Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

Order Number 9514439

An office document retrieval system with the capability of processing incomplete and vague queries

Liu, Qianhong, Ph.D.

New Jersey Institute of Technology, 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

ABSTRACT

AN OFFICE DOCUMENT RETRIEVAL SYSTEM WITH THE CAPABILITY OF PROCESSING INCOMPLETE AND VAGUE QUERIES

by
Qianhong Liu

TEXPROS (TEXt PROcessing System) is an intelligent document processing system. The system is a combination of filing and retrieval systems, which supports storing, classifying, categorizing, retrieving and reproducing documents, as well as extracting, browsing, retrieving and synthesizing information from a variety of documents. This dissertation presents a retrieval system for TEXPROS, which is capable of processing incomplete or vague queries and providing semantically meaningful responses to the users. The design of the retrieval system is highly integrated with various mechanisms for achieving these goals. First, a system catalog including a thesaurus is used to store the knowledge about the database. Secondly, there is a query transformation mechanism which consists of context construction and algebraic query formulation modules. Given an incomplete query, the context construction module searches the system for the required terms and constructs a query that has a complete representation. The resulting query is then formulated into an algebraic query. Thirdly, in practice, the user may not have a precise notion of what he is looking for. A browsing mechanism is employed for such situations to assist the user in the retrieval process. With the browser, vague queries can be entered into the system until sufficient information is obtained to the extent that the user is able to construct a query for his request. Finally, when processing of queries responds with an empty answer to the user, a query generalization mechanism is used to give the user a cooperative explanation for the empty answer. The generalizations of any given failed queries (i.e., with an empty answer) are derived by applying both

the folder and type substitutions and weakening the search criteria in the original query. An efficient way is investigated for determining whether the empty answer is genuine and whether the original query reflects erroneous presuppositions, and therefore answering any failed query with a meaningful and cooperative response. It incorporates with a methodical approach to reducing the search space of generalized subqueries by analyzing the results of executing the query generalization and by efficiently applying the possible substitutions in a query to generate a small subset of relevant subqueries which are to be evaluated.

AN OFFICE DOCUMENT RETRIEVAL SYSTEM WITH THE CAPABILITY OF PROCESSING INCOMPLETE AND VAGUE QUERIES

by
Qianhong Liu

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

October 1994

APPROVAL PAGE

AN OFFICE DOCUMENT RETRIEVAL SYSTEM WITH THE CAPABILITY OF PROCESSING INCOMPLETE AND VAGUE QUERIES

Qianhong Liu

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

Dr. Jason T. L. Wang, Dissertation Co-Advisor — Date
Assistant Professor of Computer and Information Science, NJIT

Dr. James A.M. McHugh, Committee Member — Date
Associate Chairperson and Professor of Computer and
Information Science, NJIT

Dr. Murray Turoff, Committee Member — Date
Distinguished Professor of Computer and Information Science
and Management, NJIT

Dr. Raymond T. Yeh, Committee Member — Date
Chairman, International Software Systems, Inc.

BIOGRAPHICAL SKETCH

Author: Qianhong Liu

Degree: Doctor of Philosophy

Date: October 1994

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1994
- Master of Science in Computer Science,
  Beijing University, P.R.China, 1987
- Bachelor of Science in Computer Science,
  Beijing Information Technology Institute, P.R.China, 1985

Major: Computer Science

Presentations and Publications:

Q.H. Liu, J.T.L. Wang, and P.A. Ng, "On Research Issues Regarding Uncertain Query Processing in an Office Document Retrieval System," *Journal of Systems Integration*, vol. 3, no. 2, pp. 163-194, June 1993.

Q.H. Liu, J.T.L. Wang, and P.A. Ng, "An Office Document Retrieval System with the Capability of Processing Incomplete and Vague Queries," In *Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, pp. 11-17, June 1993.

J.T.L. Wang, F.S. Mhlanga, Q.H. Liu, W.C. Shang, and P.A. Ng, "An Intelligent Documentation Support Environment," In *Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, pp. 429-436, June 1993.

J.T.L. Wang, F.S. Mhlanga, Q.H. Liu, W.C. Shang, and P.A. Ng, "Database Support for Software Documentation: The TEXPROS Project," To appear as a book chapter in *Software Automation and Productivity Improvement*, 1995.

This dissertation is dedicated
to
my parents
Fuzi Liu & Jianting Zhang.

ACKNOWLEDGMENT

I am particularly grateful to my dissertation advisor, Professor Peter A. Ng, for his insightful guidance and encouragement throughout this research and his invaluable efforts in improving the writing of this dissertation. Thanks also go to Professor Jason T.L. Wang for his suggestion on earlier components of this research.

I would also like to thank the individual committee members. Professor Murray Turoff gave extensive comments on various issues, including incomplete and conflicting data. Professor James A.M. McHugh and Dr. Raymond T. Yeh provided a thorough review of the entire dissertation.

This research was supported in part by New Jersey Institute of Technology and by a grant from AT&T Foundation.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Information circulated in offices is often kept in documents. Some documents have rigid structures, such as forms [95]; some are text-oriented, such as letters, memos, brochures, reports, electronic mails, facsimile, etc. The documents may also contain graphics, images, audio and video data [96]. There has been a growing interest on developing document information retrieval systems, which support office workers to manage their information. Most of the previous work is based on the Office Document Architecture (ODA) [21, 38], which is part of the standards for document interchange developed by the International Standardization Organization (ISO) and the European Computer Manufacturers Association (ECMA). Basically, the systems fall into four categories [60, 107].

The first group deals with multimedia information including text, form, image and voice data. Diamond [91] allows users to create, edit, and transmit multimedia documents with simple retrieval methods. The MULTOS [2] office server supports a well-defined query language and query processing techniques. MINOS [16] provides integrated facilities for creating complex document objects and for extracting and formulating new information from existing documents. There are various data models proposed for multimedia documents, spanning form relational [89, 109], semantic [21, 76] to object-oriented approaches [32, 39, 40, 110].

The second group deals with bibliographic information retrieval by incorporating AI techniques into them. For example, SMART [79] supports keyword based retrieval for bibliographic database. EX-P [87] is an expert system which has the capability of retrieving information from documents concerning environmental pollution. Other document-based retrieval systems include CANSEARCH [73], RUBRIC [93], THOMAS [70], Expert/Consultation System [84], and others [14].

The third group is concerned with document categorization. Resumix [92] is one of such systems. It reads resumes, creates a summary of the resumes, matches applicants to job openings, generates reports, and prints letters of applicant acknowledgment with a bitmap signature from the appropriate hiring manager. Other systems such as the new story categorization system, CONSTRUE/TIS [36], also provide similar functions.

The fourth group is concerned with message exchanging and filtering. Examples are INFORMATION LENS system [54], ISCREEN [74], MIFIA [52], and the system described in [13]. The purpose of the systems is to help user filter, sort and prioritize messages that are already addressed to them, and also help them find useful messages they would not otherwise have received. Most of the systems only handle a special type of documents.

While these systems appear to be successful in their own domains, their functional capabilities are considerably limited. In a distributed, cooperative environment, where the most common documents are perhaps electronic messages [54], a document-based retrieval system must also support information sharing and exchange. These generally include the following activities: composing messages to be sent; selecting, filing and prioritizing messages that are received; and responding to messages. However, most of the existing systems have a monolithic design; it is difficult, though not impossible, to replace their components or to improve their functions for different user's need.

As part of a program of research in the Document Processing Group at the Institute of Integrated Systems Research, an initiative is set forth to investigate and develop a text processing system. Our research is directed towards producing a document processing system which can be used in a variety of domains and is intended to meet the above functional requirements.

1.1 TEXPROS

TEXPROS (TEXt PROcessing System) [107] is a personal, customizable system for processing office documents. The system has functional capabilities of automating (or semi-automating) common office activities such as document classification, filing, retrieval and reproduction, and information extraction, browsing, retrieval and synthesizing. To accomplish these goals, the system includes the following components:

- A state-of-the-art data model capable of capturing the behavior of the various office activities [60, 61, 106].
- Extracting the synopsis or the most significant information from a document (such information is often sufficient to satisfy the user's needs when information retrieval occurs) [34, 35, 108].
- A knowledge-based, customizable document classification handler that exploits both spatial and textual analysis to identify the type of a document [34, 35, 83, 108].
- An agent-based architecture supporting document filing and file reorganization [104, 105, 117].
- A retrieval system that can handle incomplete and vague queries [50, 51].

In brief, TEXPROS is for personal use, whereas the systems mentioned above are designed for a multi-user or distributed, cooperative environment (as a consequence, they need a standard protocol for document exchange). However, when using TEXPROS in an information sharing environment, it requires to specify protocols for governing the definitions of frame template, which describe the properties (or attributes) for the document classes. For example, when using TEXPROS as a

library bibliographic retrieval system, one may need to stipulate that the significant information for books in library contain attributes "authors", "affiliation", "subject", "title", "abstraction", "category", "classification", and so forth [106, 107].

This dissertation presents the retrieval system for TEXPROS.

1.2 Preliminaries

Most research concerning information retrieval in database systems is based on assumptions of precision and completeness of both the data stored in the database and the queries entered by the user for retrieving data. In reality, however, both may be incomplete or vague. A considerable amount of research has focused on issues which represent imprecise data in database ([27, 28, 30, 49]) and imprecise or vague requests to retrieve data ([23], [68]).

Consider a collection of documents to be stored in an information base. From each document, a synopsis of information is extracted to form a frame instance (reminiscent of the tuple in the relational data model). Frame instances can be classified according to their types which are called frame templates (reminiscent of the schema in the relational data model). The frame instances can be categorized based on the nature of their information and are placed in folders. Thus, a folder can contain a collection of frame instances of various frame template types[1] [107].

Figure 1.1 shows a folder named Q.E. that contains frame instances regarding qualifying examinations. Assume that this folder contains frame instances of the types Q.E.Result, Q.E.Application Form, Q.E.Question and Comprehensive Exam Result. Furthermore, assume that both the frame templates Q.E.Result and Comprehensive Exam Result have the attributes *Student_Name*, *Date_Taken* and *Outcome* in common. In order to retrieve information from frame instances, the user represents his request

[1]This is a deviation from the *relation* [99] of the classical relational model, in which a relation is associated precisely with one schema.

in a formal query. For example, the formal query for finding all the students who passed the qualifying examinations in the Spring and Fall of 1990 is given as follows:

SELECT Q.E.(Y).*Student_Name*

FROM Q.E.(Y)

WHERE

(Q.E.(Y).*Date_Taken* = "*Spring* 1990" **OR**

Q.E.(Y).*Date_Taken* = "*Fall* 1990") **AND**

Q.E.(*Y*).*Outcome* = "*Pass*";

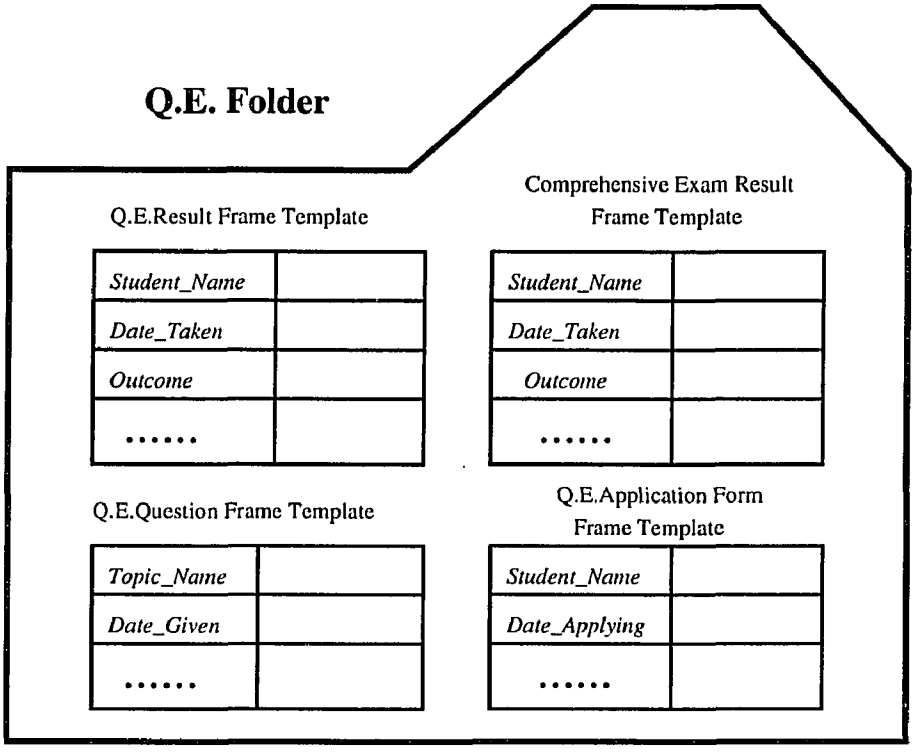

Figure 1.1 A Folder Containing Frame Instances Regarding Qualifying Examinations

In this query, the name of the folder Q.E. is explicitly specified from where the information will be searched. But the query is considered to be *incomplete* with respect to Q.E. folder, since the frame template Y containing the attributes

Student_Name, *Date_Taken* and *Outcome*, is not explicitly specified. Y in this case could be either one of the frame templates Q.E.Result or Comprehensive Exam Result, because both have these attributes. However, the request here is to find out those students who passed the qualifying examination in the Spring and Fall of 1990, and not those who passed the comprehensive examination on the specified dates.

In general, the explicit specifications of the folders, frame templates and attributes ensure that the system will retrieve precise information (i.e., frame instances of the frame templates as types from the various folders). But instead of putting a burden on the user to be responsible for giving the explicit specifications with great difficulties, he must be allowed to use variables to specify folders (the location of frame instances to be retrieved), frame templates (the type of frame instances to be retrieved) and attributes (some properties of these frame instances).

If the user uses Qualifying Exam in place of Q.E. (which is the precise keyterm for the name of the folder in which the query is to be applied), then this query is considered to be *imprecise*. Furthermore, in order to represent his request as a formal query, the user needs additional information about the qualifying examination, such as whether Qualifying Exam is the name of a folder or frame template, any frame templates related to the qualifying examination, any attributes and their domains for describing the results of the qualifying examination, the precise keyterms for folders, frame templates and attributes, and so forth. Such information is needed to formulate a complete and precise query. In reality, it would be a great advantage if a system would provide the user with the capability of entering a *vague query* such as "What is Qualifying Examination?". This vague query can be specified as

TOPIC Qualifying Exam

Assume that the response of the query for finding all the students who passed the qualifying examinations in the Spring and Fall of 1990 is an empty answer. Obviously, this empty answer is a meaningless response to the user. There can be

three interpretations to such response. First, the response can be interpreted to be a genuine one. This would mean that indeed several students took their qualifying examination in the Spring or the Fall of 1990 but none of them passed it. On the other hand, the query may reflect an erroneous presupposition on behalf of the user. The empty answer is also yielded because either no student took the qualifying examination or there was no qualifying examination held in the Spring and Fall of 1990. Therefore, it is essential for a system to provide the user with *meaningful responses.*

1.2.1 The Retrieval Mechanisms

In TEXPROS [107], the retrieval system is capable of processing incomplete or vague queries and providing meaningful responses to users when empty answers arise. The design of the retrieval system is highly integrated with various mechanisms for achieving these goals. First, there is a query transformation mechanism which consists of context construction and algebraic query formulation modules. Given an incomplete query, the context construction module searches the system for the required terms and constructs a query that has a complete representation. This resulting query is then formulated into an algebraic query. Second, in practice, the user may not have a precise notion of what he is looking for. We employ a browsing mechanism for such situations to assist the user in the retrieval process. Third, if the result of a query is an empty set, a generalizer mechanism is used to give the user more cooperative responses.

To accomplish these goals, the system needs to store the knowledge about the database. Knowledge representation and repository have been explored in many systems (e.g., [10, 29, 57, 69]).

1.2.2 The System Catalog

We employ a system catalog to store the information used for retrieval. The system catalog (or the data dictionary) is an important facility which provides the capability of managing and maintaining the consistency and integrity of the data stored in the database. In TEXPROS, an integrated system catalog provides a centralized retrieval environment for processing incomplete and vague queries in addition to providing an environment for processing complete queries and retrieving the meaningful information about the entities of the database. In addition to reflecting the meta-data of the document filing organization, the system catalog also includes a thesaurus[2]. The thesaurus comprises three major components. The first component contains synonymous keyterms. The second component describes the terms that have semantic associations with keyterms. The third component describes the associations of the keyterms in terms of folders, frame templates and attributes. Since the user can query the system catalog, we organize the system catalog as a special kind of a folder which mimics the document filing organization at the system level. This provides a natural and consistent operational approach for the user's environment.

1.3 Organization of the Dissertation

The remainder of this dissertation is organized as follows: Chapter 2 contains a survey of research which is related to my work. Chapters 3 through 9 present my proposed research work. In Chapter 3, the overall architecture of the proposed retrieval system is described. This chapter informally describes the scenario that underlies the formal treatment of the retrieval model. Chapter 4 presents the system catalog which is utilized during the retrieval process. The system catalog is a self-contained data dictionary which provides a centralized retrieval environment for

[2]A set of concepts in which each concept is characterized by hierarchical, synonymous, horizontal, and other relations [77].

processing incomplete and vague queries. In chapter 5 the query transformation mechanism is discussed. Chapter 6 and Chapter 7 present an intelligent browser and an enhanced generalizer, respectively. The browser enables the user to gain knowledge about the entities stored in the database. The generalizer is utilized to provide the user with meaningful and cooperative responses as interpretations to empty answers by looking into the generalizations of any given failed queries (i.e., with an empty answer) which are derived by applying both the folder and type substitutions and weakening the search criteria. Chapter 8 and Chapter 9 discuss an efficient way for determining a meaningful and cooperative response of any given failed query. The two chapters present a methodical approach to reducing the search space of generalized subqueries by analyzing the results of executing generalization and then by efficiently applying the possible substitutions to generate a small subset of relevant subqueries. Finally, Chapter 10 summarizes the dissertation and discusses some ongoing research topics that are related to the work in this dissertation.

CHAPTER 2

MOTIVATION AND RELATED WORK

This chapter discusses work related to my research, that has been done in the areas of query formulation, incomplete and vague query retrieval system and the representation of meta knowledge and domain knowledge in retrieval systems.

2.1 Query Formulation

Many Database Management Systems provide the facilities to assist the users in formulating their queries. Research is proceeding in many directions.

- Systems that provide better interfaces to the user.

  QBE (Query-by-Example)[118] is a successful query system for relational databases. The visual forms utilized in QBE can help the user describe a simple query. However, it is very difficult for the novice users to use these forms to formulate a complex query. Campbell et al.[7] defined a query language whose theoretical foundation is based on the ER algebra (similar to the algebra in [71]), in which users graphically manipulate entity-relationship (ER) diagrams to formulate queries. Each diagram represents a partial query which is particularly helpful in formulating ad hoc queries. The burden here is that the user needs to understand and remember the algebraic operators as he graphically specifies a path in the ER diagram. Wong and Kuo [111] investigated the difficulty in using and understanding query languages. They point out that (1) the user has to remember too many things as the database has a very complex schema; (2) the language lacks meta-data browsing facility; and (3) the user can not get feedback during query processing. Instead, they created a graphical user interface that allows the users to formulate their queries in a piecemeal fashion with feedback of partial results available to

them at any time. Their facility provides a mechanism that can guide and encourage the user to explore and browse the meta-data to obtain a general view of the database and select matters that are of interest. However, this facility only provides menus, examples, illustrations and help messages at the stage of query formulation. The user has to traverse a network and select a path himself.

- Systems that use natural language processing techniques to select index terms. Integrating natural language interfaces into database query systems has gained some attention. Bouzeghoub and Metais [4] designed the SECSI system, in which users' requests are expressed in natural language. The system translates the natural language into internal semantic network descriptions, creates a relational database schema from the semantic network, and performs a normalization process on the schema by evaluating a knowledge base. Rolland and Proix [78] created the OICSI system which can generate a conceptual schema of an information system from natural language descriptions. A bottleneck in these systems, however, is the requirement of natural language processing. Some of the criticisms of natural language processing have concentrated on the high cost of translating natural language query expressions into internal semantic descriptions.

- Systems that build knowledge bases from document contents.
  Jakobson et al. [43] developed a knowledge-based database retrieval system, called intelligent database assistant, to help the user in database retrieval. They proposed a system, called FRED, which gives users substantial help in query formulation, database selection and data interpretation. RABBIT [94] is a database front-end that utilizes an intelligent database assistant. It is a menu-based user interface which provides an interactive database query constructing

facility. KARMA [3] is another knowledge-based assistant which utilizes a menu-base system for the novice user. To achieve the high performance of query-by-reformulation, Wu and Ichikawa [112] provided a query guiding facility, called KDA, which has several kinds of skeletons to guide users in performing retrieval actions, such as forming a query, refining previously formed queries and modifying misconstructed queries. KDA is based on a semantic network transformation approach that translates a semantic network description into a relational database schema description.

