Given an original query Q, applying the query rewriting and normalization processes, Q is rewritten as Q', which is in the disjunctive normal form. Therefore, Q′ can be represented by $\bigcup_{i=1}^{r} qi$, where each qi is in the conjunctive normal form. We call qi the And-Clause. For each qi, its signature is defined as follows.

Definition 7-2: (The signature for And-Clause)

The signature of a And-clause qi, Sig(qi)=[F_{qi},T_{qi},A_{qi},V_{qi}], where

1. F_Q is the set of folders appearing in qi;
2. T_Q is the set of templates appearing in qi;
3. A_Q is the set of attributes appearing in qi, and
4. V_Q is the set of values appearing in qi.

We extract information about the sets of folders, templates, attributes and values from each qi in Q and put them in the signature. Sometimes, some information is not available. For instance, the And-Clause qi =*(VALUE[D. Sanders] AND FOLDER[CIS])* does not contain any information about the document type and the attributes. For this case, the information about the document type and the attributes in qi is insignificant. We use "*" in Sig(qi) to indicate the don't-care values. Therefore, Sig(qi)=[{CIS}, *, *, {D. Sanders}]. With the definition of the And-Clause, we now define the signature of the query.

Definition 7-3: (The signature for the query)

The signature of a query $Q = \bigcup_{i=1}^{r} qi$, $Sig(Q) = \bigcup_{i=1}^{r} Sig(qi)$.

Both the signatures of a document and the qi's appearing in a user's query are structurally identical. This allows us to compute and to compare the similarity of these two signatures. In the next section, we will describe how we can compute the similarity between these two signatures. The similarity between a document and the user's query can be computed, if the similarity between a document and each qi in Q is computed. Therefore, it is possible to rank the relevant frame instances in our system.

7.4 Ranking Function

The ranking function is to compute the similarity between a document and the given query. The query is represented in a format of the disjunctive normal form. The ranking function needs to take into consideration the dependency between terms introduced by AND and OR operators. Our approach is to use the signature to take care of AND operator; then we take care the OR operator by summing up the scores of each And-Clause. The document which receives higher score will have the higher rank. The similarity is computed using the following equation.

Equation 7-1:

Sim(Fi, Q)= $\sum_{qi \in Q} (Sim(F_D, F_{qi}) + Sim(T_D, T_{qi}) + Sim(Fi_D, (A_{qi}, V_{qi})))$.

This equation consists of the following three parts.

1. Sim(F_D.F_{qi})

Sim(F_D, F_{qi}) can be perceived as an equation for computing the similarity between the query and the document based on the information revealed by the folders. The similarity can be computed by counting the number of common frame instances associated with the query and the document.

Let

$$F_{qi} = \{F_{Qil} \mid F_{Qil} \text{ is a folder}, 1 \le l \le n\} \text{ and}$$
$$F_D = \{F_{Dj} \mid F_{Di} \text{ is a folder}, 1 \le j \le m\}$$

be sets of folders, where n and m are the number of the folders in F_Q and F_{qi}, respectively. Let $FI(F_{Qi})$ represents the frame instance set associated with the folder F_{Qi}. Then FI_{qi} and FI_D can be computed as follows:

$$FI_{qi} = FI(\bigcup_{l=1}^{n} F_{Qil}) \text{ and}$$
$$FI_D = FI(\bigcup_{j=1}^{m} F_{Di}).$$

If $F_{qi} = \varnothing$, then Sim(F_D, F_{qi}) is equal to 0. When $F_{qi} \neq \varnothing$, Sim(F_D,F_{qi}) can be computed using the following equation.

Equation 7-2:

$$Sim(F_D, F_{qi}) = \frac{2\#(FI(F_D \bigcap F_{qi}))}{\#(FI(F_D)) + \#(FI(F_{qi}))}.$$

In this equation, #A represents the number of elements in the set A.

2. Sim(T_D, T_{qi})

In the ranking function, the second contribution comes from the information about the document types.