- Systems that employ automatic query formulation

  Korth et al. [48] discussed System/U, a relational DBMS which is based on the universal relation assumption. The System/U relieves the user from the responsibility of navigating the database relations. Instead, the user relies on the predefinition of schematic constructs called maximal objects. Other related efforts based on the universal relation assumption can be found in [47, 53, 100].

  Motro[64] proposed a query interpreting system based on the automatic inference of the connections required to answer a query. The system provides an uniform treatment of data and metadata, so that the user does not need to distinguish between them. The user specifies his requests using tokens. The system interprets the tokens into a proper query by following a set of algorithms. However, the user can not represent more information (such as the relationships between tokens) in his query. This increases the ambiguity of interpreting the queries. Other approaches for automatic query formulation have been discussed in [31, 33].

2.2 Incomplete and Vague Queries

A considerable amount of research has focused on issues which represent imprecise data in the database (e.g., [27, 28, 30, 49]), and imprecise or vague requests to retrieve data (e.g., [23, 68]). Several representations for imprecise data have been suggested. These include "fuzzy" values [115], values accompanied by certainty factors [98] and null values [42]. So far, three basic approaches for processing vague queries have been proposed.

- The VAGUE system described in [67] is based on the vector space model. For each attribute from a vague condition specified in the query, the user may choose between a number of different metrics for the comparison of attribute values with the corresponding value from the query. Then the distance between the query and a database object is computed as a function of the distance for the different query conditions. Motro [65] classified user's requests into two kinds: (1) a specific request which is concerned only with data that matches it precisely and (2) a goal which is concerned with data which is close to the target. He extended the relational database model to support goal queries. The concept of distance between data values is defined and is incorporated into relational systems. The typical query language QUEL is extended to express goals. The system is capable of answering questions with information which is similar to the information requested.

- Vague queries have also been discussed in the context of fuzzy systems (e.g., [5, 75, 116]). The formal aspects of these works are based on the theory of fuzzy sets[1]. Informally, a fuzzy set is a class in which the distinction from membership to non-membership is vague rather than crisp and precise. Prade and Testemale [75] discussed the representation of incomplete and uncertain information by

[1]Formal definitions can be found in [90, 113].

means of possibility distributions[2]. Zemankova [116] demonstrated the fuzzy set theory as a suitable framework for the representation and manipulation of certain information in databases.

Buckles and Petry [5] extended the relational model to take into account nonprobabilistic uncertainties. Here, relations are extended to allow set-valued domain elements. Each domain element has an associated similarity matrix that assigns to each pair of domain elements, a value between 0 and 1.

Some of the criticisms of fuzzy set theory concentrate on the subjectivity of assigning membership functions to concepts [115].

- Recently, a probabilistic model for vague fact retrieval has been developed [28]. A set of conditions in a user's query can be either text conditions or fact conditions. Fact conditions can be interpreted as being vague, thus leading to nonbinary weights for fact conditions with respect to database objects. In the probabilistic approach, imprecise or missing attribute values can be stored as probability distributions over the set of possible attribute values. The system integrates text and fact retrieval by regarding both conditions relating to text or facts as being vague. Another system that combines vague fact and text retrieval is the office information system described in [19].

2.3 The Representation of Meta-data Knowledge and Domain Knowledge in the Retrieval System

The system catalog (or the data dictionary) is an important facility for managing and maintaining the consistency and integrity of the data stored in the database. Date [22] discussed an INGRES system catalog, which is a repository for information concerning various objects that are of interest to the system itself, such as base tables, indexes, forms, reports, access rights, integrity constraints, and so on. Davis

[2]They propose a model based on possibility theory introduced by Zadeh [114].

and Bonnell [24] described an approach, referred to as EDICT, creating an enhanced relational data dictionary which represents the high-level semantic information about the enterprise whose data is stored as tables in the database. EDICT provides a centralized management environment for maintaining information about the data in the database relations. Sibley [85] proposed an active and extensible dictionary system in which the meta-database is stored to completely control the database management system.

With the integration of database management systems and information retrieval systems, it is desirable to develop a mechanism that provides a generalized retrieval facility. Saxton et al.[80] and Croft[20] proposed that the introduction of the domain knowledge into a document retrieval system would increase the effectiveness of retrieval. Morgenstern [62] discussed the role of constraints in database and knowledge representation. He proposed that the similarities between database schema and knowledge representation frameworks may help to extend the semantics expressible in schema. Current system catalogs (or data dictionaries), however, are not used to store domain knowledge.

A number of information retrieval systems employ additional mechanisms to store the domain knowledge. Siegel and Madnick [86] described a rule-based approach to semantic specification that can be used to establish semantic agreement between a database and an application. Fikes and Kehler [26] used a frame-based representation to store concept descriptions. This representation combines and generalizes aspects of the representations used by Shoval[84] and Tong[93]. Schauble[82] proposed a thesaurus based concept space which would provide adequate term dependencies. Chen and Dhar[14] identified three types of knowledge which are necessary to perform a successful retrieval. These include: the subject area knowledge, the classification scheme knowledge, and the system knowledge. They proposed an automatic process of generating the semantic network knowledge base from an existing thesaurus (LCSH

Handbook). Smith et al.[87] analyzed several thesaurus systems (such as, [25, 73, 102]) and proposed that thesauri may contain certain types of knowledge that must be dealt with in designing an intelligent retrieval system.

CHAPTER 3

OVERALL ARCHITECTURE OF RETRIEVAL SYSTEM

Figure 3.1 illustrates the overall architecture of the retrieval system in TEXPROS, which is capable of processing incomplete or vague queries and providing semantically meaningful responses to users. Upon receiving a query from a user, the parser first checks the input query to determine whether it is a formal query or a vague query. Specifications of formal and vague queries are given, respectively, in the top and bottom part of Figure 3.2.

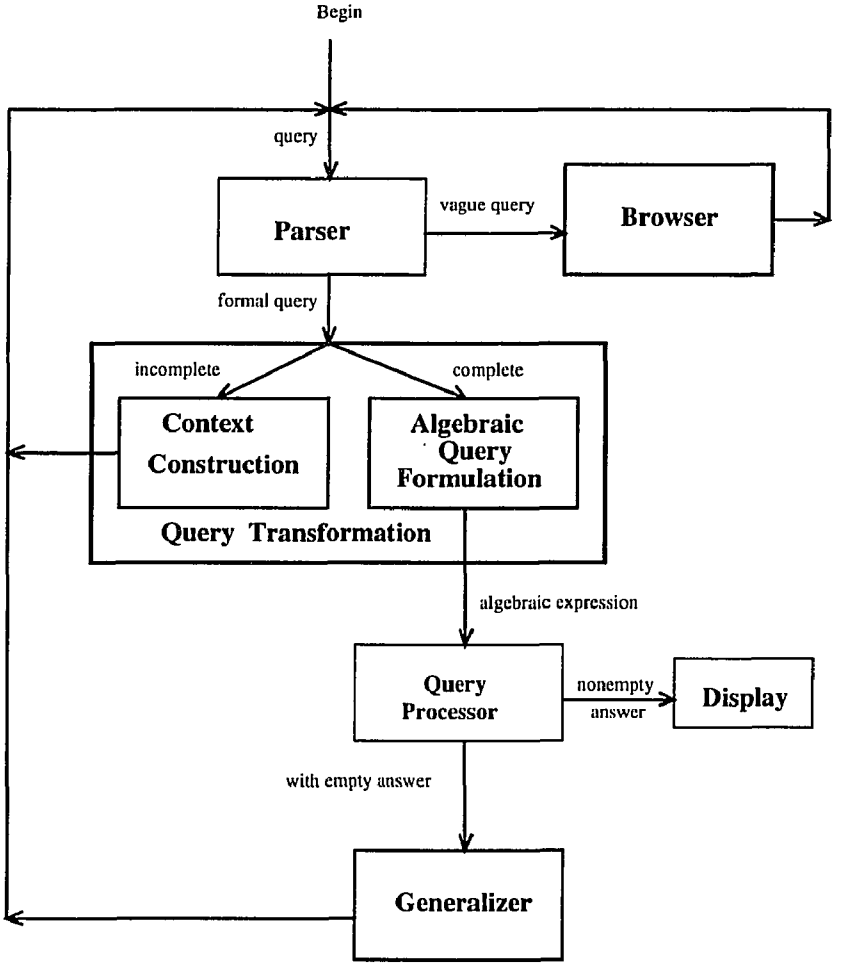

Figure 3.1 Overall Architecture

If the user does not have any idea of how to specify a formal query for his request, the "TOPIC" part as shown in Figure 3.2 will be used to describe his retrieval goal. An example is given in Figure 3.4. The vague query is then passed to the browser, which goes through the system catalog looking up relevant information (i.e. all frame templates possibly related to the user's request), and possible repositories of information attributes to describe the properties of the data to be retrieved. Vague queries can be entered to the system until sufficient information is obtained to the extent that the user is able to use this information to construct a formal query for his request.

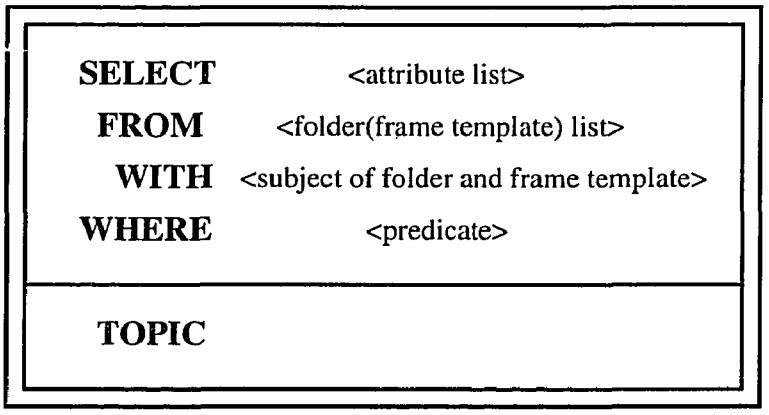

Figure 3.2 Query Interface

Once the input query is stated formally according to the specifications (an example is given in Figure 3.3), the query is transferred to the query transformation mechanism. The objective of the query transformation is to transform a formal query into a set of algebraic queries, which are to be processed by the query processor to assist in answering the corresponding user's original query. To accomplish this objective, the formal query is first examined to determine whether it is complete. An user's query is said to be complete if each term (called keyterms in TEXPROS) appearing in the query is consistent with the index term which exists in the database,

and no variables (such as "X" and "Y" in Figure 3.3) are used to specify any term in the user's query. Otherwise, the query is said to be incomplete. The complete query is directly passed to the algebraic query formulation mechanism, which eventually produces a corresponding set of algebraic queries. Given an incomplete query, a complete query is generated by using the context construction mechanism.

QUERY1: Find all the students who passed Q.E. in Fall 1990 or Spring 1990.

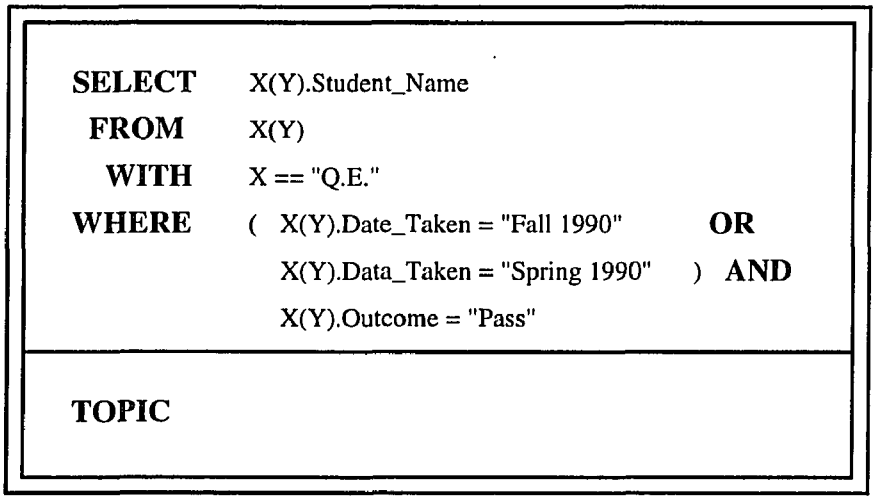

Figure 3.3 An Example of the Formal Query

The query processor executes the set of algebraic queries after its formulation. When processing of queries fails by responding with an empty answer, possibly without any semantical meaning to the user, the original query is passed to the query generalizer to produce cooperative explanation for the empty answer.

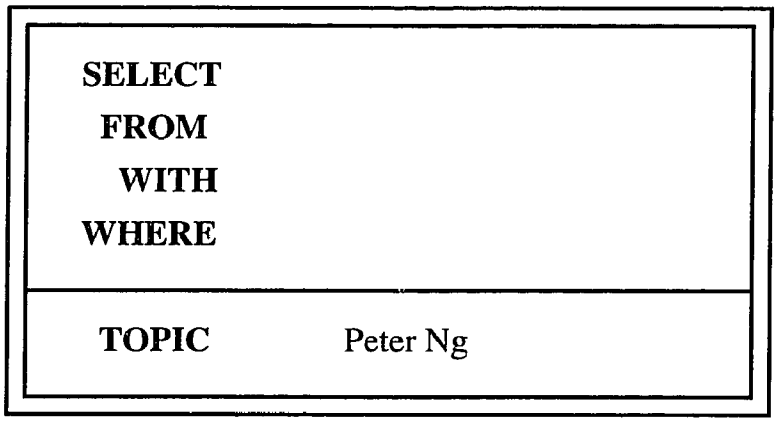

Figure 3.4 An Example of the Vague Query

CHAPTER 4
SYSTEM CATALOG

In TEXPROS, an integrated system catalog provides a centralized retrieval environment for processing incomplete and vague queries. The system catalog presents the information in a form which can be incorporated directly into the database system of TEXPROS. Since the uniform representation of the system catalog and the database itself (e.g., frame instances, the synopses of the documents) is adopted, the user can retrieve the information in the system catalog using the same query format to retrieve any general frame instances in the database. The details of retrieving information from the system catalog are provided in Appendix B.

4.1 Formalism of the System Catalog

We proceed to formally define the system catalog as follows:

Let $\mathcal{A} = \{\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_n\}$ be a finite set of attributes. Let $\mathcal{D} = \{\mathsf{D}_1, \mathsf{D}_2, \ldots, \mathsf{D}_n\}$ be a finite set of (not necessarily distinct) domains. Let *dom:* $\mathcal{A} \longrightarrow \mathcal{D}$ be a total function which associates each attribute $\mathsf{A} \in \mathcal{A}$ with a domain $dom(\mathsf{A}) \in \mathcal{D}$. We define a system frame template $\mathsf{SF} = \{\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_m\}$ as a finite set of attributes where $\mathsf{A}_i \in \mathcal{A}$, $1 \leq i \leq m$. Let $\mathsf{SF} = \{\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_p\}$ be a system frame template. A system frame instance *sfi over* SF is a finite set of attribute-value pairs $\{< \mathsf{A}_1, V_1 >$ $, < \mathsf{A}_2, V_2 >, \ldots, < \mathsf{A}_p, V_p >\}$, where $\mathsf{A}_j \in \mathsf{SF}$, and $V_j \subseteq dom(\mathsf{A}_j), 1 \leq j \leq p$. The set of all system frame instances reflects the state of the document filing organization. Let $\mathsf{SFI} = \{sfi_1, sfi_2, \ldots, sfi_q\}$ be the finite set of system frame instances reflecting the state of the filing organization. The system catalog is a finite set of subsystem folders $\mathsf{SC} = \{\mathsf{sf}_1, \mathsf{sf}_2, \ldots, \mathsf{sf}_r\}$ where each $\mathsf{sf}_j \subseteq \mathsf{SFI}, 1 \leq j \leq r$. All the system frame instances in a subsystem folder sf_j are over the same frame template SF, denoted as $\mathsf{SF}(\mathsf{sf}_j)$. We also use the notation $\mathsf{sf}(\mathsf{SF})$ or simply sf to denote a subsystem folder sf, in which it contains frame instances of the system frame template SF as type. The

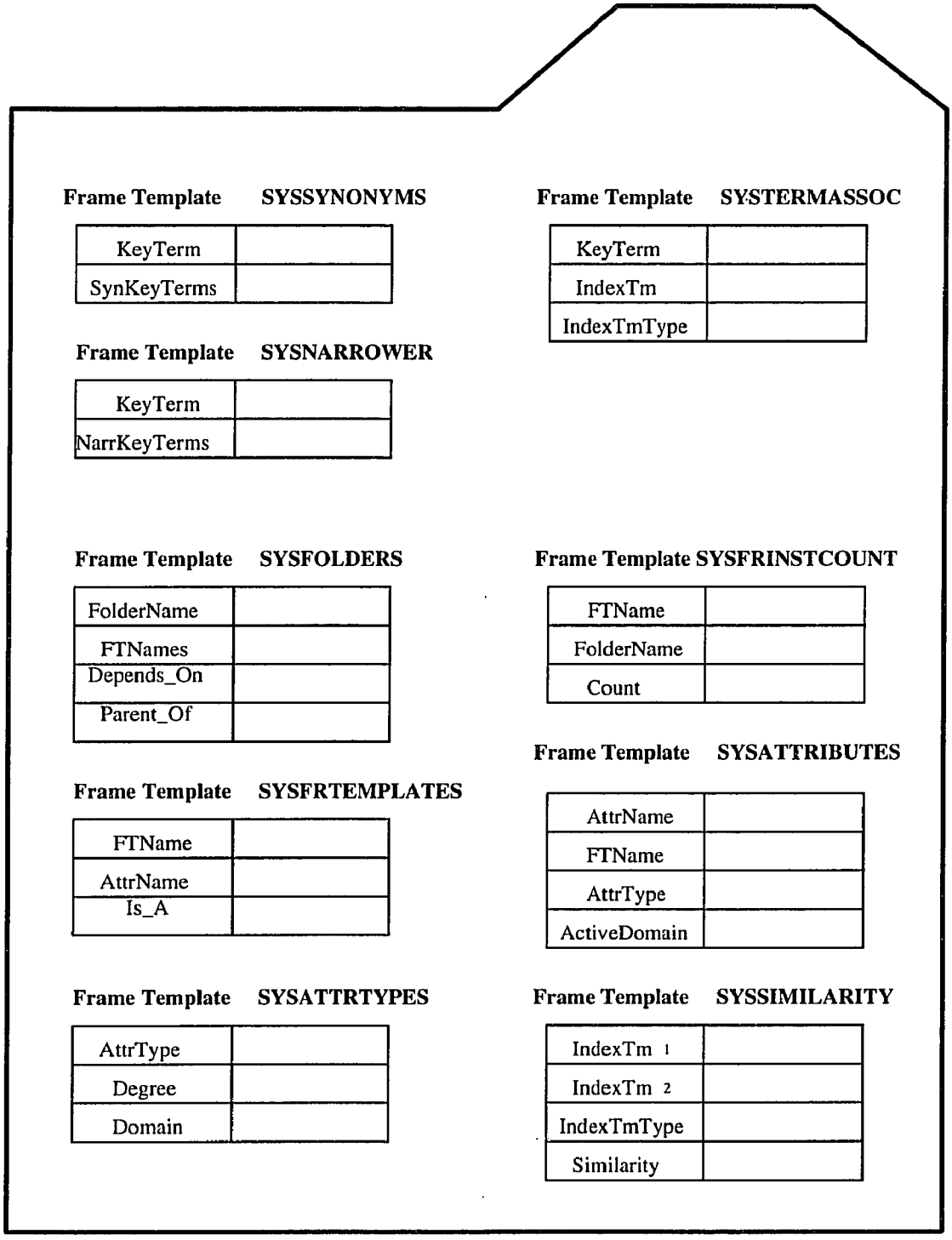

Figure 4.1 A System Catalog Structure

notation $\mathcal{SYSCATALOG}(\mathsf{SF})$ is used to restrict the system catalog to the system frame template SF.

Let $sfi = \{< \mathsf{A}_1, V_1 >, < \mathsf{A}_2, V_2 >, \ldots, < \mathsf{A}_p, V_p >\}$ be a system frame instance. Let X be any subset of $\{\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_p\}$. The X value of sfi, denoted by $sfi(X)$, is the system frame instance obtained by deleting those elements $< \mathsf{A}_j, V_j >$ from sfi where $\mathsf{A}_j \notin X$. If X consists of a single attribute, say A, then $sfi(X)$ is simply written as $sfi(\mathsf{A})$. (In this case, we use the notation $sfi(\mathsf{A})$ to denote the value V in the attribute-value pair $< \mathsf{A}, V >$.) Figure 4.1 depicts a system catalog structure which comprises the set of system frame templates. We expound on each of them in Appendix A.