If $T_{qi} = \varnothing$, then Sim(T_D,T_{qi}) is equal to 0. When $T_{qi} \neq \varnothing$, we compute Sim(T_D,T_{qi}) by using the following equation.

Equation 7-3:

$$Sim(T_D, T_{qi}) = \frac{2\#(A_D \bigcap A_{qi})}{\#(A_D) + \#(A_{qi})}.$$

Frame templates are used to represent various document types. Each frame template is identified by a set of attributes. For instance, if we treat e-mails as a document type, then it is reasonable to have SENDER, RECEIVER, SUBJECT, and CC as their attributes. In this case, we say that the document type T_D is an e-mail and it is identified by A_D = {SENDER, RECEIVER, SUBJECT, CC}.

Let $T_{qi} = \{T_{qip} \mid T_{qip} \text{is a frame template}, 1 \le p \le k\}$. be a set of frame templates specified by the user's query.

Each T_{qip} can be identified by a set of attributes A_{qip}. Then A_{qi} can be represented as follows:

$$Aqi = \bigcap_{p=1}^{k} Aqip \quad .$$

3. Sim(Fi_D, (A_{qi}, V_{qi}))

In computing Sim(Fi_D, (A_{qi}, V_{qi})), we take all of the values filled in the frame template into consideration. In the vector model [1], the index terms form a vector space. In this vector space, we use the distance between vectors to compute the similarity between a user's query and documents, which are represented by vectors.

Each value filled in the frame template is treated as an index term. For each document, it corresponds a set V_D of values. V_{qi} is used for values specified by a user. A weight is assigned to each index term. In [1], the weight for an index term is computed as follows:

Equation 7-4:

$$W(V) = \log(\frac{N}{n}) + 1 .$$

In equation 7-4, V is an index term occurring in a document; N is the number of total documents stored in the system, and n is the number of documents containing the value V. Then, the common value set between the values occurred in document and values specified by the user is $V = V_D \cap V_{qi}$ where:

$V = \{V_i \mid V_i \text{ is a value}, 1 \le i \le r\}$,

$V_D = \{V_j \mid V_j \text{ is a value}, 1 \le j \le s\}$, and

$V_{qi} = \{V_k \mid V_k \text{ is a value}, 1 \le k \le t\}$.

Then Sim(V_D, V_{qi}) can be computed as follows.

Equation 7-5:

$$Sim(V_D, V_{qi}) = \frac{\sum\limits_{V_i \in V} (W(V_i))^2}{\sqrt{\sum\limits_{V_j \in V_D} (W(V_j))^2 \times \sum\limits_{V_k \in V_{qi}} (W(V_k))^2}}.$$

7.5 Normalization

All components in the ranking function share a common characteristic. That is, if s represents the range for the computed value for each components, then $0 \le s \le 1$. The purpose of the normalization process is to keep Sim(Fi, Q) within a reasonable range for every different frame instance and query pair.

Assume p is the number of the And-clauses appearing in the query Q. Then the normalized ranking value R can be derived as follows.

Equation 7-6:

$$R = \frac{Sim(Fi, Q)}{p * 3}$$

It's apparent that, $0 \le R \le 1$. The higher value R means that the document Fi is more relevant to the query Q.

In the remainder of the section, a complete ranking example will be given.

7.6 Example

Given a user's query **Q** and two exemplary frame instances (namely **f1**, **f2**). The similarity between the query and each frame instance will be computed. The query and the description of three frame instances are given in Fig 7-32. Assume that after the object identification and the query rewriting processes, **Q** will be rewritten to **Q′**.

Q′≡ (ATTRIBUTE(Sender) **AND** VALUE(Roy) **AND** TEMPLATE(Memo) **AND** VALUE(TA Meeting) **AND** VALUE(CIS))

OR (ATTRIBUTE(Sender) **AND** FOLDER(Roy) **AND** TEMPLATE(Memo) **AND** VALUE(TA Meeting) **AND** VALUE(CIS))

OR (ATTRIBUTE(Sender) **AND** VALUE(Roy) **AND** TEMPLATE(Memo) **AND** VALUE(TA Meeting) **AND** FOLDER(CIS))

OR (ATTRIBUTE(Sender) **AND** FOLDER(Roy) **AND** TEMPLATE(Memo) **AND** VALUE(TA Meeting) **AND** FOLDER(CIS)).

According to the ranking model, four And-Clauses can be derived.

$\mathbf{q}_1$ ≡ ATTRIBUTE(Sender) **AND** VALUE(Roy) **AND** TEMPLATE(Memo) **AND** VALUE(TA Meeting) **AND** VALUE(CIS);

$\mathbf{q}_2$ ≡ ATTRIBUTE(Sender) **AND** FOLDER(Roy) **AND** TEMPLATE(Memo) **AND** VALUE(TA Meeting) **AND** VALUE(CIS);

$\mathbf{q}_3$ ≡ ATTRIBUTE(Sender) **AND** VALUE(Roy) **AND** TEMPLATE(Memo) **AND** VALUE(TA Meeting) **AND** FOLDER(CIS);

$\mathbf{q}_4$ ≡ ATTRIBUTE(Sender) **AND** FOLDER(Roy) **AND** TEMPLATE(Memo) **AND** VALUE(TA Meeting) **AND** FOLDER(CIS).

Then the signature for each And-Clause can be represented as follows:

Sig(q_1) = [*, {Memo}, {Sender}, {Roy, TA Meeting, CIS}];

Sig(q_2) = [{Roy}, {Memo}, {Sender}, {TA Meeting, CIS}];

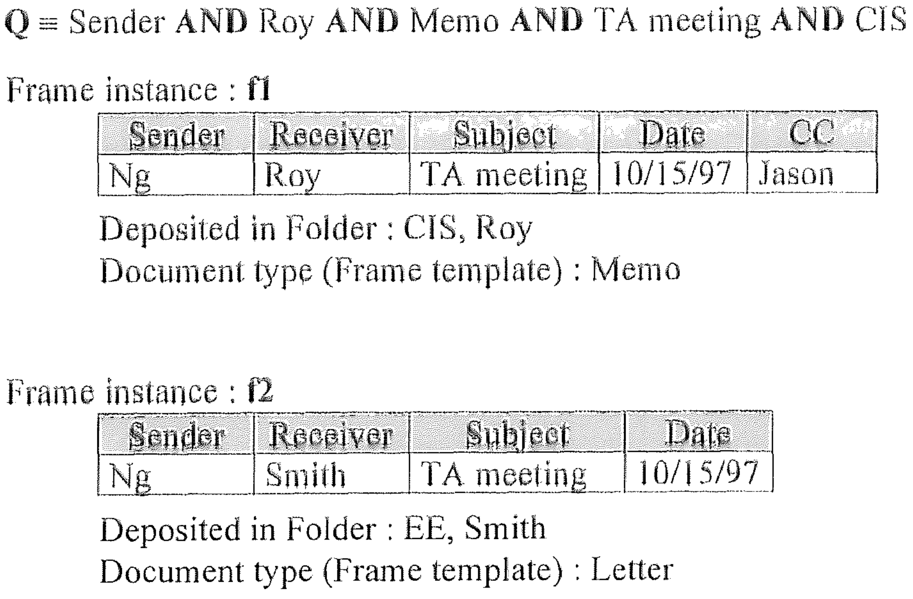
Sig(q_3) = [{CIS}, {Memo}, {Sender}, {Roy, TA Meeting}];

Sig(q_4) = [{Roy, CIS}, {Memo}, {Sender}, {TA Meeting}].

The signatures of two exemplary frame instances are represented as follows:

Sig(f1) = [{CIS, Roy}, {Memo}, {(Sender, Ng), (Receiver, Roy), (Subject, TA Meeting), (Date, 10/15/97), (CC, Jason)}];

Sig(f2) = [{EE, Smith}, {Letter}, {(Sender, Ng), (Receiver, Smith), (Subject, TA Meeting), (Date, 10/15/97)}].