4.2 The Novelty of the System Catalog in TEXPROS

The novelty of this system catalog is that not only it reflects the actual meta-data of the document filing organization, but also includes a thesaurus. Furthermore, the use of the concept of frame templates, frame instances and folders at the system and operational levels provides a consistent view to the user of his/her personal TEXPROS. At the operational level, the concept of frame templates is used to form the document type hierarchy for classifying the given documents; the concept of frame instances describe the synopses of documents pertaining their significance to the user; and the concept of folders containing frame instances of various types is used to describe a logical file structure of the document file organization. Similarly, at the system level, the concept of system frame templates is used to classify the information contained in the system catalog; and the frame instances describe the synopses of the information regarding the folder organization, document classification (in terms of frame templates) and keywords that will be used by the user at different times. This consistent approach to describing the operational knowledge of the environment, where the documents are reposited, and the knowledge about documents, structures

and contents (in synopsis form), provides the user with an ease of classifying, filing and retrieving documents.

4.3 System Catalog Management

The system catalog describes the document filing organization and document classification at system level. It is managed dynamically during document classification and filing.

We define a set of primitive functions that manage the system catalog as triggers. For instance, during document classification, if a user selects a frame template which does not exist in the system, the function **InsertFrTemplate**(FTName, AttrName,Is_A) is invoked. (This function will append a new frame template containing relevant information about the name of the frame template, its attribute names, and its Is_A relationship in the document type hierarchy as a system frame instance of $\mathcal{SYSCATALOG}$(SYSFRTEMPLATES). During document filing, if a user creates a folder which does not exist in the system, the function **InsertFolderName**(*folder*) is invoked. (This function will create a system frame instance *sfi* of SYSFOLDERS type in the $\mathcal{SYSCATALOG}$(SYSFOLDERS), in which *sfi*[**FolderName**] is *folder*, the name of a folder, and the values for the other attributes are *NIL*).

We design various algorithms to update the system catalog using these primitive functions. For instance, in the filing organization, it may be desirable to distribute a set of frame instances fi_s from a folder fd_p into a folder fd_c. The sequence of functions is invoked as follows:

For each fi in fi_s

Do $ft :=$ **DetermineFT**(fi);

InsertFRINST(ft,fd_c,1);

If ft does not appear in the **FTNames** of the frame

instance of SYSFOLDERS type associated with fd_c

then **InsertFTName**(fd_c,ft);

If **CheckFICount**(ft,fd_p)= 1

then **DeleteFTName**(fd_p,ft);

DeleteFRINST(ft,fd_p,1)

end

All the algorithms for system catalog management can be found in Appendix C.

CHAPTER 5
QUERY TRANSFORMATION

In this chapter, an automatic method to refine and formulate the user's query into an algebraic query is proposed. In TEXPROS, the formal query is specified in SQL-like syntax. The examples of the formal queries are shown in Figure 3.3 and Figure 5.1 . The user specifies the names of the folders and frame templates required to process the query in the "FROM" clause, the names of attributes whose values are to be retrieved by the query in the "SELECT" clause, and the predicate that identifies the frame instances to be retrieved by the query in the "WHERE" clause. If the user does not know the name of any of these terms, he can use variables instead (e.g. the "X" and "Y" in Figure 3.3) and then specify the subjects of the corresponding folders or frame templates in the "WITH" clause if he knows. The system can infer all the variables to the proper names of folders, frame templates or attributes by retrieving the system catalog. Intuitively, the user can express his queries by entering any information he knows freely. Therefore, the user focuses on the general idea of his queries rather than trying to remember a symbolic language or the precise names of individual entities in system (or to look up the system catalog to find them), such as, the names of the folders, frame templates and attributes. The terms for specifying the names of folders, frame templates and attributes in a user's original query are called keyterms in the system catalog. These keyterms may not be the index terms which are used in the database. The objective of the query transformation described in this chapter is to assist users in finding the appropriate index terms, which are a set of folders containing the frame instances to be retrieved, a set of frame templates which are the types of the frame instances to be dealt with, and a set of predicates to be satisfied by these frame instances, corresponding to those given keyterms from the user's query; and then apply the algebraic operators to the index terms to generate the algebraic queries.

QUERY2: Find all the students who were admitted in Fall 1990 and passed Q.E. before Spring 1992.

```
SELECT   Q.E.(Q.E.Result).Student_Name
FROM     Q.E.(Q.E.Result)
         X(Admission_Acc_Letter)
WHERE    X(Admission_Acc_Letter).Date = "Fall 1990"          AND
         Q.E.(Q.E.Result).Date_Taken <= "Spring 1992"                AND
         Q.E.(Q.E.Result).Outcome = "Pass"              AND
         Q.E.(Q.E.Result).Student_Name = X(Admission_Acc_Letter).Name
TOPIC
```

Figure 5.1 An Example of the Formal Query

5.1 Context Construction

The context construction mechanism generates a complete query from the user's incomplete query (i.e., the construction of index terms stored in the database from the set of keyterms that appear in the user's query). A user's query is called an incomplete query if it contains imprecise terms (non-index terms), subject terms (the subjects of folders or frame templates), or missing information (unknown index terms). A mapping of the keyterms into a set of appropriate index terms can be created through interaction with the system catalog. (The details of algorithms to retrieve the system catalog are described in Appendix B.) In fact, the context construction plays the role of a *search computerized intermediary system* [72] for information retrieval, which provides significant support for processing the incomplete query. The procedure of context construction is shown in Figure 5.2.

We develop a search strategy for finding the appropriate index terms, which comprise the search space, corresponding to the keyterms in the user's query. Also,

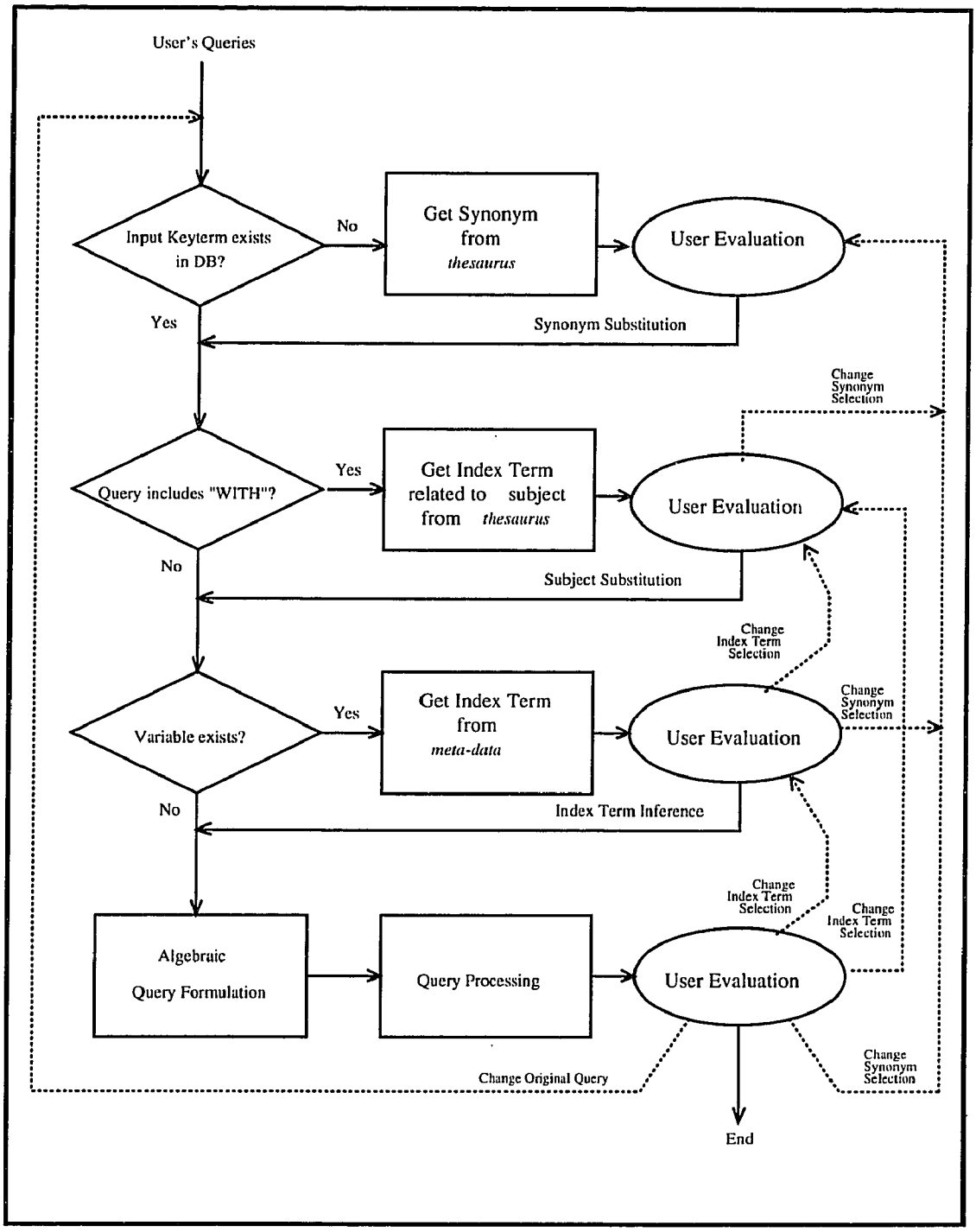

Figure 5.2 Query Transformation

we develop an interactive evaluation strategy for ensuring the precision of the search space.

- Search Strategy

  - Synonym Substitution: Processing Imprecise Terms.

    In the system catalog, the system frame instances of type SYSSYNONYMS contain information about synonymous keyterms that are relevant to the user. Associated with the keyterms, the frame instances of the type SYSTERMASSOC specify index terms to be the names of folders, frame templates and attributes. If a term is used by the user in his query, the synonym substitution determines the keyterm and the corresponding index term for the synonymous term by searching through the system frame instances of the types SYSSYNONYMS and SYSTERMASSOC, respectively. For example, looking for some information about Peter, the user may enter "Peter Ng" as the name of the folder. However, there may be no folder name labeled "Peter Ng" in the system. Through the synonym substitution, the system obtains the folder "Peter A. Ng" by retrieving the system frame instances of the types SYSSYNONYMS and SYSTERMASSOC.

  - Subject Substitution: Processing Subject Terms.

    In the system catalog, the system frame instances of the type SYSTERMASSOC contain the domain knowledge that folders and frame templates are labeled according to the subjects that they cover or touch upon. If the user does not remember the precise name of a folder or frame template, he can express the information needed in terms of concepts, denoted by the subject of the folder or the subject of the frame template. For instance, in Figure 3.3, X denotes the folder which may contain the frame instances

the user needs. X is specified to represent the subject "Q.E." in the **WITH** clause. When this query is executed, the system retrieves the system frame instances of the type SYSTERMASSOC to find the name of the folder X which deals with the subject "Q.E.".

- Index Term Inference: Processing Missing Information.

  In the system catalog, the system frame instances of the types SYSFOLDER, SYSFRTEMPLATES, and SYSATTRIBUTES contain the meta-data knowledge that describes the organization of the database in TEXPROS. In conventional database systems, the user is required to know the structure of the underlying schemas in detail to formulate his queries. However, in TEXPROS, the user does not have to enter complete information about the schemas; the system can infer the precise terms from the missing information by retrieving these meta-data from the system catalog. For Example, in Figure 5.1, X denotes the unknown names of the folders which contain the frame template "Admission_Acc_Letter". The system obtains the names of the folders X by using the following algorithm:

Algorithm: (Get folders from frame templates)

Getfd_fr_ft(ft_name)

begin

$f1 = \sigma_{FTNames \supseteq ft\_name}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;

$fds = \{sfi[FolderName] | sfi \in f1\}$;

for each $fd \in fds$ do

$FolderNames = fds \cup \text{GetPredecessor}(fd)$;

$f2 = \sigma_{Is\_A \supseteq ft\_name}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES}))$;

if $f2 \neq empty$ then

begin

$fts = \{sfi[FTName] | sfi \in f2\}$;

for each $ft \in fts$ do

$FolderNames = FolderNames \cup$ Getfd_fr_ft(ft)

end

return($FolderNames$)

end

GetPredecessor(fd)

begin

$f1 = \sigma_{FolderName=fd}(\mathcal{SYSCAT ALOG}$(SYSFOLDERS));

$fps = \{sfi[Depends\_On] | sfi \in f1\}$;

if $fps \neq empty$ then

$fd = fd \cup$ GetPredecessor(fps);

return(fd)

end

- Evaluation Strategy

In Figure 5.2, there are four ellipses representing the user's interaction with the transformation procedure. The procedure of the synonym substitution may return a collection of index term to the user. The procedure of the subject substitution may return a collection of names of folders or frame templates to the user. The procedure of the index term inference of the system may return a collection of index terms to the user. In these cases, the user is asked to determine whether the returned terms are the index terms he needs. For instance, these procedures return a collection of index terms which are either the names of the folders or the frame templates. The folders whose names are

the index terms may possibly contain the frame instances to be retrieved; and the frame templates with the index terms as their names are the possible types of the frame instances to be retrieved. The user is then asked to select a set of index terms for refining his query. The user is permitted to select an alternative set of index terms (represented as dashed lines in Figure 5.2), whenever he finds that the previously selected index terms are not correct. These selected Index terms will be the input of the algebraic query formulation phase. After query processing, a set of frame instances is returned to the user. If the user is not satisfied with the outcome, he is still permitted to select an alternative set of index terms or to modify his original query. Therefore, the system assists the user to confirm whether these index terms represent the folders and frame templates from which the frame instances are to be retrieved or synthesized.

5.2 Algebraic Query Formulation

In our system, an algebraic operator table (as shown in Table 5.1) containing the set of algebraic operators [61] is maintained. In the process of the context construction, a set of index terms, denoted by a set of folder names, frame template names, attribute names and attribute values, is obtained. Utilization of the algebraic operators to these index terms will generate the set of algebraic queries that can assist in answering the user's query.

For some sample queries, the following method can be used for the algebraic formulation.

- Let folders found in the context construction be $fd[1], fd[2], \ldots, fd[n]$.
  Let frame templates found in the folder $fd[i]$ be $ft[i,1], ft[i,2], \ldots, ft[i,m]$, $(1 \leq i \leq n)$.

Table 5.1 Operators of the $\mathcal{D}$-Algebra

Class	Operators	Type	Operands	Results
1	$\cup, \cap, -$	binary	folders	folder
2	π	unary	folder	folder
3	$\bullet$	binary	fr. instances	fr. instance
3	$\times, \bowtie$	binary	folders	folder
3	ρ	unary	folder	folder
4	σ	unary	folder	folder
5	$\eta_{\mathsf{A}}, \mu_{\mathsf{A}}$ (A is an attribute)	unary	folder	folder
6	$\gamma_{\mathsf{A}_\beta}$ (β is a subset of the component attributes of A)	unary	folder	folder
7	$\mathsf{count}_{\mathsf{A}}, \mathsf{sum}_{\mathsf{A}}, \mathsf{avg}_{\mathsf{A}}, \mathsf{min}_{\mathsf{A}}, \mathsf{max}_{\mathsf{A}}$ (A is an attribute)	unary	folder	NUM

Let predicates containing attributes found in $ft[i,j]$ be $p[i,j]$, $(1 \leq i \leq n,$ $1 \leq j \leq m)$.

Let predicates containing attributes found in $ft[i,j]$ and $ft[u,v]$be $p[i*j, u*v]$, $(1 \leq j, v \leq m,\ 1 \leq i, u \leq n)$.

The following cases may arise to produce a set of algebraic queries.

- For all the $p[i,j]$ $(1 \leq i \leq n, 1 \leq j \leq m)$, the following algebraic query is produced:

  $temp[i*j, i*j] = \sigma_{p[i,j]}(\pi_{ft[i,j]}(fd[i]))$.

- For all the $p[i*j, u*v]$ $(1 \leq j, v \leq m,\ 1 \leq i, u \leq n)$, the following algebraic query is produced:

  $temp[i*j, u*v] = \sigma_{p[i*j, u*v]}((\pi_{ft[i,j]}(fd[i])) \bowtie (\pi_{ft[u,v]}(fd[u])))$.

- For $temp[i*j, u*v]$ ($temp[i*j, i*j]$ is the special case) $(1 \leq j, v \leq m,$ $1 \leq i, u \leq n)$, the following algebraic query is produced:

  $temp\_result = \bowtie temp[i*j, u*v]$.

- The set of above queries is applied to the attributes in the **SELECT** clause.

  begin

  if $\nexists$ aggregate operator in the **SELECT** clause then

  $$Result = \pi_{AttributeNames}(temp\_result)$$

  else

  $$Result = aggrop_{AttributeNames}(temp\_result)$$

  end

5.3 Example

Here an example is given to illustrate an execution of the query transformation. The user's original query is shown in Figure 5.3, in which the user wants to find all the Ph.D students who passed the Qualifying Examination in the Spring of 1990. Assume that the user knows the folder *Q.E.*, from which the frame instances are to be retrieved, but he does not know the types of frame instances (that is, the name of the frame template). He uses *Date_Taken* and *Result* to express the names of attributes in the predicate.

- Context Construction.

  By following the procedure depicted in Figure 5.2, the user's original query is transformed to the complete query as shown in Figure 5.3.

  - Check whether the input keyterms, such as *Q.E.*, *Date_Taken* and *Result*, exist in the system by consulting the system catalog as follows:

    * $ef = \text{count}_{FolderName}(\sigma_{FolderName=Q.E.}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS})))$;
      The folder *Q.E.* is in the system since ef is not equal to zero.

    * $ac1 = \text{count}_{AttrName}(\sigma_{AttrName=Date\_Taken}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES})))$;
      The attribute *Date_Taken* is in the system since $ac1$ is not equal to zero .

* $ac2 = \text{count}_{AttrName}(\sigma_{AttrName=Result}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES})))$;
  The attribute *Result* is not in the system since $ac2$ is equal to zero.
* $ac3 = \text{count}_{AttrName}(\sigma_{AttrName=Student\_Name}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES})))$;
  The attribute *Student_Name* is in the system since $ac3$ is not equal to zero.

– Apply Synonym Substitution for *Result* by consulting the thesaurus in the system catalog. The system returns *Outcome*, which is the synonym of *Result*, to the user by using the following algorithm:
$f1 = \sigma_{SynKeyTerms \supseteq Result}(\mathcal{SYSCATALOG}(\text{SYSSYNONYMS}))$;
if $f1 \neq empty$ then
$y = sfi[KeyTerm]$ where $sfi \in f1$;

– Apply Index Term Inference for getting the names of the frame template by consulting the meta-data in the system catalog:
$y = \sigma_{FolderName=Q.E.}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;
$ft = \{sfi[FTName] | sfi \in y\}$;
A set of frame templates ft from the folder $Q.E.$ is obtained. The user is asked to select one of them. The user selects the name of frame template, $Q.E.Result$.

- Algebraic Query Formulation.
  By employing the algebraic operators, the system generates the following algebraic queries to assist in answering the user's query.
  $temp\_result = \sigma_{Date\_Taken=Spring1990 \wedge Outcome=Pass}(\pi_{Q.E.Result}(Q.E.))$;
  $Result = \pi_{Student\_Name}(temp\_result)$;

QUERY: Find all the Ph.D students who passed the Qualifying Examination in the Spring of 1990.

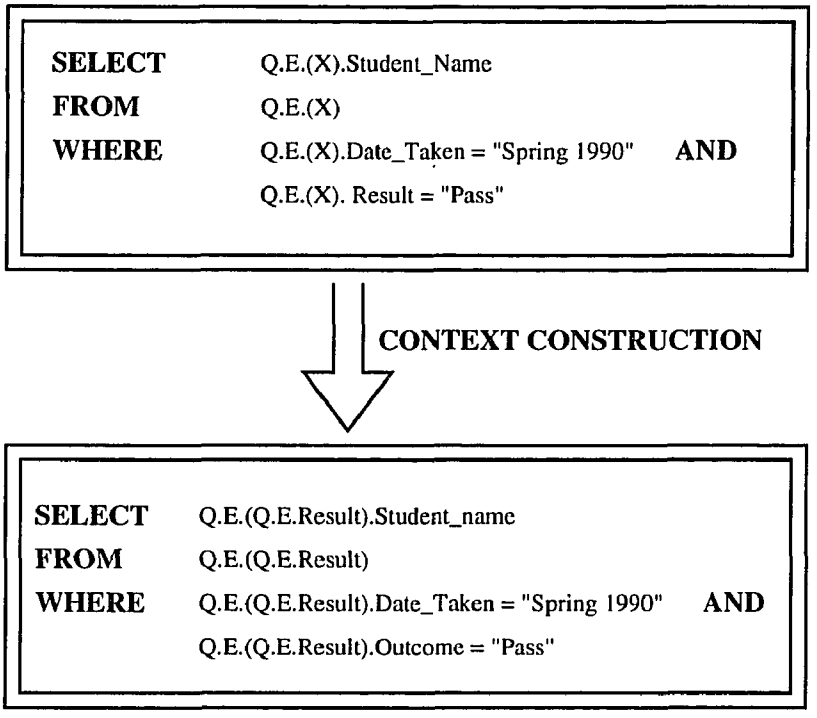

Figure 5.3 An Example of Context Construction Application

CHAPTER 6

BROWSER

In the previous chapter, we discussed an efficient and standard method for retrieving information from databases, which is called *systematic retrieval* [63]. The user presents his request in a formal query; and upon receiving this query, the system executes the query transformation to find, if necessary, the proper index terms corresponding to those given keyterms from the user query by retrieving the system frame instances in the system catalog, and then to generate the equivalent algebraic queries by applying the algebraic operator to these index terms. There are some situations, however, in which the systematic retrieval is difficult to achieve the objectives. For instance, the user may only have a vague retrieval target (e.g. What is Peter Ng?). Here, the user does not know exactly what kinds of information he needs until some kind of description is displayed to him. (The user needs to gain knowledge about both schemas and instances from the database.) In such situations, TEXPROS employs a browsing mechanism as a complementary retrieval method.

Several database management systems have provided the user with tools that allow users to explore their environment. Cattell [8] designed a browser for an Entity-Relationship database, which could display each entity with its context to the user by scanning a network of entities and relationships. D'Atri and Tarantino [23] pointed out the major limitations of most of the relational database browsers (e.g., SDMS[37], TIMBER[88]). The primary limitation is that the user is confined to a single relation at a time, and it is very hard to browse across relation boundaries. Motro [63] presents a browser, called BAROQUE, which supports inter-relation browsing by using network views of relational databases. BAROQUE needs the additional space to store the relational schemas and an item directory to support *access by value.* In TEXPROS, we create an object network to present the view of the schema (meta-data) of the database (about document type hierarchy and folder organization) and

the database itself (frame instances). However, all this information is incorporated in the system catalog. Therefore, the object network always represents a snapshot of a subset of the system catalog.

In the first part (section 6.1, 6.2 and 6.3) of this chapter, we define the object network, the architecture and the functionality of the browsing mechanism. The second part (section 6.4 and 6.5) discusses the different components of the browser. We conclude with some examples to illustrate how the mechanism works in section 6.6.

6.1 Object Network

In Figure 6.1, we describe each object in terms of schema elements (meta-data) and data elements. A database schema describes the structure of the database and a set of integrity constraints. In TEXPROS, this description includes the names of the folders along with their *depends_on* relationships, the names of the frame templates along with their *is_a* relationships, and the names of the attributes along with their attribute types.

As we discussed above, the user can obtain the information about the specific schema elements by retrieving the system frame instances in the system catalog using the formal query, just like retrieving any general frame instances in the database, since the uniform representation of the system catalog and database itself is adopted. However, it requires technical understanding of the data model of TEXPROS (i.e., the user needs a clear target for the retrieval). For instance, the user may want to know the names of all the frame templates in the "Assistants" folder. To avoid these requirements, we describe the information presented in the schema into an object network. As mentioned above, the way of representing the schema in the system catalog is the same as of representing the data in the database, and therefore the user is not required to distinguish between the schema elements and data elements.

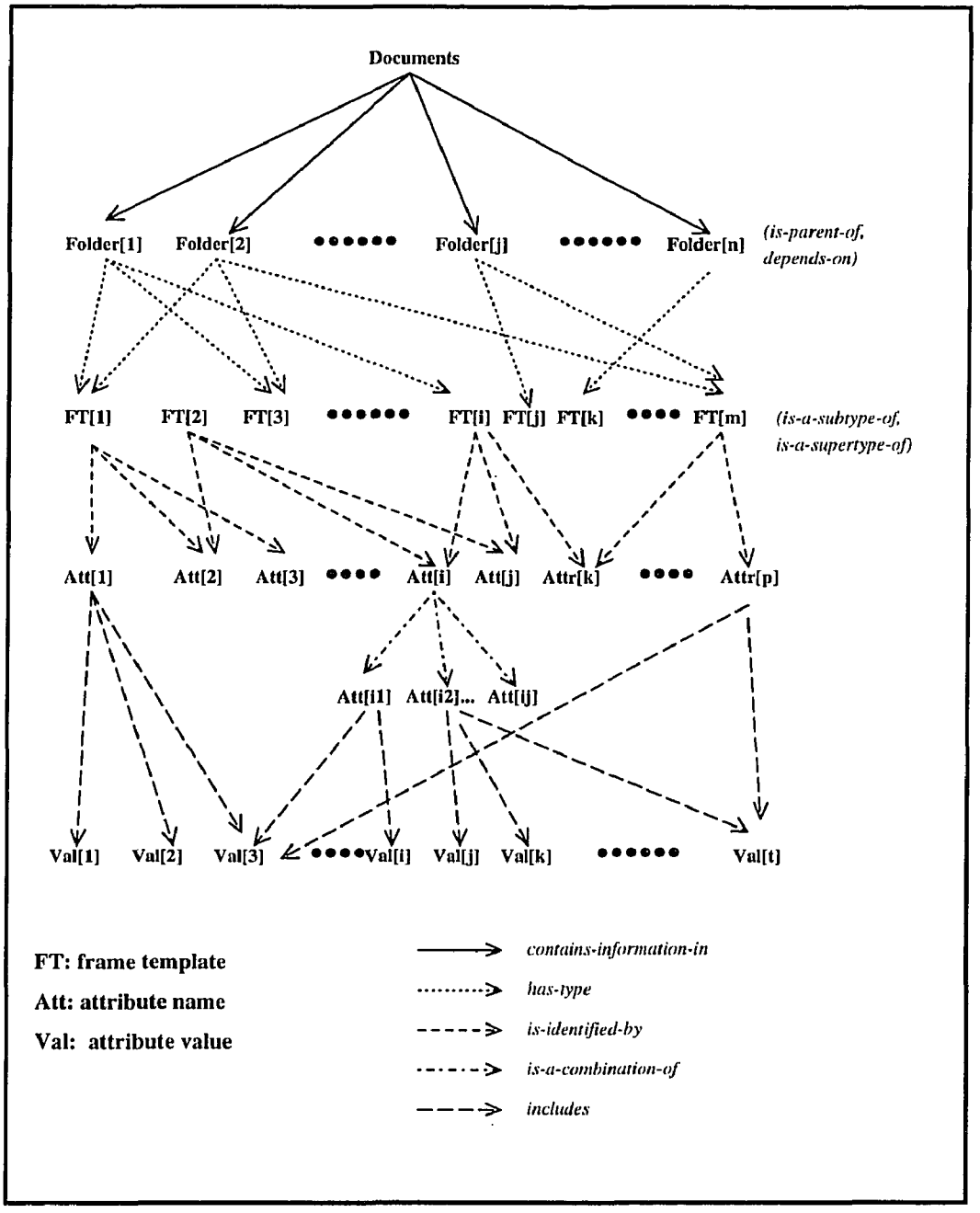

Figure 6.1 Object Network

- The schema elements in the object network.

  We represent the schema elements with four vertical levels in the object network: the documents in the database TEXPROS, the folders, the frame templates, and the attributes. Each element is represented by an object. The relationships between objects are described as follows: (1) the relationship *contains-information-in* relates the documents in the TEXPROS database to folders; (2) the relationship *has-type* relates every folder to its frame templates which represent the types of frame instances in the folder; (3) the relationship *is-identified-by* relates every frame template to its attributes; and (4) to the composite attributes, the relationship *is-a-combination-of* relates a composite attribute to each of its components.

- Dual model in the object network.

  We incorporate the folder organization (i.e. logical file organization) and document type hierarchy into the object network. To accomplish this, the object network is extended with the additional horizontal levels, which represent the relationship among folders and the relationship among frame templates. (1) The relationship *is-parent-of* relates every folder to its subfolders. The relationship *depends-on* relates every folder to its parent folders. These relationships are reflected in the folder organization. (2) The relationship *is-a-supertype-of* relates every frame template to each of its subtype frame templates. The relationship *is-a-subtype-of* relates every frame template to its supertype frame template. These relationships reflect the generalization and specialization relationships in the document type hierarchy.

- The data elements in the object network.

  In [63], the concept of *access by value* is proposed. This concept gives the user the capability of retrieving all the occurrences of an attribute value from the

database. The occurrences of an attribute value are in terms of attributes under which the given values appear. For example, the value *Jason* may appear in the database as a value of the attribute sender of a memo or the attribute author of a publication. In [63], an item directory is needed to store the mapping from the values into attribute names. In TEXPROS, all this information is stored in the system frame instances of SYSATTRIBUTES type in the system catalog $\mathcal{SYSCATALOG}$. Each of these frame instances over SYSATTRIBUTES describes not only the attribute names appearing in a specific frame template, but also the attribute types. The latter part of the information is helpful in the case that attributes with same name have different attribute types.

We present a view of the relationships between the attributes and the attribute values in the object network. The relationship *includes* relates every attribute to its values. Furthermore, the relationship between an attribute value and other values can be obtained only if they occur in the same frame instance. Formally, let $fi = \{< \mathsf{A}_1, V_1 >, < \mathsf{A}_2, V_2 >, \ldots, < \mathsf{A}_n, V_n >\}$ be a frame instance over frame template FT in the folder f. The following implied relationships are established: (1) the relationship *is-*A_1*-of-*FT*-in-*f*-having-*A_i relates the value V_1 to $V_i (i = 2, \ldots, m)$; and (2) the relationship *is-*A_i*-of-*FT*-in-*f*-having-*A_1 relates the value V_i $(i = 2, \ldots, m)$ to V_1.

6.2 Architecture of Browser

In TEXPROS, the database can be viewed as a network of objects, which consist of the schema elements and data elements. All the information, except the relationship among data elements, which can be obtained from the database itself, shown in the object network can be derived simply by retrieving information from the system frame instances in the system catalog, $\mathcal{SYSCATALOG}$.

The components of the browser are depicted in the Figure 6.2. When a user enters a vague query as a topic, the system looks up all its related information in the system catalog. The topic interpreter finds all the relevant objects by retrieving the system frame instances from the system catalog. The objects include all possible index terms (including the names of folders, frame templates, attributes, and values) and their relationships, which are pertaining to the topic specified in the vague query. And then the answers are combined to form an object network, along with some descriptions, which represents all the information pertinent to the selected topic. These description can be expressed in terms of the relationship *is-*A_i*-of-*FT*-in-*f*-having-*A_j for bringing together all the attribute-value pairs as a whole from the same frame instance. Therefore, the overall object network is not stored explicitly in the system. Only a portion (i.e., subgraph) of the object network for the vague query, dynamically constructed by accessing the system catalog, is returned to the user.

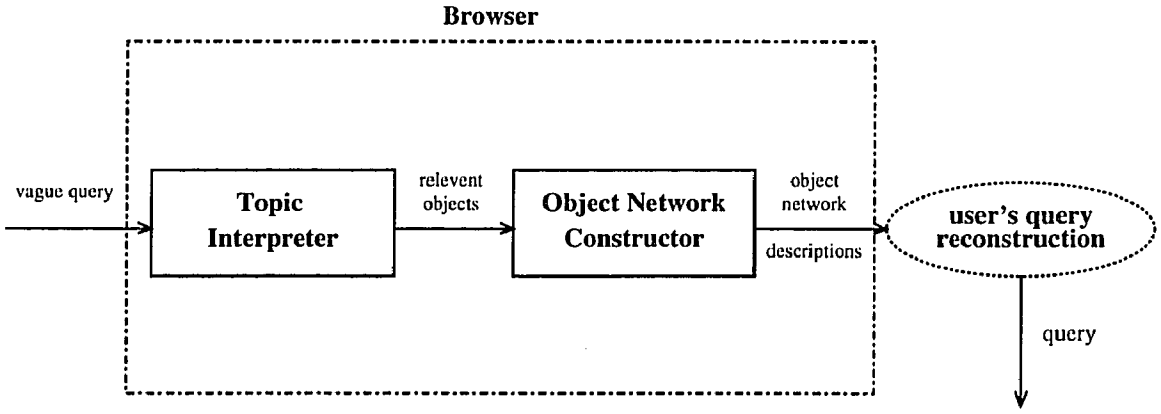

Figure 6.2 Architecture of Browser

In the system, there are two principal retrieval methods, querying and browsing. The user may select any object from the obtained object network to form the next browsing topic. Such vague queries can be repeatedly entered into the system until sufficient information is obtained to the extent that the user is able to use this information to construct a formal query of his request. The system is designed in such a way that the browsing and querying may be interleaved.

6.3 Browsing in TEXPROS

Using the query interface as shown in Figure 3.4, the user can enter any topic. By browsing through the system frame instances in the system catalog, the system is able to respond with an object network which represents all the information related to the topic.

If the topic entered by a user is a schema element, such as the name of a folder, the name of a frame template, or the name of an attribute, the system will return an object network in which the objects represent all the database definitions related to this topic. If the topic is a data element, such as an attribute value, the system will respond with a description which represents its relationships with other attribute values (i.e., they occur in the same frame instance), provided the information about the topic is stored as the frame instances in the system. Indeed the browsing method in TEXPROS supports the *concept retrieval* of some sorts.

We can extend our browser mechanism to accept more than one topic entered by the user. For each topic, there corresponds an object network with the necessary descriptions. The connectedness among the object networks depends on the relatedness of the corresponding topics. For simplicity, the relatedness of given topics is considered to be the same folder, frame template, attribute or value, and their relationships. The system attempts to find the relatedness among these topics. Several individual object networks, each of which is associated with a topic, are

constructed first. According to the user's request, the further process may involve two issues:

- How to connect these object networks into a connected object network.

  Since the object network for each topic is only a subgraph of the object network for the entire system (such as, the object network depicted in Figure 6.1), the system will return an object network to the user by connecting these subgraphs together, provided these subgraphs are "joint".[1] Since the object network for the entire system is a connected graph, the subgraphs, each of which is associated with a topic, can be eventually connected to form an object network by adding a large number of objects, possibly loosely related to the topics. To avoid this situation, the system will limit the number of objects to be added into the subgraphs. Therefore, there may exist several disjoint object networks for several unrelated topics which are entered by the user.

- What query can be formed from this connected object network.

  This issue can be resolved by observing the sequence of consecutive topics entered by the user since they need to know the prerequisite information to construct a formal query.

6.4 Topic Interpreter

The topic interpreter is used to interpret an input topic as objects in the system, and then retrieve other objects which are associated with them by accessing the system catalog and the database. The following algorithm, described in the form of algebraic expressions, provides an unified strategy for accessing "schema" and "data" from the system catalog and the database. The results will further be used to construct the

[1] "joint" means that they have common nodes or they will have common nodes after adding some other objects to the object networks. Two object networks have a common node provided their corresponding topics are related to each other, and the relatedness of topics is of the same folder, frame template, attribute or value, and their relationships.

object network which represents all possible objects and their relationships related to the topic.

Algorithm 6.1: (Check whether the *topic* in the query is a folder name, a frame template name, an attribute name or a value in the system; and then call their respective procedure. Otherwise, find its related index terms by looking into the thesaurus.)

BEGIN

$f_1 = \sigma_{FolderName=topic}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;

$f_2 = \sigma_{FTName=topic}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES}))$;

$f_3 = \sigma_{AttrName=topic}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES}))$;

$f_4 = \sigma_{ActiveDomain \supseteq topic}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES}))$;

case($f_1 \neq empty$) **CallFolder**(*topic*);

case($f_2 \neq empty$) **CallFrameTm**(*topic*);

case($f_3 \neq empty$) **CallAttribute**(*topic*);

case($f_4 \neq empty$) **CallValue**(*topic*);

case($f_i = empty$) **CallThesaurus**(*topic*)

END

CallFolder(fd)

(Get information related to the folder fd, such as, the parent(s) of fd, the subfolder(s) of fd, and the frame template(s) associated with fd.)

BEGIN

$f = \sigma_{FolderName=fd}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;

$fd_c = \{sfi[Parent\_Of] | sfi = f\}$;

$fd_p = \{sfi[Depends\_On] | sfi = f\}$;

$ft = \{sfi[FTNames] | sfi = f\}$;

OUTPUT$(fd,, fd_c, fd_p, ft)$

END

CallFrameTm(ft)

(Get information related to the frame template ft, such as, its attributes, its superclass(es) and subclass(es), and the folders associated with ft.)

BEGIN

$f = \sigma_{FTName=ft}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES}))$;

$att = \{sfi[Attr\_Name] | sfi = f\}$;

$ft_p = \{sfi[Is\_A] | sfi = f\}$;

$f' = \sigma_{Is\_A \supseteq ft}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES}))$;

if $f' \neq empty$ **then**

$ft_c = \{sfi[FTName] | sfi \in f'\}$;

$f'' = \sigma_{FTName=ft}(\mathcal{SYSCATALOG}(\text{SYSFRINSTCOUNT}))$;

$fd = \{sfi[FolderName] | sfi = f''\}$;

OUTPUT$(ft, ft_c, ft_p, att, fd)$

END

CallAttribute(att)

(Get information related to attribute att, such as, the frame templates including att, the folders associated with these frame templates, and the attribute type of att.)

BEGIN

$f = \sigma_{AttrName=att}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES}))$;

$(ft_s, type_s) = \{(sfi[FTName], sfi[AttrType]) | sfi \in f\}$;

For each $(ft, type) \in (ft_s, type_s)$ **Do**

{

$f^{(1)} = \sigma_{FTName=ft}(\mathcal{SYSCATALOG}(\text{SYSFRINSTCOUNT}))$;

$fd_s = \{sfi[FolderName] | sfi \in f^{(1)}\}$;

OUTPUT$(att, type, ft, fd_s)$;

}

END

CallValue(v)

(The procedure **CallValue**(v) supports *access by value.* The system returns the other attribute values which occur in the frame instance(s) where the given attribute value v is.)

BEGIN

$f = \sigma_{ActiveDomain \supseteq v}(\mathcal{SYSCATALOG}$(SYSATTRIBUTES));

$(att_s, ft_s) = \{(sfi[AttrName], sfi[FtName]) | sfi \in f\}$;

For each $(att, ft) \in (att_s, ft_s)$ **Do**

{ /* get folders satisfying $att = v$.*/

$f^{(1)} = \sigma_{FTName=ft}(\mathcal{SYSCATALOG}$(SYSFRINSTCOUNT));

$(ft, fd_s) = \{(sfi[FTName], sfi[FolderName]) | sfi \in f^{(1)}\}$;

For each $fd \in fd_s$ **Do**

{ /* get the frame instances satisfying $att = v$.*/

$f^{(2)} = \sigma_{att=v}(fd(ft))$

OUTPUT$(f^{(2)}, fd, ft)$;

}

}

END

CallThesaurus(t)

(The thesaurus can be readily incorporated into the browser to find the objects whose semantics are closely related to the topic(a vague query).)

BEGIN

$f^{(1)} = \sigma_{KeyTerm=t}(\mathcal{SYSCATALOG}(\text{SYSTERMASSOC}))$;

if $f^{(1)} = empty$ **then**

{ /* check SYSSYNONYMS.*/

$f^{(2)} = \sigma_{SynKeyTerms \supseteq t}(\mathcal{SYSCATALOG}(\text{SYSSYNONYMS}))$;

if $f^{(2)} = empty$ **then**

{ /* check SYSNARROWER.*/

$f^{(2)} = \sigma_{NarrKeyTerms \supseteq t}(\mathcal{SYSCATALOG}(\text{SYSNARROWER}))$;

if $f^{(2)} = empty$ **then**

RETURN($unknown$)

}

$k = \{sfi[KeyTerm] | sfi \in f^{(2)}\}$;

$f^{(1)} = \sigma_{KeyTerm=k}(\mathcal{SYSCATALOG}(\text{SYSTERMASSOC}))$

}

$(indextm, type) = \{(sfi[IndexTm], sfi[IndexTmType]) | sfi \in f^{(1)}\}$;

case($type =$ "$Folder$") **CallFolder**($indextm$);

case($type =$ "$FrameTm$") **CallFrameTm**($indextm$);

case($type =$ "$Attribute$") **CallAttribute**($indextm$);

case($type =$ "$value$") **CallValue**($indextm$)

END

6.5 Object Network Constructor

In the previous sections, we pointed out that the browser mechanism allows users to enter multiple topics. The object network for each topic entered by the user is only a subgraph of the object network for the entire system. The connectedness among these subgraphs (i.e., partial object network) depends on the relatedness of their corresponding topics. The object network constructor finds the connections among these topics and forms an object network from multiple object networks before displaying. We shall proceed to give a formal definition of the object network.