Q ≡ Sender **AND** Roy **AND** Memo **AND** TA meeting **AND** CIS

Frame instance : **f1**

Sender	Receiver	Subject	Date	CC
Ng	Roy	TA meeting	10/15/97	Jason

Deposited in Folder : CIS, Roy
Document type (Frame template) : Memo

Frame instance : **f2**

Sender	Receiver	Subject	Date
Ng	Smith	TA meeting	10/15/97

Deposited in Folder : EE, Smith
Document type (Frame template) : Letter

Fig 7-32 The ranking example

The similarity between the document and the query consists of three components, which compute the similarities in terms of folders, templates, and frame instances. Considering the And-clause q_3 and the frame instance **f1**. The similarity between the document and the query can be computed using the following procedure.

Step 1: Compute Sim(F_D, F_{qi})

In this example, $F_D = F_{f1}$ {Roy, CIS} and $F_{qi} = Fq_3$ = {CIS}. Assume that the number of the frame instances deposited in the folder Roy and CIS are 6 and 10 respectively, and the number of the common frame instances deposited in these two folders is 5. According to Equation 7-2,

$$Sim(F_D, F_{qi}) = \frac{2\#(FI(F_D \bigcap F_{qi}))}{\#(FI(F_D))+\#(FI(F_{qi}))} = \frac{2*10}{11+10} = 0.95\,.$$

Step 2: Compute Sim(T_D, T_{qi})

The similarity between the document and the query is defined based on the common attributes contained in the document type. For this example, since $T_D = T_{f1}$= {Memo} and $T_{qi} = T_{q3}$ = {Memo} are of the same type. Therefore, the similarity is 1.

Step 3: Compute Sim(Fi_D, (A_{qi}, V_{qi}))

Assume that there are 400 documents in the system and 100 of them contain the value *Roy* and 200 of them contain the value *TA Meeting*. Then the weights for the value *Roy* and the value *TA meeting* are computed as follows:

$$W(Roy) = \log(\frac{400}{100}) + 1 = 1.60\,;$$

$$W(TA\ Meeting) = \log(\frac{400}{200}) + 1 = 1.30\,.$$

Similarly, assume *W(Ng)* = 1.20, *W(10/15/97)* = 2, *W(CIS)*=1.02, *W(Smith)* = 2.50, and *W(Jason)* = 2.6. Then using the Equation 7-5,

Table 7-2 The result of the computation of the similarity

	q_1	q_2	q_3	q_4	**Sim**	**R**
f1	1.46	1.96	2.46	2.32	8.20	0.68
f2	1.09	1.17	1.11	1.25	4.62	0.38

$$Sim(V_D, V_{qi}) = \frac{\sum_{V_i \in V}(W(V_i))^2}{\sqrt{\sum_{V_j \in V_D}(W(V_j))^2 \times \sum_{V_k \in V_{qi}}(W(V_k))^2}}$$

$$= \frac{(1.60)^2 + (1.30)^2}{\sqrt{((1.20)^2 + (1.60)^2 + (1.30)^2 + (2)^2 + (2.6)^2) \times ((1.60)^2 + (1.30)^2)}}$$

$$= 0.51.$$

Step 4: Compute Sim(Fi, qi)

Sim(Fi, Q)= $\mathrm{Sim}(F_D, F_{qi}) + \mathrm{Sim}(T_D, T_{qi}) + \mathrm{Sim}(Fi_D, (A_{qi}, V_{qi})) = 0.95 + 1 + 0.51 = 2.46$.

Up to step 4, only computation of the similarity of one And-clause (q_3) is completed. The computation of the similarity between **Q'** and **f1** requires computing the similarity of the rest of the And-clauses. The complete result of the similarity and the normalized ranking value between q1 and the f1 and f2 are listed in Table 7-2. Since f1 has a higher ranking, it is closer to the user's request and should be considered first.