6.5.1 Formal Definition for the Object Network

An object network can be denoted by $ON = (N, E, f_N, f_E)$, where

1. $N = N_{fd} \cup N_{ft} \cup N_{at} \cup N_v$, a collection of sets of nodes, where
   (a) N_{fd} is a set of nodes representing the folders in the system;
   (b) N_{ft} is a set of nodes representing the frame templates in the system;
   (c) N_{at} is a set of nodes representing the attributes in the system, and
   (d) N_v is a set of nodes representing the attribute values in the system.
2. $E = E_{(fd,fd)} \cup E_{(fd,ft)} \cup E_{(ft,ft)} \cup E_{(ft,at)} \cup E_{(at,at)} \cup E_{(at,v)}$, a collection of sets of edges, where
   (a) $E_{(fd,fd)} \subseteq N_{fd} \times N_{fd}$. An edge $(fd, fd') \in E_{(fd,fd)}$ denotes the *depends_on*[2] relationship between folders fd and fd' (that is, fd' is a parent of fd);
   (b) $E_{(fd,ft)} \subseteq N_{fd} \times N_{ft}$. An edge $(fd, ft) \in E_{(fd,ft)}$ denotes the *has_type* relationship between a folder fd and a frame template ft (that is, fd contains frame instances over the frame template ft);

[2]the inverse relationship is *is_parent_of*.

(c) $E_{(ft,ft)} \subseteq N_{ft} \times N_{ft}$. An edge $(ft, ft') \in E_{(ft,ft)}$ denotes the $is\_a\_subtype\_of$[3] relationship between frame templates ft and ft' (that is, ft is a subtype of ft');

(d) $E_{(ft,at)} \subseteq N_{ft} \times N_{at}$. An edge $(ft, at) \in E_{(ft,at)}$ denotes the $is\_identified\_by$ relationship between a frame template ft and an attribute at (that is, the at is an attribute of the frame template ft);

(e) $E_{(at,at)} \subseteq N_{at} \times N_{at}$. An edge $(at, at') \in E_{(at,at)}$ denotes the $is\_a\_combination\_of$ relationship between the composited attribute at and its component attribute at', and

(f) $E_{(at,v)} \subseteq N_{at} \times N_v$. An edge $(at, v) \in E_{(at,v)}$ denotes the *includes* relationship between an attribute at and its value v.

3. $f_N = \{f_{fd}, f_{ft}, f_{at}, f_v\}$, a set of mappings, where

(a) $f_{fd} : N_{fd} \rightarrow \{fd\}$, where $\{fd\}$ is the set of folder names in the system;

(b) $f_{ft} : N_{ft} \rightarrow \{ft\}$, where $\{ft\}$ is the set of frame template names in the system;

(c) $f_{at} : N_{at} \rightarrow \{at\}$, where $\{at\}$ is the set of attribute names in the system, and

(d) $f_v : N_v \rightarrow \{v\}$, where $\{v\}$ is the set of attribute values in the system.

4. $f_E = \{f_{(fd,fd)}, f_{(fd,ft)}, f_{(ft,ft)}, f_{(ft,at)}, f_{(at,at)}, f_{(at,v)}\}$, a set of mappings, where

(a) $f_{(fd,fd)} : E_{(fd,fd)} \rightarrow \{is\_parent\_of, depends\_on\}$.

(b) $f_{(fd,ft)} : E_{(fd,ft)} \rightarrow \{has\_type\}$.

(c) $f_{(ft,ft)} : E_{(ft,ft)} \rightarrow \{is\_a\_subtype\_of, is\_a\_supertype\_of\}$.

(d) $f_{(ft,at)} : E_{(ft,at)} \rightarrow \{is\_identified\_by\}$.

[3]the inverse relationship is $is\_a\_supertype\_of$.

(e) $f_{(at,at)} : E_{(at,at)} \rightarrow \{is\_a\_combination\_of\}$.

(f) $f_{(at,v)} : E_{(at,v)} \rightarrow \{includes\}$.

An example of illustrating the construction of an object network for a user's topic is given in Example 6.1.

6.5.2 Connecting Multiple Object Networks

The user can enter more than one topic by connecting them using operator *AND* or *OR*. The *AND* operator is used to connect the topics of the same type, such as, a set of folders, frame templates, attributes or values. The *OR* operator can be used to connect the topics of different types.

When a user enters several topics using connecting operator *OR*, the system may take two kinds of action, *forming object network* and *refining object network*, to complete the object network construction task. By forming an object network, the browser is applied separately on each topic to form its object network (that is, an object network for each topic is formed). If the user asks for further refinement, the system will take an action of *refining object network* to find all the possible connections among these object networks as follows:

- If there are common nodes among these object networks, such as, the nodes corresponding to the same folder, frame template, attribute or value, the object networks are connected by unifying these common nodes.

- If there is *depends_on* relationship between any pair of folders, the object networks are connected by adding *depends_on* edge between these folders.

- If there is *is_a_subtype_of* relationship between any pair of frame templates, the object networks are connected by adding *is_a_subtype_of* edge between these frame templates.

When a user enters several topics using connecting operator AND, the system constructs an object network for each topic first using the browser, and then forms an object network containing only the common objects (objects are related directly or indirectly to these topics) among these object networks before displaying. The obtained object network contains only topics entered by the user if they are nothing in common, or displays all the possible common nodes with respect to the given topics (each topic has its object network) using the connecting operator AND.

For example, upon receiving

TOPICS : *Meeting_Memo* **AND** *Proceedings_Paper*,

possible resultant object network is depicted in Figure 6.3(a), which specifies that a folder **Peter Ng** has types Meeting_Memo and Proceedings_Paper. This resultant object network is different from the object network, as shown in Figure 6.3(b), obtained by entering

TOPICS : *Meeting_Memo* **OR** *Proceedings_Paper*.

Very often the information, which is provided in the obtained object network, is insufficient for fulfilling user's retrieval target. Then the user can continue issuing the topics from the object network or outside the network. If the topics entered by the user using AND operator are from the existing object networks (or at least one of the topics is from the existing object network), the system only extends the existing object networks by adding the common objects among the object networks. Each of the common objects is related to the topics. The relatedness relationships are the

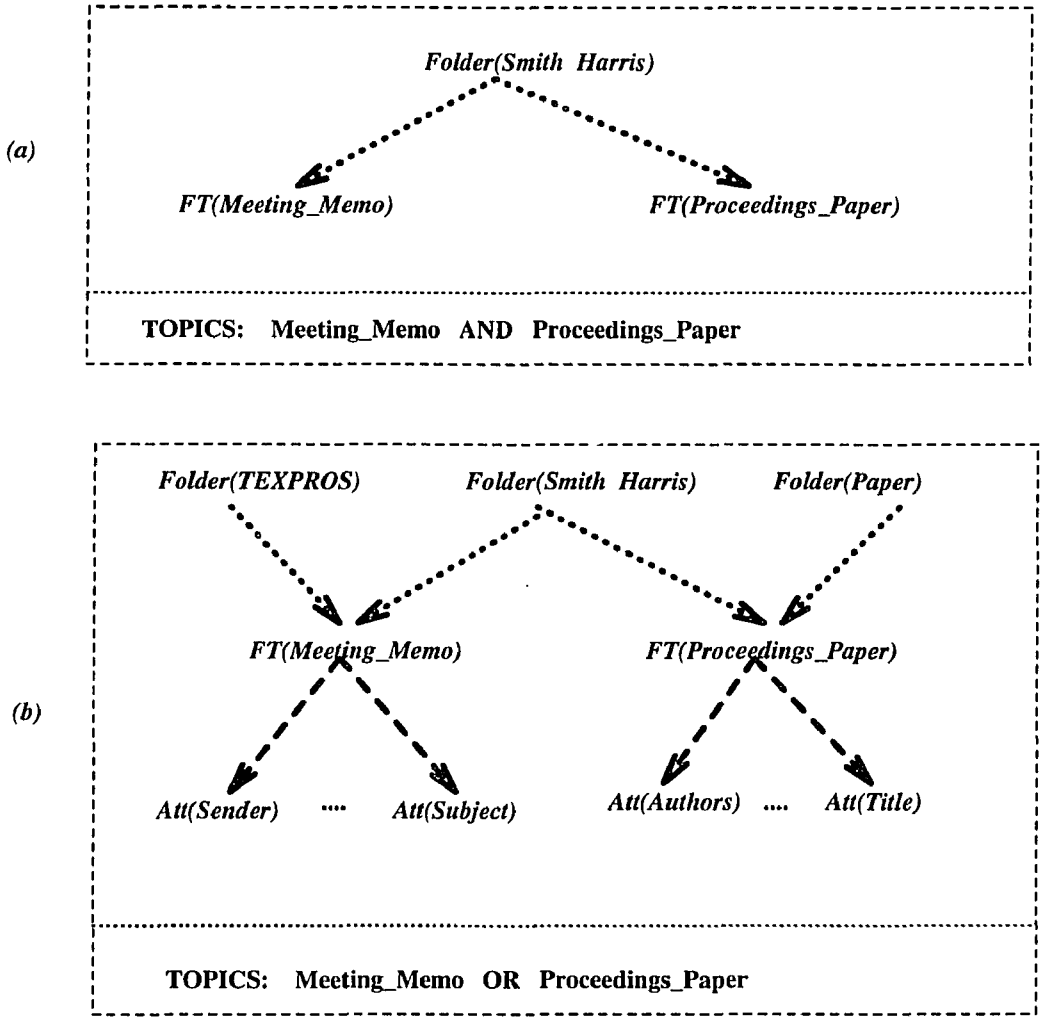

Figure 6.3 Connecting Multiple Object Networks by (a)ANDing Frame Templates and (b)ORing Frame Templates

relationships among objects of the object network model. Therefore, the following browsing targets can be achieved further:

- If the entered topics are the folders, the question, "What are the other frame templates associated to all these folders?" can be answered.
- If the entered topics are the frame templates, the following questions can be answered from the resultant object network:
  - What are the other folders having all these frame templates?
  - What are the attributes included in all these frame templates?
- If the entered topics are the attributes, the question, "What are the other frame templates including all these attributes?" can be answered.
- If the entered topics are the values, the question, "What are the other attributes including all these values?" can be answered.

6.6 Examples

Example 6.1: Using the query interface as shown in Figure 3.4, when the user enters a topic, such as "Peter Ng", the system gathers and responds with all the information related to the topic in the following manner.

- The topic interpreter can interpret this topic as follows:
  - The system searches through the system frame instances of the type SYSTERMASSOC in the system catalog and learns from one of the frame instance that *Peter Ng* is a folder name in TEXPROS.[4]
  - The system searches through the system frame instances of the type SYSFOLDERS in the system catalog and learns that:

[4]Note that the index term *Peter Ng* can be of different index term types.

"the folder **Peter Ng** depends on **Faculty**", and the folder **Peter Ng** contains many frame instances of the types "Letter_of_Appointment_Offer", "Meeting_Memo", "Resume", "Performance_Evaluation_Report" "Faculty_Annual_Summary", "Proceedings_Paper", and others.

- The system searches through the system frame instances of the type SYSATTRIBUTES and SYSFRINSTCOUNT in the system catalog and learns that *Peter Ng* is an attribute value. Therefore, the system retrieves other values related to *Peter Ng* from the database, such that the following information reflecting *is-A_i-of*-FT-*in*-f-*having-A_j* relationships may be displayed to the user:
  * *Peter Ng* is the Sender of a Meeting_Memo having the Subject *Ph.D. Qualifying Examination* in the folder **Peter Ng**.
  * *Peter Ng* is one of the Authors of a Proceedings_Paper having the Title *A Query Algebra for Office Documents System* in the folder **Peter Ng**.

- Figure 6.4 depicts a portion of the object network pertaining to the vague query "What is Peter Ng", resulting from the process of object network constructor. The formal specification of the object network is given as follows:

  1. $N = N_{fd} \cup N_{ft} \cup N_{at} \cup N_v$, where
     $N_{fd} = \{fd, fd_p\}$;
     $N_{ft} = \{ft_{mm}, ft_r, ft_{pp}, ft_{jp}, \ldots\}$;
     $N_{at} = \{at_{se}, at_{su}, \ldots, at_{au}, at_t, \ldots\}$, and
     $N_v = \{v_{pn}, v_{pqe}, \ldots, v_{aqa}, \ldots\}$.
  2. $E = E_{(fd,fd)} \cup E_{(fd,ft)} \cup E_{(ft,at)} \cup E_{(at,v)}$, where
     $E_{(fd,fd)} = \{(fd, fd_p)\}$;
     $E_{(fd,ft)} = \{(fd, ft_{mm}), (fd, ft_r), (fd, ft_{pp}), (fd, ft_{jp}), \ldots\}$;

$$E_{(ft,at)} = \{(ft_{mm}, at_{se}), (ft_{mm}, at_{su}), \ldots, (ft_{pp}, at_{au}), (ft_{pp}, at_t), \ldots,$$
$$(ft_{jp}, at_{au}), (ft_{jp}, at_t), \ldots\}, \text{ and}$$
$$E_{(at,v)} = \{(at_{se}, v_{pn}), (at_{su}, v_{pqe}), \ldots, (at_t, v_{aqa}), \ldots\}.$$

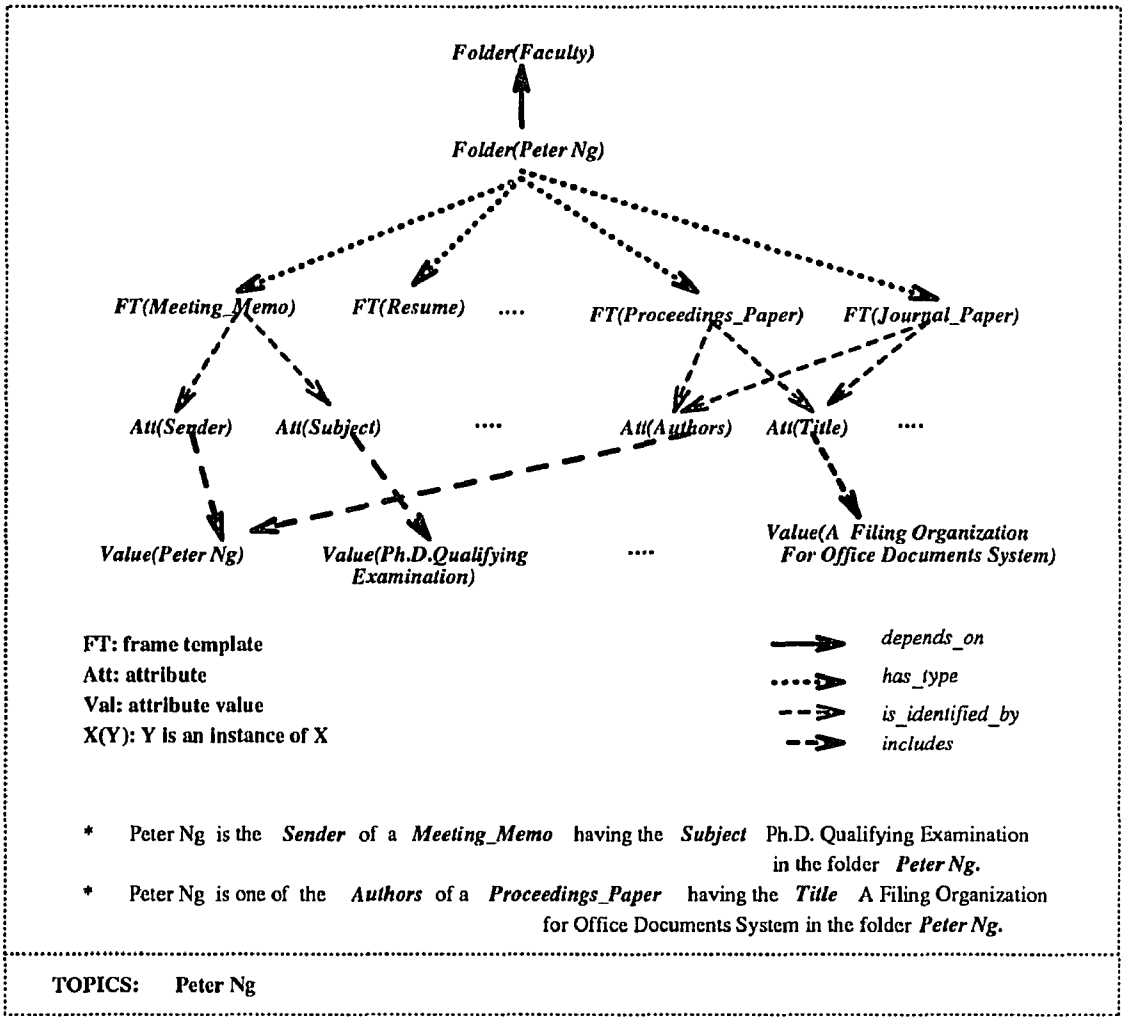

Figure 6.4 Constructing an Object Network

3. $f_N = \{f_{fd}, f_{ft}, f_{at}, f_v\}$, where

   $f_{fd}(fd) = PeterNg$;

   $f_{fd}(fd_p) = Faculty$;

   $f_{ft}(ft_{mm}) = Meeting\_Memo$; $f_{ft}(ft_r) = Resume$;

   $f_{ft}(ft_{pp}) = Proceedings\_Paper$;

   $f_{ft}(ft_{jp}) = Journal\_Paper$;

   $\ldots$;

   $f_{at}(at_{se}) = Sender$; $f_{at}(at_{su}) = Subject; \ldots$; $f_{at}(at_{au}) = Authors$;

   $f_{at}(at_t) = Title$;

   $\ldots$;

   $f_v(v_{pn}) = Peter\ Ng$;

   $f_v(v_{pqe}) = Ph.D.Qualifying\ Examination; \ldots$;

   $f_v(v_{aqa}) = A\ Query\ Algebra\ for\ Office\ Document\ System; \ldots$

4. $f_E = \{f_{(fd,fd)}, f_{(fd,ft)}\}$, where

   $f_{(fd,fd)}((fd, fd_p)) = depends\_on$;

   $f_{(fd,ft)}((fd, ft_{mm})) = has\_type$; $f_{(fd,ft)}((fd, ft_r)) = has\_type$;

   $f_{(fd,ft)}((fd, ft_{pp})) = has\_type$; $f_{(fd,ft)}((fd, ft_{jp})) = has\_type$;

   $\ldots$;

   $f_{(ft,at)}((ft_{mm}, at_{se})) = is\_identified\_by$; $f_{(ft,at)}((ft_{mm}, at_{su})) = is\_identified\_by \ldots$;

   $f_{(ft,at)}((ft_{pp}, at_{au})) = is\_identified\_by$; $f_{(ft,at)}((ft_{pp}, at_t)) = is\_identified\_by \ldots$;

   $f_{(ft,at)}((ft_{jp}, at_{au})) = is\_identified\_by$; $f_{(ft,at)}((ft_{jp}, at_t)) = is\_identified\_by$;

   $\ldots$;

   $f_{(at,v)}((at_{se}, v_{pn})) = includes$;

   $f_{(at,v)}((at_{su}, v_{pqe})) = includes\ \ldots$;

   $f_{(at,v)}((at_t, v_{aqa})) = includes; \ldots$

Example 6.2: (Connecting Multiple Object Networks by Unifying their Common Nodes)

Upon receiving the vague query,

TOPICS : *Q.E.Application_Form* **OR** *Journal_Paper*,

the system first generates two object networks, which are related to the frame templates *Q.E.Application_Form* and *Journal_Paper*, respectively. After refining these two object networks, an object network, as shown in Figure 6.5, is constructed by unifying the common node, namely, the folder **Fortune**.

Example6.3 :(Connecting Multiple Object Networks by Adding *depends_on* Edge)

Upon receiving the vague query,

TOPICS : *Jennifer* **OR** *Paper*,

the system first generates two object networks with respect to the folders *Jennifer* and *Paper*. After refining these two object networks, an object network, as shown in Figure 6.6, is constructed by adding the *depends_on* edge between the folders **Ph.D.Students** and **Publication**.

Example6.4 :(Connecting Multiple Object Networks by ANDing Frame Templates)

From the object network in Figure 6.4, a user may issue a vague query,

TOPICS : *Proceedings_Paper* **AND** *Journal_Paper*,

when he wants to know "What are the other folders having the frame templates *Proceedings_Paper* and *Journal_Paper*. The folder **Paper** having the types

Proceedings_Paper and *Journal_Paper* is added to the object network in Figure 6.4 to yield the resultant object network as shown in Figure 6.6.

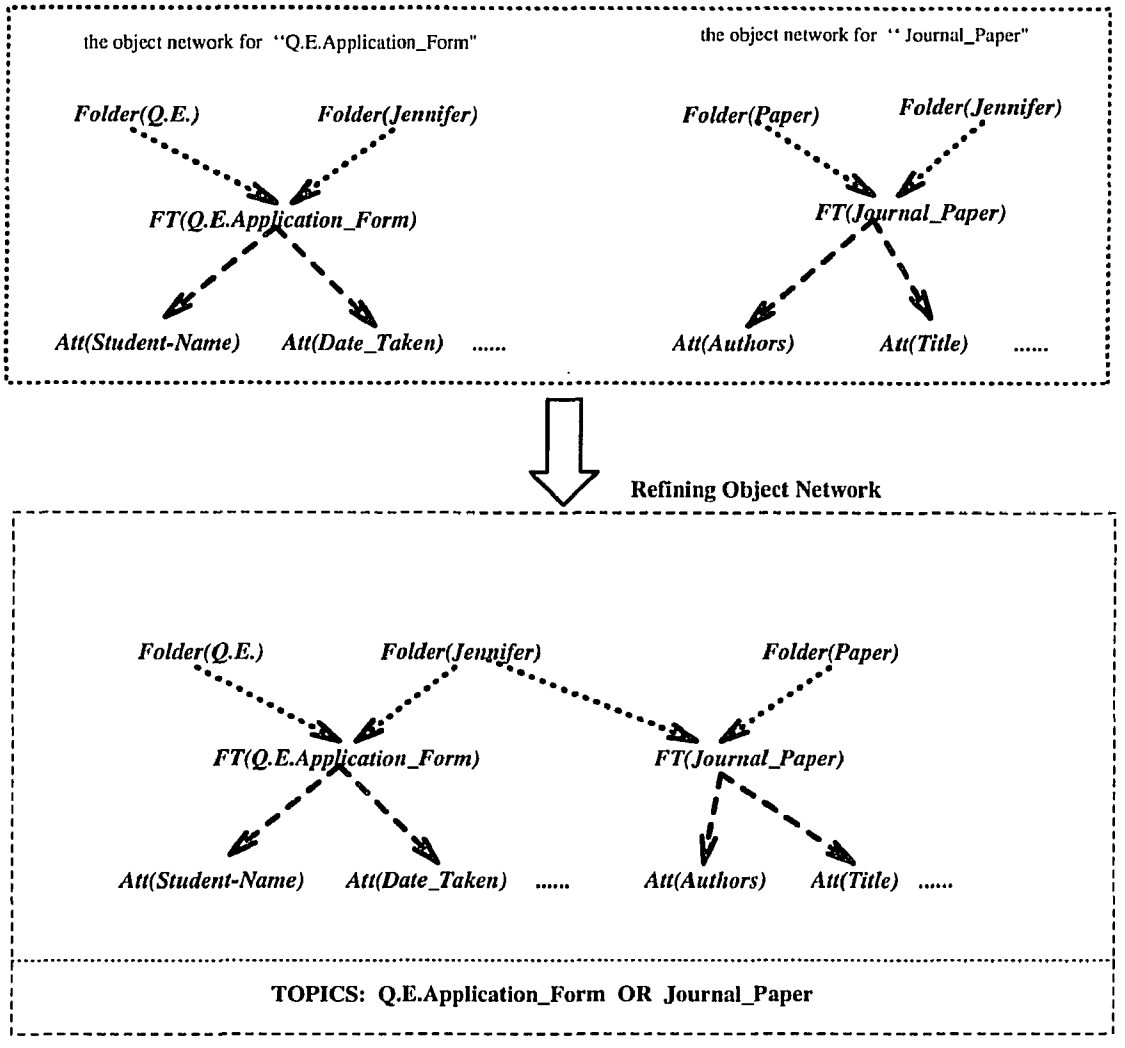

Figure 6.5 Connecting Multiple Object Networks by Unifying their Common Nodes

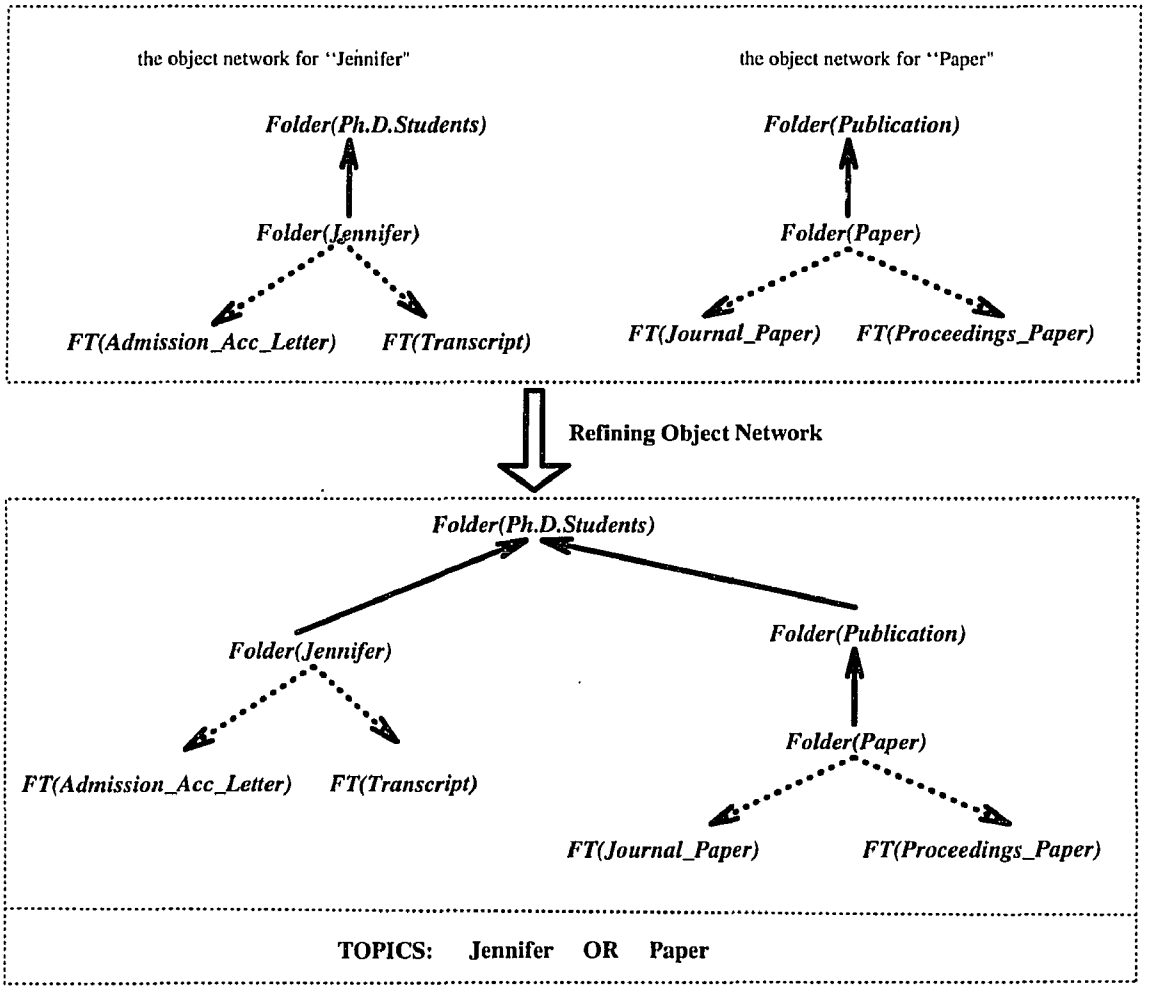

Figure 6.6 Connecting Multiple Object Networks by Adding *depends_on* Edge

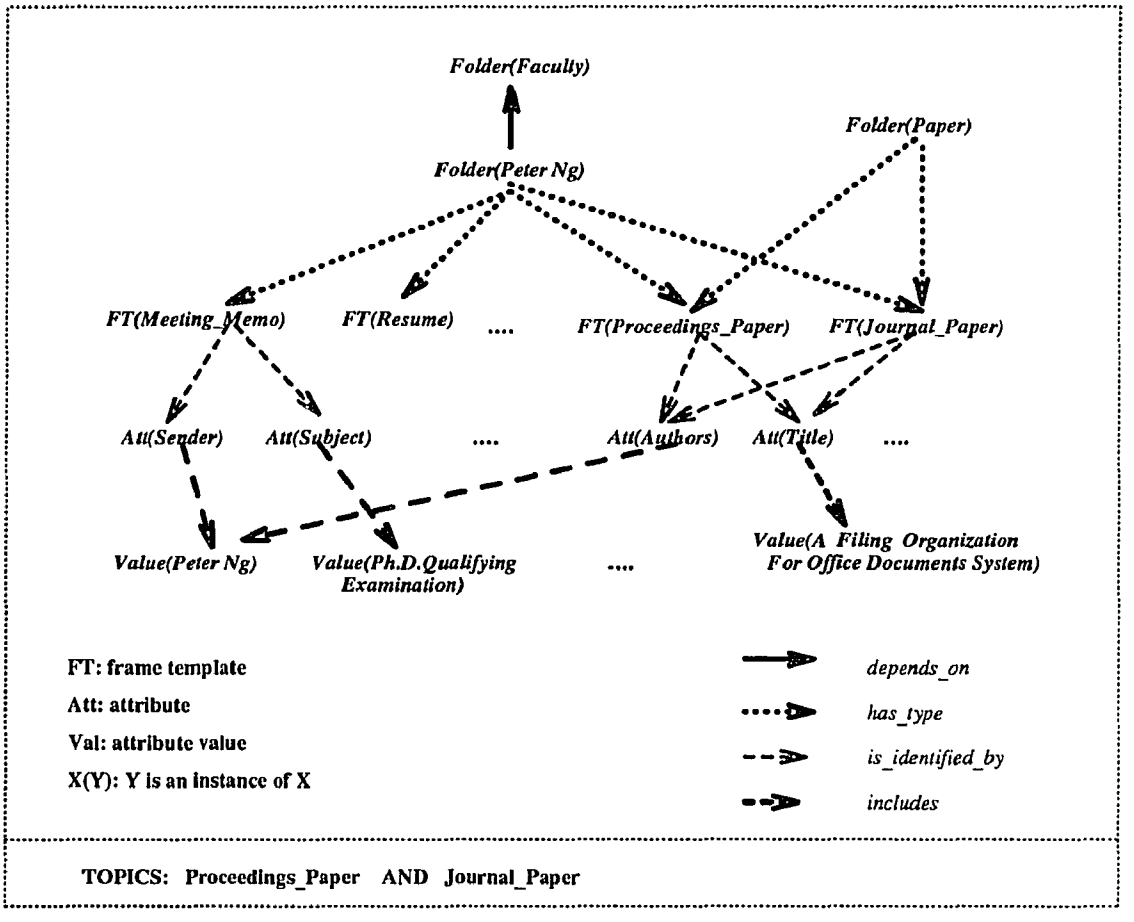

Figure 6.7 Connecting Multiple Object Networks by ANDing Frame Templates

CHAPTER 7

GENERALIZER

The context construction mechanism is introduced into our system primarily to relieve users from the necessity of remembering the precise terms (such as, index terms and keyterms) of individual entities in the system. However, since the query entered by the user is less restrictive, the response given to the user by the system may be less cooperative. According to Kao et al.[45], the requirements for achieving cooperative responses from the system are as follows: (1) the maxim of quantity: be as informative as required; (2) the maxim of quality: contribute only when an adequate amount of evidence is present; (3) the maxim of relation: be relevant; and (4) the maxim of manner: avoid ambiguity.

Several systems which are capable of generating cooperative responses have been developed. Schank and Lehnert[81] extended the response to the user's vague and ambiguity query. McCoy's ENHANCE system [57] and the McKeown's TEXT system [58] attempted to generate answers for requesting the meta-knowledge. They employed the knowledge base that includes the concept used in the database, to accomplish the generalization hierarchy from the data itself. Kaplan [46] presented a portable natural language query system with capability of generating cooperative response to natural language query. Especially in the case of null answer query, the kinds of cooperative response that the system can offer include: corrective indirect response, suggestive indirect response, and supportive indirect response. To accomplish these, it employs the domain transparent mechanism and Meta-Query Language. Kalita [44] described how to give the summary response for short non-enumerative answers. The system employs a knowledge base which consists of frames that are used to store the information about database schema. Motro [69] presented another approach to interpreting null answers. According to his idea, every query reflects a presupposition that the retrieval request being expressed is plausible and

the source of a null answer is in erroneous presupposition. A verification mechanism is employed to detect these erroneous presuppositions [66]. A generalizer is employed to generate a set of output presuppositions which are minimally more general than the given input presupposition. This can be done by weakening mathematical conditions placed upon the queries or by deleting conjunction from the queries. ARES [41] is a system with the capability of performing flexible interpretation of the queries that is based on the relational data model and allowing for a certain amount of ambiguity as well. This can be achieved by functionally augmenting the relational operations with the additional comparison operator "approximately equal to".

7.1 The Design of Our System: An Enhanced Generalizer

All of the systems mentioned above require extending the original data model to one with general information about the meta-data and domain knowledge of some sorts. TEXPROS requires these kinds of information which are stored in the System Catalog.

The following example demonstrates that the null answer is rarely satisfactory in our system. Consider a query which retrieves all the students who were enrolled in the course CIS792 (Pre-doctoral Research) and received a grade A from "M.S. Students" folder. As there is no enrollment for which the course is CIS792 and the student received the grade "A" in "M.S. Students" folder, the system returns a null answer. The null answer can be interpreted as follows:

- There is no information (i.e, no M.S. student takes CIS792) in the "M.S. Students" folder.
- There is no M.S. student who received a grade A in the course CIS792.
- The information is located in other folders.

- The information is stored in the system as frame instances of other types rather than those of the type which are used for examining the query.

Actually the query reflects a presupposition of the user that some of M.S. students were enrolled in CIS792. In fact, only Ph.D. students were enrolled in this course; so the original query reflects an erroneous presupposition and the null answer is a fake empty answer.

In this dissertation, we present a generalizer mechanism for answering the queries that reflects erroneous presuppositions with informative messages instead of a null answer.

7.2 Principles of Generalizer

Motro [69] proposed a query generalizer, which issues a set of more general queries from the original query to determine whether the empty answer is *genuine*, or whether the original query reflects *erroneous presuppositions* on behalf of the user. Consequently, the procedure can be described as follows: when a query fails (with an empty answer), its immediate generalizations are generated and attempted. If all the immediate generalizations succeed (with nonempty answers), the original empty answer was *genuine*, and the answers of the generalizations may be considered as the partial answer of the original query. If at least one of immediate generalizations fails, the original empty answer was *fake*. This procedure is continued until all *significant failures* of queries are detected. (A failure of a query is considered to be significant only if all of its generalizations succeed.)

Motro used the SQL query language to demonstrate his approach. To generalize a query with conjunctive normal form in the WHERE clause, in which every primitive term is a comparison between two attributes or between an attribute and a value,

a set of queries was produced by weakening a single primitive term at a time. For example, "$GPA > 3.6$" was replaced by "$GPA > 3.4$".

7.3 Motivation

We are employing the logical file organization and document type hierarchy in our model; consequently, the user needs to specify the *folder*, the *frame template* or the *attribute* in the query to retrieve the information. As mentioned before, the context construction mechanism relieves users of the necessity to remember the precise names (such as *folder name* or *frame template name*) of individual entities in the system. However, since the query entered by the user is less restrictive, the response to the query given to the user by the system may be less cooperative. In TEXPROS, generating precise and meaningful responses is our target in the situation when empty answers arise, and therefore the generalizer is developed by incorporating both the folder substitution and the type substitution.

7.4 Folder Substitution

To generalize a failed query, the folder name in the query is substituted by the name of those folders whose semantics are similar to the original folder and are relevant to the original query. To accomplish the folder substitution, the similarity between two folders in the logical file organization is taken into consideration.

7.4.1 Similarity Definition

Similarity (as defined in [41]) is used in the flexible interpretation such that the values of attributes which are semantically close to an exact match with the query condition can be obtained. We extend this concept to the similarity between folders in the logical file organization based on their semantics (such as the content of the folders) in our system. For instance, in the filing organization as shown in Figure

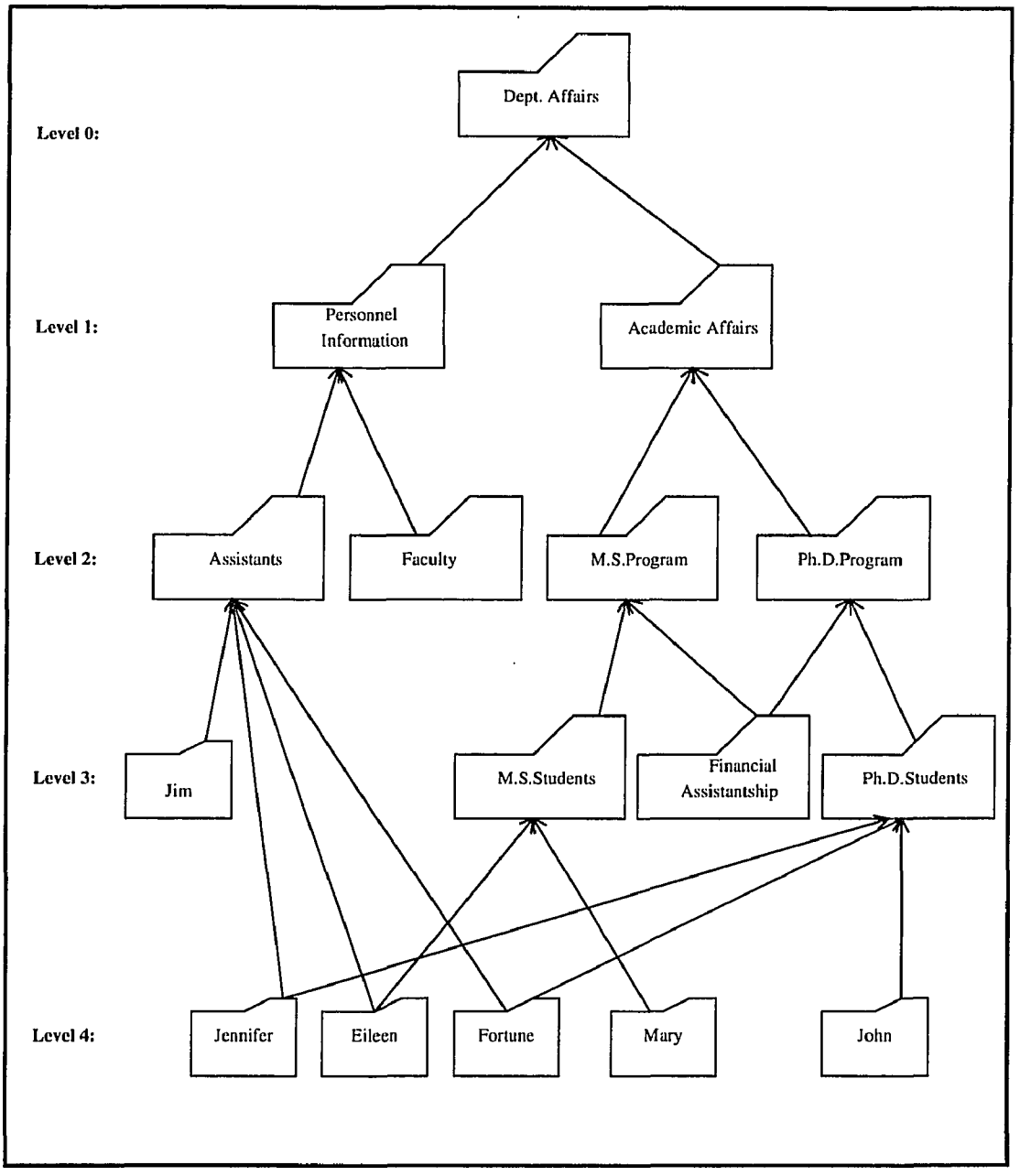

Figure 7.1 Part of Filing Organization

7.1, the content of the folder "John" is more similar to the content of the folder "Fortune" than to the content of the folder "Jim", since both John and Fortune are Ph.D. students and Jim is not.

Given a logical file organization (which is possibly a DAG structure), the folder fd_0 which is not a subfolder of any folders is considered to be at level 0. Assume that there is a folder fd_j containing no subfolder. The folder fd_j is at the level n if there exists a path of maximal distance n from fd_0 at level 0 to fd_j.

For example, in Figure 7.1, the "Dept. Affairs" folder (the superfolder for this case) is at level 0. The folder "Jim" is at level 3, and the folders "Jennifer", "Eileen", "Fortune", "Mary" and "John" are at level 4. The folders "Assistants", "Faculty", "M.S. Program" and "Ph.D.Program" are at level 2. The folders "M.S. Students", "Financial Assistantship" and "Ph.D. Students" are at level 3.

We derive the similarity between the folders from the bottom level (level n) of the hierarchy of the logical file organization. The similarity between two folders is set to the level of the folder which is the least common parent of both. For instance, in Figure 7.1, the similarity between the folders "Fortune" and "John" is set to the level of the folder "Ph.D. Students", which is 3. For the folders, which have more than one common parent, the similarity between the folders is calculated using the following formula:

$$L_c + \frac{P_c - 1}{N}$$

where L_c is the level of the least common parent; P_c denotes the number of common parents, and N denotes the total number of folders in the filing organization. For instance, these are two common parents, namely, the "Ph.D. Students" and "Assistants", for the folders " Fortune" and "Jennifer"; so the similarity between them would be $3 + \frac{1}{16} = 3.06$.[1]

[1]Assume that there are totally sixteen folders in the filing organization.

7.4.2 Similarity in SYSTEM CATALOG

The similarities between folders may be stored in the system catalog as the system frame instances whose type is SYSSIMILARITY as shown in Figure 7.2. The system updates those frame instances dynamically during document filing. However, updating the similarities in the system catalog according to every change in the filing system is usually expensive and may not be realistic. Furthermore, it is not necessary that the similarities among all the folders are maintained. In other words, some of similarities have been used rarely.