CHAPTER 8

CONCLUSION AND FUTURE RESEARCH

In this dissertation, we proposed a new infrastructure (called an OP-Net) for the browsing process. Several of the major differences between an OP-Net and other networks are as follows: Firstly, we simplify the semantic meanings of the links on the network. In OP-Net, there is only one kind of link, and therefore users are easy to understand the relation between two connected objects. Secondly, documents are not explicitly represented in the network, and therefore the size of the network is small. This improves the efficiency of the retrieving performance during the browsing process. Thirdly, since the size of the network is small, instead of locating the relevant objects on the large static network, we are able to construct dynamically the OP-Net based upon the arrival of the user's query. Fourthly, every node of an OP-Net is a frame instance repository, which contains the relevant documents. Since the whole network can be displayed, users need not to be restricted in a short-sighted navigation when they evaluate the result. This makes the examining process more powerful. Finally, to create the representative for the documents and the query, we combine the Boolean query and ranking of the documents by introducing the signatures of the document and the query. The ranking suggests to the user which document should be examined first. In the near future, we intend to develop a ranking function for frame instance repositories based on the ranking of their associated frame instances.

At present, our research is focused on understanding the user's information need. We are taking an approach which identifies the concepts revealed by the user's request instead of only doing some string manipulation. We employ the concept matching technique for understanding user's needs more precisely. Currently, we use a simplified version of thesaurus, which is part of the system catalog. The thesaurus now features synonyms and narrow terms which is presented in [14]. We intend to augment the current thesaurus with the description of facts within the realm of application domains, and inference rules which allows the system to derive facts by reasoning and convert it into a knowledge base. The other issue is the problem of dealing with an empty result during the browsing process. During the browsing process, a user refines continuously their query and receives the immediate result generated by the system accordingly. However, it is possible that the system returns an empty result. That means, there is no document which is relevant to the current query. However, sometimes this is not the case, and in fact, there might be relevant documents for a given query [52]. The system needs to analyze why the empty answer set is generated and give the explanation, possibly with suggestions.

Future research will also focus on the document-based information retrieval instead of the document retrieval. Research work on information mining in the original documents and knowledge discovery from the original documents, mostly from their unstructured part, needs to be conducted. How the system can automatically build the knowledge base and how it can support the decision making are critical issues if we intend to make the document processing system applicable and useful in many office environments.

REFERENCES

1. G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*, NY, McGraw-Hill, 1983.

2. W.B. Croft and R.H. Thompson. I^{3}R: A New Approach to the Design of Document Retrieval. *Journal of the American Society for Information Science*. Vol. 38, No. 6, pages 389-404, 1987.

3. A. D'Atri and L. Tarantion. From Browsing to Querying. *IEEE Data Engineering*, Vol. 12, No. 2, pages 46-53, June 1989.

4. M.M. Zloof. Query-by-Example: A Database Language. *IBM Systems Journal*, Vol. 21, No. 3, pages 324-343, 1977.

5. G. Ozsoyoglu and H. Wang. Example-Based Graphical Databases Query Languages. *Computer*, pages 25-38, May 1993.

6. R.G. Cattell. An Entity-Based Database Interface. *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 144-150, Santa Monica, CA, May 1980.

7. C. Herot. Spatial Management of Data. *ACM Transactions on Database Systems*, Vol. 5, No. 4, pages 493-513, December 1980.

8. M. Stonebraker and J. Kalash. TIMBER: A Sophisticated Relation Browser. In *Proceedings of the 8th International Conference on Very Large Data Bases*, pages 1-10, Mexico City, Mexico, September 1982.

9. A. Motro. Browsing in a Loosely Structured Database. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 197-207, Boston, MA, June 1984.

10. A. Motro. BAROQUE: A Browser for Relational Databases. *ACM Transactions on Office Information Systems*, Vol. 4, No. 2, pages 164-181, April 1986.

11. A. Motro, D'Atri, and L. Tarantino. KIVIEW: The Design of an Object-Oriented Browser. In *Proceedings of the 2nd International Conference on Expert Database Systems*, pages 17-31, Vienna, VA, 1988.