One solution to this is a lazy computation approach, which computes the similarities between folders when they need. Hence, when the generalizations of a query are generated, a similarity generator will be called to return the most updated similarities between folders involved in the query. [2] Requesting the similarities when query is generalized ensures that the most updated similarities are being used.

7.4.3 Semantic and Structural Interdependency

We need to distinguish the folders which have the same similarities with a specific folder *fd*. For example, the similarity of "Ph.D. Students" and "Assistants" is the same as the similarity of "Ph.D. Students" and "Faculty". (Both are of the level of the folder "Dept.Affairs".) Consider the semantic and structural interdependencies among folders for solving the problem. Four types of interdependencies among folders are defined: *jointness*, *disjointness*, *partial_jointness* and *covering*. *Jointness* holds between two folders having common frame templates. *Disjointness* holds between two folders having no common frame templates. *Covering* holds when a folder is a superset of the union of other folders. *Partial_jointness* holds among a set of folders if there exists a folder which is a subset of the union of this set of folders. The *covering* and *partial_jointness* are considered as semantic interdependencies because

[2] It seems reasonable to keep the information about the levels of folders in the system catalog.

The corresponding frame instances for SYSSIMILARITY

IndexTerm1	IndexTerm2	IndexTmType	Similarity
Fortune	John	Folder	3
Fortune	Jennifer	Folder	3.06
Fortune	Eileen	Folder	2
Fortune	Mary	Folder	1
John	Mary	Folder	1
John	Jim	Folder	0
.....			
Ph.D. Students	Assistants	Folder	0
Ph.D. Students	Faculty	Folder	0
Ph.D. Students	John	Folder	3
M.S. Program	Ph.D. Program	Folder	1
M.S. Program	Academic Affairs	Folder	1
M.S. Students	Ph.D. Students	Folder	1
M.S. Students	Ph.D. Program	Folder	1
M.S. Students	Financial Assistantship	Folder	2
.....			

Figure 7.2 Similarity in SYSTEM CATALOG

they deal with the content of the folders, whereas, the *jointness* and *disjointness* are of structural interdependencies since they deal with the type of content in the folders.

For instance, consider the folders "Ph.D. Students", "Assistants", and their subfolders, such as "Fortune", "Jennifer" and "John", etc . Since the union of "Ph.D. Students" and "Assistants" is the superset of the union of these subfolders, the relationship of "Ph.D. Students" and "Assistants" with these subfolders is a *covering.* However, the "Ph.D. Students" does not cover all its subfolders. The *partial_jointness* holds between the folder "Ph.D. Students" and the "Assistants", since a folder "Jennifer" is the subset of the union of "Ph.D. Students" and "Assistants". The *jointness* holds between the folder "Fortune" and "Jennifer", if they contain some common frame templates, such as the "Full_Transcript" of Ph.D. students. The *disjointness* holds between the folder "John" and "Jim", if they contain the frame instances of different types.

We proceed to formally define the semantic and structural interdependencies as follows:

Definition 7.1: Let C_j be the criteria for a folder f_j. Then $\mathsf{C}_j(fi)$ must be true for any frame instance fi to be located in the folder f_j. Let $\mathsf{ft}(fi)$ denotes a frame instance fi over the frame template ft.

- Covering:

  Let f_j be a folder with criteria C_j.

  Let $f_{j1}, f_{j2}, \ldots, f_{jn}$ $(n > 0)$ be a set of folders with criteria $\mathsf{C}_{j1}, \mathsf{C}_{j2}, \ldots, \mathsf{C}_{jn}$, respectively. Then the relationship between f_j and f_{jk} $(1 \leq k \leq n)$ is a covering, or f_j covers f_{jk}, if for every frame instance fi from f_{jk} $(1 \leq k \leq n)$,

  $((\mathsf{C}_j(fi) \wedge \mathsf{C}_{j1}(fi)) \vee (\mathsf{C}_j(fi) \wedge \mathsf{C}_{j2}(fi)) \vee \ldots \vee (\mathsf{C}_j(fi) \wedge \mathsf{C}_{jn}(fi)))$ is true.

It should be noted that the folder f_j could possibly contain some frame instances which does not satisfy any f_{jk} $(1 \leq k \leq n)$.

- Jointness:

  Let f_1 and f_2 be two folders with criteria C_1 and C_2 respectively.

  Then f_1 and f_2 are joint (or satisfy the jointness condition)

  if $(\exists ft)(\exists fi_1)(\exists fi_2)((ft(fi_1) \wedge ft(fi_2)) \wedge (C_1(fi_1) \wedge C_2(fi_2)))$ is true.

- Partial_jointness:

  Let f_1 and f_2 be two folders with criteria C_1 and C_2 respectively. Then f_1 and f_2 are partially joint with respect to f_j (or satisfy the partial_jointness condition) if

  1) $\exists$ a folder f_j with criteria C_j such that for every frame instance fi in f_j, $((C_1(fi) \wedge C_j(fi)) \vee (C_2(fi) \wedge C_j(fi)))$ is true[3], and

  2) for each $1 \leq k \leq 2$, there is at least one fi in f_j such that $((C_k(fi) \wedge C_j(fi))$ is true.

  Note that the first condition of the *partial_jointness* is to consider all the frame instances in the folder f_j, and the second condition is to ensure that each of the folder f_1 and f_2 must have at least one frame instance from f_j satisfying its criteria.

- Disjointness:

  Let f_1 and f_2 be two folders with criteria C_1 and C_2 respectively.

  Then f_1 and f_2 are disjoint

  if $(C_1(fi_1) \wedge C_2(fi_2))$ is false for every frame instance over the same frame template, fi_1 from f_1 and fi_2 from f_2.

[3]It means that f_j depends on $f_1 \cup f_2$.

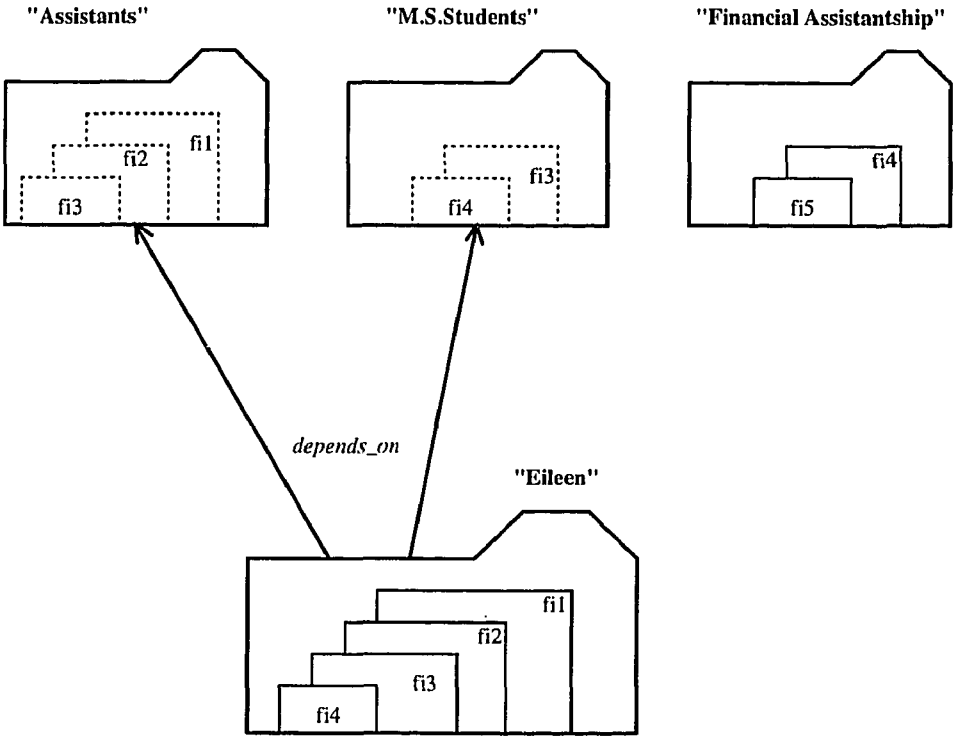

Figure 7.3 Contents of the Folders

Before concluding this subsection, let us consider the folders "Eileen", "Assistants", "M.S. Students" and "Financial Assistantship" as shown in Figure 7.3. Assume that the folder "Eileen" contains four frame instances: there are two frame instances (say fi_1 and fi_2) of personnel information of being a student assistant; a frame instance (say fi_3) states that she requires to enroll as a full-time M.S. student to be able to work as an assistant; and a frame instance (say fi_4) offers her tuition fee waiver for completing the M.S. degree. In the folder "Financial Assistantship", it also contains a frame instance (say fi_5) of information which is irrelevant to Eileen. In Figure 7.3, the folder "Eileen" contains the frame instances fi_1, fi_2, fi_3 and fi_4, and the folder "Financial Assistantship" may contain the frame instances fi_4 and fi_5. The abstract folders "Assistants" and "M.S. Students" virtually contains those frame instances

depicted in the dotted lines which satisfy their criteria but are actually deposited in the concrete folder "Eileen".

We say that the folders "Assistants" and "M.S. Students" are partially joint with respect to the folder "Eileen" since for all the frame instances in "Eileen", some satisfy the criteria for the "Assistants" and some satisfy the criteria for the "M.S. Students". The folders "Assistants" and "Financial Assistantship" are also partially joint with respect to the folder "Eileen". And the folders "Assistants", "M.S. Students" and "Financial Assistantship" satisfy also the partial jointness condition with respect to "Eileen". But the folders "M.S. Students" and "Financial Assistantship" do not satisfy the partial jointness condition with respect to "Eileen" because some of the frame instances in the folder "Eileen", such as fi_1, fi_2, do not meet the criteria of "M.S. Students" or "Financial Assistantship". (That is, the combined folder of "M.S. Students" and "Financial Assistantship" does not cover the "Eileen".)

The folders "Assistants" and "Eileen" do not satisfy the covering condition. In general sense, we can say that the combined folder of "Assistants" and "M.S. Students" covers the folder "Eileen".

For structural interdependency, in addition to the folders "Assistants" and "M.S. Students", the folders "M.S. Students" and "Financial Assistantship" satisfy the jointness condition, since they contain a common frame instance (fi_4) concerning her tuition fee waived as a M.S. Student. However, the folders "Assistants" and "Financial Assistantship" do not satisfy the jointness conditions; and therefore they satisfy the disjointness condition. In fact, the folder "Eileen" is joint with the folder "Assistants", "M.S. Students", or "Financial Assistantship".

7.4.4 Rules of Folder Substitution

The folder substitution is established by the following rules:

- The system searches the system catalog and returns to the user a sequence of folders, one by one, in the order that the first returned folder has highest similarity to the folder in the original query (in the order with the similarity of the highest first, etc).

- To reduce the number of irrelevant substitutions, the user can discontinue any substitution if the returned folder is considered to be irrelevant to the query.[4] For instance, in the example of section 7.6, the user rejects the substitution of the folder "Financial Assistantship" for the "M.S. Students" folder since it is irrelevant to the original query about the grade of the students.

- The system displays the folders which are similar to the original folder in the sequence according to the appropriate priorities, which are based on the semantic and structural interdependencies defined in the previous section. For example, given the original folder, say the "Ph.D. Students", the "Assistants" precedes the "Faculty" in the sequence of folders returned by the system for folder substitution, since the relationship of "Assistants" and "Ph.D. Students" is a partial_jointness but the relationship of "Faculty" and "Ph.D. Students" is a disjointness.

- If two folders have the same relationship with the original folder, then the folder with the higher level number (of the logical file organization) is prior to the other folder with the lower level number.

[4]We should point out that comparing the similarities between folders is more meaningful when their context is taken into consideration.

For instance, given the original folder "M.S. Students", the "Ph.D. Students" is preceded to the "Ph.D. Program" in the sequence of folders for substitution, since the "Ph.D. Students" is with the higher level number than the level number of the "Ph.D. Program" folder. However, both relationships of "M.S. Students" and "Ph.D. Students", and of "M.S. Students" and "Ph.D. Program" are of disjointness.

- If two folders at the same level have the same relationship with an original folder, then the system assigns them in an arbitrary order to appear in the sequence of folders for substitution.

7.5 Type Substitution

A failed query can be generalized by substituting other frame templates, which may possibly be the types of frame instances retrieved by the user, for the frame template appearing in the failed query. This process is called *type substitution.* The general rules for type substitution are as follows:

1. Select the frame templates which are the siblings[5] of the original one to be its substitutes first. For instance, in Figure 7.4, when the frame template "Grade_Report" is specified in a failed query, it is replaced by "Full_Transcript" or "Course_Grade_Report" which are the siblings of "Grade_Report" in the document type hierarchy.

2. When all the substitutions in (1) fail, we substitute the frame templates of its immediate parent for the original frame template . For instance, replacing "Grade_Report" in the failed query by "Transcript".

3. If (2) still fails, treat the parent as the original frame template and return to (1).

[5] ft_i and ft_j are the siblings if they have the same immediate parent.

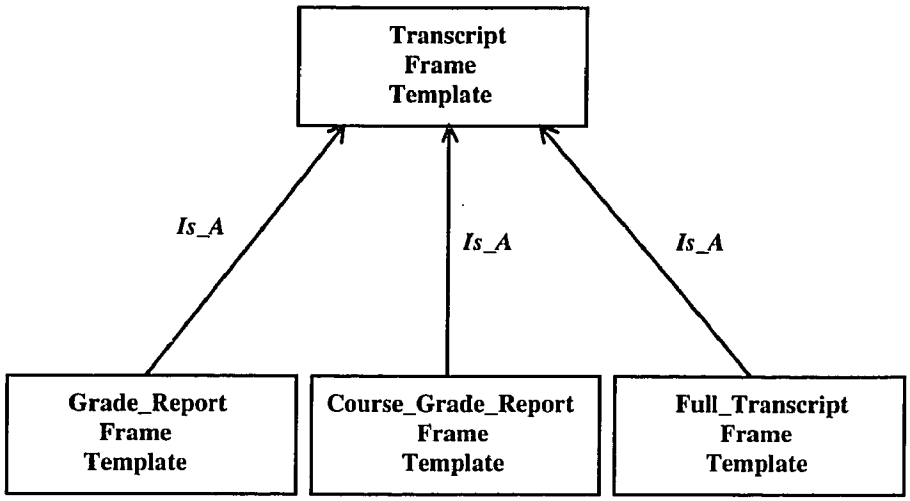

Figure 7.4 A Document Type Hierarchy

7.6 Example

The following example demonstrates our approach. As the evaluation of a given query shown in Figure 7.5 preduces an empty answer, the system makes an attempt to determine the reason of producing the empty answer by generalizing the original query, which is specified in a hierarchy of query generalizations shown in Figure 7.6. Then the generalizations of the query are further accomplished by executing the folder substitution.

The generalizations of the query are derived continuously by weakening the search criteria. The search criterion of the original query include the *M.S.Students* folder (**F**), *Course_Grade_Report* frame template (**T**), *Course_No* = "*CIS*792" (**C**), and *Grade* = "*A*" (**A**). The original query is generalized to the following queries by reducing the conditions *Course_No* = "*CIS*792", *Grade* = "*A*", *Course_Grade_Report* as the frame template type for the frame instances, or *M.S.Students* as the folder where the frame instances to be looked for.

QUERY: Retrieve all the students who were enrolled in the course CIS792 and received the grade A from the "M.S.Students" folder.

```
SELECT   M.S.Students(Course_Grade_Report).Student_Name
FROM     M.S.Students(Course_Grade_Report)
WHERE    M.S.Students(Course_Grade_Report).Course_No = "CIS792"
                          AND
         M.S.Sudents(Course_Grade_Report). Grade = "A"
```

Figure 7.5 The Query with Empty Answer

- **Q1**: "retrieve all the students who received a grade A in a Course_Grade_Report from the "M.S. Students" folder. " (**FTA**)
- **Q2**: "retrieve all the students who received a grade A for the course CIS792 in the Course_Grade_Report." (**TCA**)
- **Q3**: "retrieve all the students who were enrolled in the course CIS792 and their Course_Grade_Report from the "M.S. Students" folder." (**FTC**)
- **Q4**: "retrieve all the students who received a grade A for the course CIS792 in the "M.S. Students" folder." (**FCA**)

The system returns nonempty answers for the queries **FTA** and **TCA**, and no further generalization for these two succeeded queries is needed. However, the system still returns an empty answer for the query **FTC** and **FCA**. Therefore, the generalizations for these two queries are further proceeded as follows by reducing the search criteria:

- **Q31**: "retrieve all the students from their Course_Grade_Report in the "M.S. Students" folder." (**FT**)
- **Q32**: "retrieve all the students who were enrolled in the course CIS792 from the Course_Grade_Report." (**TC**)

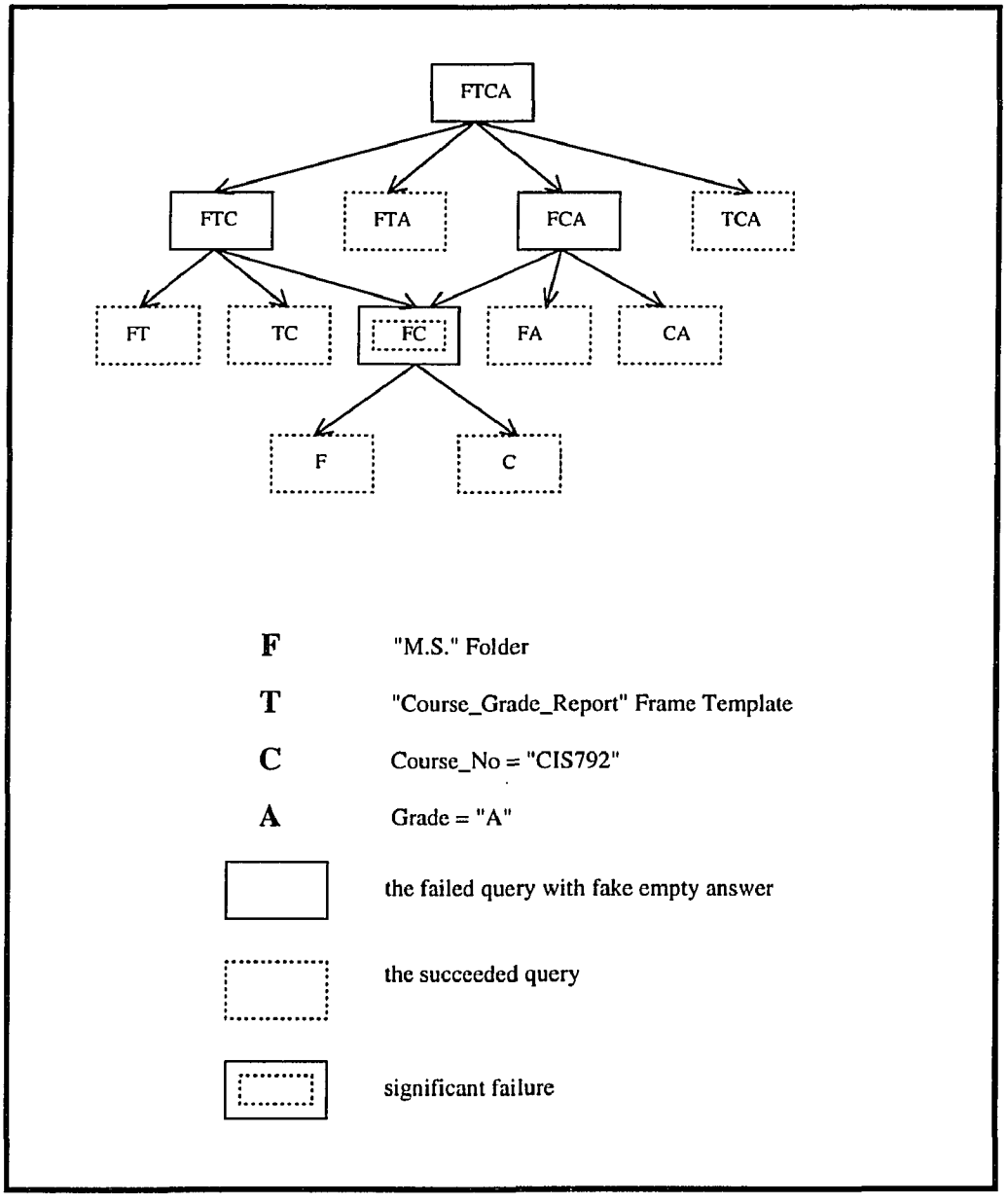

Figure 7.6 A Hierarchy of Generalizations

- **Q33**: "retrieve all the students who were enrolled in the course CIS792 from the "M.S. Students" folder." (**FC**)
- **Q41**: same as **Q33.**
- **Q42**: "retrieve all the students who received the grade A from the "M.S. Students" folder." (**FA**)
- **Q43**: "retrieve all the students who were enrolled in the course CIS792 and got grade A." (**CA**)

The system still returns an empty answer for the query **FC**, while the other generalized queries, **FT**, **TC**, **FA** and **CA** succeed with non-empty answers. The failed query **FC** is generalized further to form the following two queries:

- **Q33$_1$**: "retrieve all the students from the "M.S. Students" folder." (**F**)
- **Q33$_2$** " retrieve all the students who were enrolled in the course CIS792." (**C**)

Since both queries **F** and **C** succeed with non-empty answers, it is an indication that the empty answer for the query **FC** was genuine. The significant failure of query **FC** is detected. The system is saying "None of the M.S. students was enrolled in the course CIS792!".