12. C.J. Van Rijsbergen. *Information Retrieval*. Boston, MA, Butterworths, 1979.

13. M. Hertzum and E. Frφkjær. Browsing and Querying in Online Documentation: A Study of User Interfaces and the Interaction Process. *ACM Transactions on Computer-Human Interaction*, Vol. 3, No. 2, pages 136-161, June 1996.

14. Q. Liu and P.A. Ng. A Browser of Supporting Vague Query Processing in an Office Document System. *Journal of Systems Integration*, Vol. 5, No. 1, pages 61-82, 1995.

15. R.H. Thompson and W.B. Croft. Support for Browsing in an Intelligent Text Retrieval System. *International Journal of Man-Machine Studies*, Vol. 30, pages 639-668, 1989.

16. A.S. Pollitt. End User Touch Searching for Cancer Therapy Literature—a Rule Based Approach. In *Proceedings of the 6th Annual International ACM SIGIR Conference*, Vol. 17, No. 4, pages 136-145, 1984.

17. I. Monarch and J. Carbonell. CoalSORT: a Knowledge-Based Interface. *IEEE Expert*, 2, pages 39-53, Spring 1987.

18. D.L. McCracken and R.M. Akscyn. Experience with the ZOG Human-Computer Interface System. *International Journal of Man-Machine Studies*, Vol. 21, pages 293-310, 1984.

19. A. Celentano, M.G. Fugini, and S. Pozzi. Knowledge-Based Document Retrieval in Office Environments: The Kabiria System. *ACM Transactions on Information Systems*, Vol. 13, No. 3, pages 237-268, July 1995.

20. C.J. Crouch, D.B. Crouch, and K. Nareddy. A Connectionist Model for Information Retrieval Based on the Vector Space Model. *International Journal of Expert Systems*, Vol. 7, No. 2, pages 139-163, 1994.

21. P.D. Kochevar and L.R. Wanger. Tecate: A Software Platform for Browsing and Visualizing Data from Networked Data Sources. *Digital Technical Journal*, Vol. 7, No. 3, pages 66-83, 1995.

22. A. Poulovassilis and M. Levene. A Nested-Graph Model for the Representation and Manipulation of Complex Objects. *ACM Transactions on Information Systems*, Vol. 12, No. 1, pages 35-68, 1994.

23. A. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Navigating Hierarchically Clustered Networks through Fisheye and Full-Zoom Methods. *ACM Transactions on Computer-Human Interaction*, Vol. 3, No. 2, pages 162-188, June 1996.

24. C.Y. Wang, Q. Liu and P.A. Ng. Intelligent Browser for TEXPROS. In *ISATED Proceedings of International Conference on Intelligent Information Systems (IIS' 97) (edited by H. Adeli)*, IEEE Computer Society Press, pages 388-398, Dec 8-10, 1997.

25. Q. Liu and P.A. Ng. *Document Processing and Retrieval: TEXPROS*. Kluwer Academic Publishers, Norwell, MA, 1996.

26. J.T.L. Wang and P.A. Ng. TEXPROS: An Intelligent Document Processing System. *International Journal of Software Engineering and Knowledge Engineering*, Vol. 15, No. 4, pages 171-196, April 1992.

27. Z. Zhu, J.A. McHugh, and P.A. Ng. A Predicate Driven Document Filing System. *Journal of Systems Integration*, Vol. 6, No. 3, pages 373-403, 1996.

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

29. C.Y. Wang, Q. Liu, and P.A. Ng. Browsing in an Information Repository. In *Proceeding of 2nd World Conference on Integrated Design and Process Technology (edited by M.M. Tanik, etc)*, IDPT-Vol2, pages 48-56 ,1996.

30. P.H. Winston. *Artificial Intelligence*. Reading, MA, Addison-Wesley, 1992.

31. E. Rich and K. Knight. *Artificial Intelligence*. NY, McGraw-Hill, 1991.

32. F. Hayes-Roth. Rule-Based Systems. *Communications of the ACM.* Vol. 28, No. 9, pages 921-932, September 1985.

33. M. Stonebraker and J. Kalash. TIMBER: A Sophisticated Relation Browser. In *Proceedings of the 8th International Conference on Very Large Data Bases*, pages 1-10, Mexico City, Mexico, September 1982.

34. N. Fuhr. Integration of Probabilistic Fact and Text Retrieval. *In Proceedings of the 15th Annual International ACM SIGIR Conference*, pages 211-222, Denmark, June 1992.

35. C.J. van Rijsbergen. A New Theoretical Framework for Information Retrieval. In *Proceedings of the 1986 ACM Conference on Research and Development in Information Retrieval.* pages 194-200, September 8-10, 1986.

36. R. Fikes and T. Kehler. The Role of Frame-based Representation in Reasoning. *Communication of the ACM*, Vol. 28, No. 9, pages 904-920, 1985.

37. D.D. Jaco and G. Garbolino. An Information Retrieval System Based on Artificial Intelligence Techniques. In *Proceeding of the ACM Conference on Research and Development in Information Retrieval*, pages 214-220, 1986.

38. A.R. Kaye and G.M. Karam. Cooperating Knowledge-Based Assistants for the Office. *ACM Transactions on Office Information Systems,* Vol. 5, No. 4, pages 297-326, October 1987.

39. L. Kerschberg. Expert Database systems: Knowledge/Data Management Environments for Intelligent Information Systems. *Information Systems*, Vol. 15, No. 1, pages 151-160, 1990.

40. A.R. Rao and R. Jain. Knowledge Representation and Control in Computer Vision Systems. *IEEE Expert*, pages 64-79, Spring 1988.

41. A. Shepherd and L. Kerschberg. PRISM: A Knowledge Based System for Semantic Integrity Specification and Enforcement in Database Systems.

42. P. Shoval. Principles, Procedures and Rules in an Expert System for Information Retrieval. *Information Processing & Management*, Vol. 21, No. 6, pages 475-487, 1985.

43. P.J. Smith, S.J. Shute, D. Galdes, and M.H. Chignell. Knowledge-Based Search Tactics for an Intelligent Intermediary System. *ACM Transactions on Information Systems.* Vol. 7, No. 3, pages 246-270, July 1989.

44. S.K.M. Wong and W. Ziarko. A Machine Learning Approach to Information Retrieval. In *Proceeding of the ACM Conference on Research and Development in Information Retrieval*, pages 228-233, 1986.

45. E. Lim and V. Cherkassky. Semantic Networks and Associative Databases. *IEEE Expert*, pages 31-40, August 1992.

46. R. Rada and B.K. Martin. Augmenting Thesauri for Information Systems. *ACM Transactions on Office Information Systems.* Vol. 5, No. 4, pages 378-392, October 1987.

47. Q. Kong and G. Chen. On Deductive Databases with Incomplete Information. *ACM Transactions on Information Systems*, Vol. 13, No. 3, pages 354-369, July 1995.

48. A. bookstein. A Comparison of Two Systems of Weighted Boolean Retrieval. *Journal of the American society for Information Science*, pages 275-279, July 1981.

49. G. Salton, A Wong and C.S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM,* Vol. 18, No. 11, pages 613-620, November 1975.

50. K.S. Jones. A Statistical Interpretation of Term Specificity and its Application in Retrieval. *Journal of Documentation*, Vol. 28, No. 1, pages 11-21, March 1972.

51. C.T. Yu and G. Salton. Effective Information Retrieval Using Term Accuracy. *Communications of the ACM,* pages 135-142, Vol. 20, No. 3, March 1977.

52. F. Corella, S.J. Kaplan, G. Wiederhold, and L. Yesil. Cooperative Responses to Boolean Queries. In *Proceedings of the 10th International Conference on Very Large Data Bases*, pages 77-85, Singapore, August 1984.