To find the folders containing the frame instances requested, the system calls the *similarity generator*, which returns a sequence of folders in the order specified in Section 7.4.4. A possible sequence can be "M.S. Program", "Ph.D. Students", etc. As the folder in the original query is replaced by the folder "Ph.D. Students", the system returns non-empty answer. Finally, a cooperative answer is responded to the user for asserting that only Ph.D. students were enrolled in the course CIS792.

CHAPTER 8

GENERALIZATION RULES

In chapter 7, we presented *query generalization* mechanisms for answering any queries that reflect erroneous presuppositions with informative messages instead of simply a null answer. The generalizations of any given failed query (i.e., with an empty answer) are derived by incorporating both the folder and type substitutions and weakening search criteria, and the system will be able to conclude a meaningful and cooperative response by looking into a small subset of query generalizations. In general, the results of evaluating these generalized subqueries contain information which is of potential interest to the user. In this chapter, we consider the general boolean queries[1] which produce empty answers. We introduce a *Conjunctive Query Graph* to represent all the possible conjunctive subqueries generated using the generalization algorithm. The generalization algorithm is executed based on this graph in which each of the nodes characterizes the search criteria and the arcs direct to the next possible search criteria to be considered. A most significant feature of the algorithm is its ability to reduce the space of generalized subqueries by restricting accesses to those facts which are effectively needed to answer a query. A set of rules is applied further to attain that property..

8.1 Conjunctive Query

We first focus our discussion on conjunctive queries [2], and then consider the general boolean queries [3] in the next section.

[1]The queries consist of boolean combinations of predicates.
[2]The queries only use AND operator.
[3]The queries use the operators AND, OR and $AND\ NOT$.

8.1.1 Conjunctive Query Graph

We define *the index term set*, $E = \{i_1, \ldots, i_n\}$ to include all index terms or primitive predicate terms[4] appearing in the original query. The power set of E, $P(E)$, is mapped into a *Conjunctive Query Graph*, which represents all the possible conjunctive generalized subqueries by applying the generalization procedure to the original query. The nodes of the graph refer to the conjunctive subqueries which are distinguished between the queries with empty answers and queries with non-empty answers.[5] The arcs of the graph represent the set-inclusion relationship in the power set $P(E)$. The leaves of the graph contain the subqueries which are denoted by the index terms or primitive predicate terms. For instance, Figure 8.1 depicts the *Conjunctive Query Graph* corresponding to the query given in Figure 7.5, where F and T an index terms, and C and A are primitive predicate terms.

An example of *Conjunctive Query Graph* for the query involving two folders is depicted in Figure 8.2. The quest for *Student_Names* involves looking for any two frame instances having the same student name (i.e., $Student\_Name = Name$), where one frame instance is of *Admission_Acc_Letter* type in the *Ph.D.Students* folder, which contains $Date =$ "*Fall* 1990", and the other is of *Q.E.Result* type in the *Q.E.* folder which contains $Date\_Taken \leq$ "*Spring* 1990" and $Outcome =$ "*Pass*".

8.1.2 Generalization

The conjunctive query graph for a query represents all the possible conjunctive subqueries generated in the generalization procedure. Given n_1 subqueries derived from the original query, there are $\sum_{m=1}^{n_1} \prod_{k=1}^{m} \frac{n_1-(k-1)}{m!}$ number of conjunctive subqueries. For determining a meaningful and cooperative response of any given failed query, we examine only a small subset of query generalization, based on a

[4]A primitive predicate term is of the form $i_1@i_2$ or $i@v$, where v is a value, and @ is a comparison operator.

[5]Finally, some nodes of the graph are labeled by the cardinalities of the result sets associated with the queries.

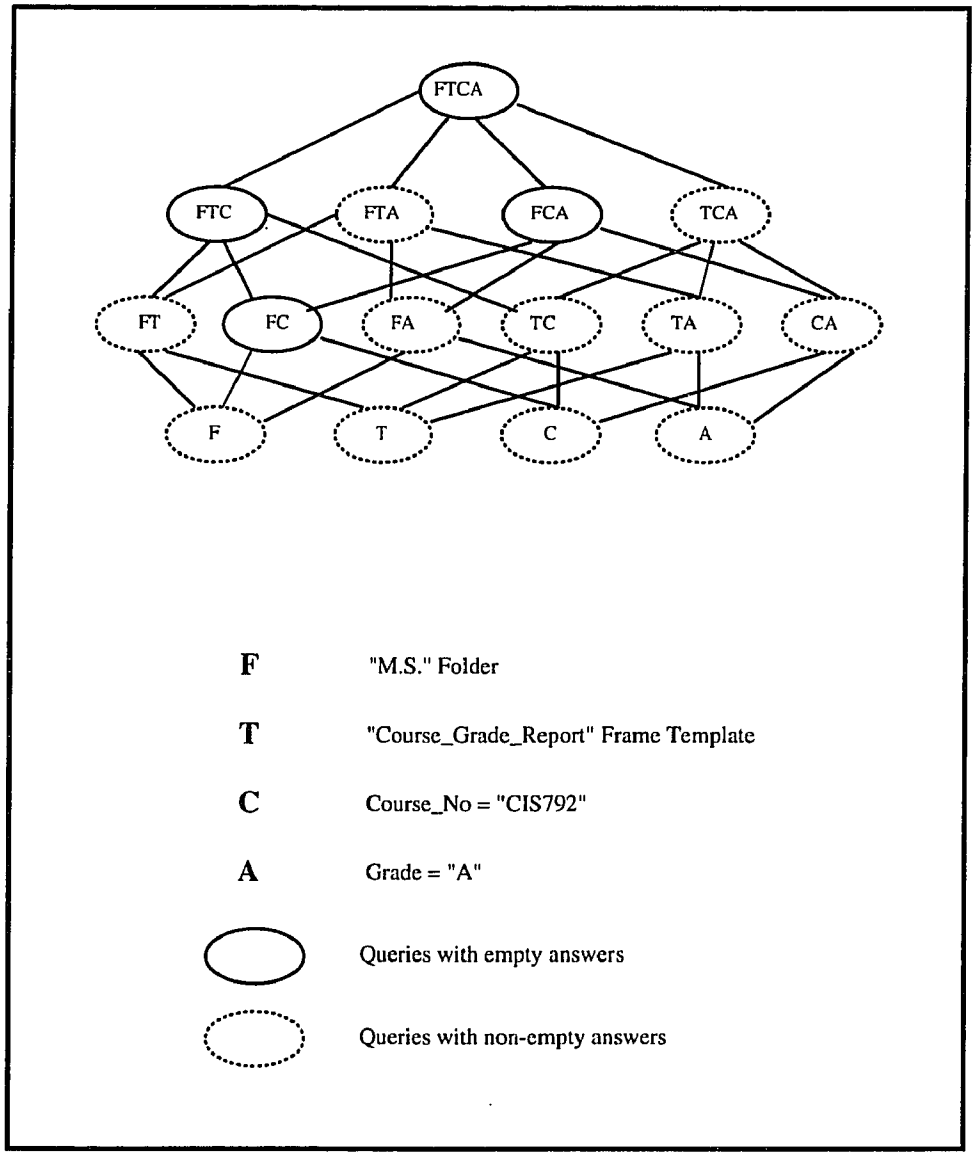

Figure 8.1 Conjunctive Query Graph Corresponding to Figure 7.5

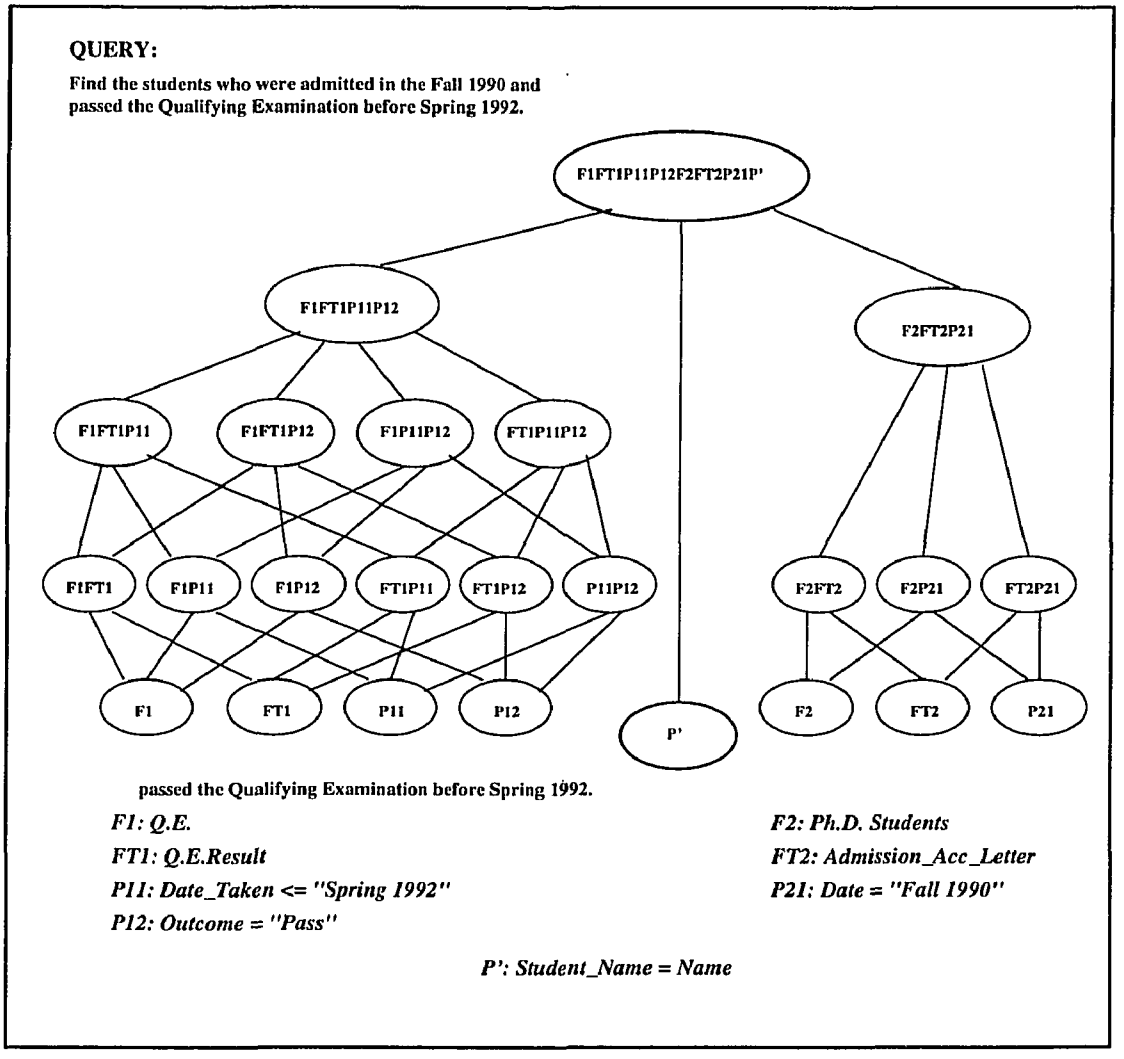

Figure 8.2 Conjunctive Query Graph for the Query Involving Two Folders

constant propagation strategy[101]; that is, the results of the first evaluated subqueries are used to restrict the search space for the following ones.

Algorithm 8.1: (For generating conjunctive query graph of a given query)
The algorithm starts to form subqueries, which are of the index terms or primitive predicate terms appeared in a given query Q_0. Each of the subqueries is represented by a node at the bottom level of the conjunctive query graph. Then the algorithm issues the subqueries from the bottom level of the *Conjunctive Query Graph*[6] and stops as the original query Q_0 is reached.

$New = \{Q_{11}, Q_{12}, \ldots, Q_{1n_1}\}$.[7]

$m = 1$. /*at the first level*/

The subqueries are issued as follows:

1. If $New = \{Q_{m1}, Q_{m2}, \ldots, Q_{mn_m}\}$ contains the n_m subqueries, each having **m terms** (where $n_m = \prod_{k=1}^{m} \frac{n_1-(k-1)}{m!}$) in the level m of the graph, the subqueries in the level $(m+1)$ issued from $Q_{m1}, Q_{m2}, \ldots, Q_{mn_m}$ are put into *Current*,[8] which is the union of the following subqueries:

   $Q_{mi}Q_{mj}$ $(1 \leq i < j \leq n_m)$ denotes the subquery with $\mathbf{m+1}$ **terms** which is the least common parents of Q_{mi} and Q_{mj} in the graph.[9]

[6]All the subqueries are issued in an order such that those in the lower level of *Conjunctive Subqueries Graph* are visited first.

[7]It includes the subqueries in the bottom level.

[8]Furthermore, they are put into two other sets. One, called *Empty*, includes all the subqueries which generate empty answers. Another, called *NonEmpty*, includes all the subqueries which generate non_empty answers.

[9]The subqueries having at least one child in the *Empty* set are put in the *Empty* set, which will not be processed by retrieving the database.

2. **If** *Current* is the original query, the system stops;

   otherwise,

   $New \leftarrow Current$,

   $m = m + 1$, and

   Return to (1).

8.1.3 Information Returned

In a conjunctive query graph, there are nodes containing subqueries which are redundant or irrelevant. A subquery in a node is considered to be redundant if it contains subquery represented by another node which yields the same result. A subquery in a node is considered to be irrelevant with respect to the original query if it does not reflect the intentional goal of the original query.

The following rules can be used to determine which nodes containing the subqueries in a conjunctive query graph should be returned to the user. That is, those nodes containing irrelevant or redundant subqueries are no longer to be in question.

Definition 8.1: An element U of a subset W of $P(E)$ is a *minimal element* of W if there is no element of W strictly included in U.

Definition 8.2: An element U of a subset W of $P(E)$ is a *maximal element* of W if no element of W strictly contains U.

Rule 8.1: (For the subqueries with empty answers)
The only subqueries with empty answers returned to the user are those that are *minimal elements* of the set of subqueries with empty answers.

For instance, in Figure 8.1, $Empty = \{FC, FTC, FCA, FTCA\}$, which is the set of subqueries with empty answers. The result of evaluating the conjunctive subquery FC will be returned to the user, since it is the minimal element of the *Empty* set. The fact that FTC gives an empty answer is an obvious consequence of the fact that FC gives an empty answer.

Rule 8.2: (For the subqueries with non-empty answers)[10]
The only subqueries with non-empty answers returned to the users [11] are those that are *maximal elements* of the set of subqueries giving non-empty answers.

For instance, in Figure 8.1, $NonEmpty = \{F, T, C, A, FT, \ldots, CA, FTA, TCA\}$, which is the set of generalized subqueries with non-empty answers. Only the results of evaluating the conjunctive subqueries FTA and TCA will be returned to the user since they are the maximal elements of *NonEmpty* set. Intuitively, each term of a conjunctive query which gives non-empty answer will also give non-empty answer.

[10]When a maximal query with non-empty result consists only of negated index terms, it is not necessary to mention it in the answer.

[11]Their cardinalities (the number of frame instances which qualify these subqueries) are to be presented to the user at the same time, which can help the user determine the appropriate follow-up queries.

Algorithm 8.2: (The generalization algorithm)

Given a failed query (i.e., it produces an empty answer) and its corresponding conjunctive query graph (which is constructed using **Algorithm 8.1**), the meaningful and cooperative responses can be derived by evaluating the subquery of each node of the graph in the following way:

1. Traverse the graph from the highest level to the bottom level of the graph.

2. For each node at each level, evaluate its subquery.

   (a) If the result of the evaluation of the subquery at the node is a non-empty answer, then assign the subquery with the answer to the *NonEmpty* set and stop traversing all its descendant nodes of the lower levels.

   (b) If the evaluation of the subquery at the node gives an empty answer, then assign the subquery to the *Empty* set, and continue to evaluate the subqueries of its descendant nodes of the lower levels.

   A node is regarded as a minimal element (a significant failure) of the *Empty* set if each of the subqueries of its immediate descendant nodes is evaluated to be a non-empty answer, or if it is at the bottom level of the graph.

3. Determine the maximal elements and the minimal elements of the *NonEmpty* set and the *Empty* set, respectively.

4. Analyze the maximal and minimal elements to obtain the reason for the original query having an empty answer.

8.2 General Boolean Queries

Given any general boolean query, the number of generalized subqueries (i.e., $\sum_{m=1}^{n_1} \prod_{k=1}^{m} \frac{n_1-(k-1)}{m!}$, where n_1 is the number of index and primitive predicate terms) in its corresponding conjunctive query graph becomes large as it (the original query) contains many index terms and primitive predicate terms. Then the process of deriving a meaningful and cooperative answer for a failed query requires to evaluate the generalized subqueries of all the nodes in the graph, and therefore, is inefficient. In the following sections, the reduction of the space of generalized subqueries is presented.

8.2.1 Transformation of DNF

A disjunctive query Q (or Q is in disjunctive normal form (DNF)) is represented as $E_1 + E_2 + \ldots + E_m$, where E_i is either an index term or a primitive predicate term. Then

Property 8.1: A disjunctive query Q gives an empty answer if and only if $(\forall i,\ 1 \leq i \leq m)$ (E_i gives an empty answer).

In general, E_i can be a term which is a conjunction of primitive predicate terms and index terms. We shall call the conjunctive parts of a disjunctive query Q the DNF terms. This **Property8.1** can be used to analyze a disjunctive query with empty answer, by simply determining the evaluation of each of its index terms and primitive predicate terms (or the conjunctive parts) to be empty answer. The following rules can be applied for transforming a general boolean query into one in the disjunctive normal form (DNF).

- Push the operators NOT down to the index terms or primitive predicate terms of the boolean query by applying De Morgan's laws repeatedly.
  For instance, $A\neg(BC) = A(\neg B + \neg C)$
  where A is asserted while B and C are negated.

- Break conjunctions into disjunctions repeatedly using the property of distributivity of AND with respect to OR until the query is of DNF.
  For instance, $A(\neg B + \neg C) = A\neg B + A\neg C$.

8.2.2 Restriction of the Space of Subqueries

Given a query of the disjunctive normal form, applying the **Algorithm 8.1** , the corresponding conjunctive query graph can be constructed by first extracting all the index terms and primitive predicate terms, including the negated terms, from the conjunctive parts of the disjunction of the query. These terms are the subqueries at the bottom level of the conjunctive query graph. The number of subqueries in the *Conjunctive Query Graph* becomes large as there are many index terms and primitive predicate terms in the original query, but most of them are of no interest. Figure 8.1 and 8.4 depict the conjunctive query graphs for the queries $FTCA$ and $FT\neg C\neg E$, respectively.

8.2.2.1 Restrict to Only Conjunctive Compatible Subqueries

Assuming that the query is in disjunctive normal form, we can restrict the space of the relevant subqueries of its corresponding conjunctive query graph for deriving the meaningful and cooperative response if the query gives an empty answer.

Definition 8.3: A subquery U is compatible with Q if each index term or primitive predicate term of U has the same signature[12] as in Q.

[12]If an index term is negated, its signature is $-$, or $+$ otherwise.

Rule 8.3: The generalized subqueries are restricted to only conjunctive compatible subqueries.

According to the **Rule 8.3**, the *Conjunctive Query Graph* can be used as long as the nodes of the bottom level of the graph are restricted to contain only the index terms and primitive predicate terms in the disjunctive query.
For instance, the nodes of the bottom level of the graph are A, $\neg B$, and $\neg C$. It is not necessary to consider B, C, and $\neg A$.

8.2.2.2 Using the Covering Set of DNF

Given a query Q_0: $A\neg(BC)$ which can be expressed in terms of $A\neg B + A\neg C$, there corresponds a conjunctive query graph which contains only generalized conjunctive compatible subqueries, as shown in Figure 8.3. The **Property 8.1** postulates that if Q_0 produces an empty answer provided both DNF terms. $A\neg B$ and $A\neg C$ must produce empty answers, since Q_0 is the disjunction of these two terms (i.e., $A\neg B + A\neg C$). This motivates us to introduce and investigate the covering set of a query.

Given a query of disjunctive normal form, there corresponds a conjunctive query graph in which each node represents a conjunctive compatible subquery of the query.

Definition 8.4: The *covering set* of the query is the set of nodes in which the subquery of each node is included in at least one of the DNF terms [13] of the query, and the set of nodes contains all the index terms and primitive predicate terms of the query.

[13]Each conjunctive part of a disjunctive query is called DNF terms.

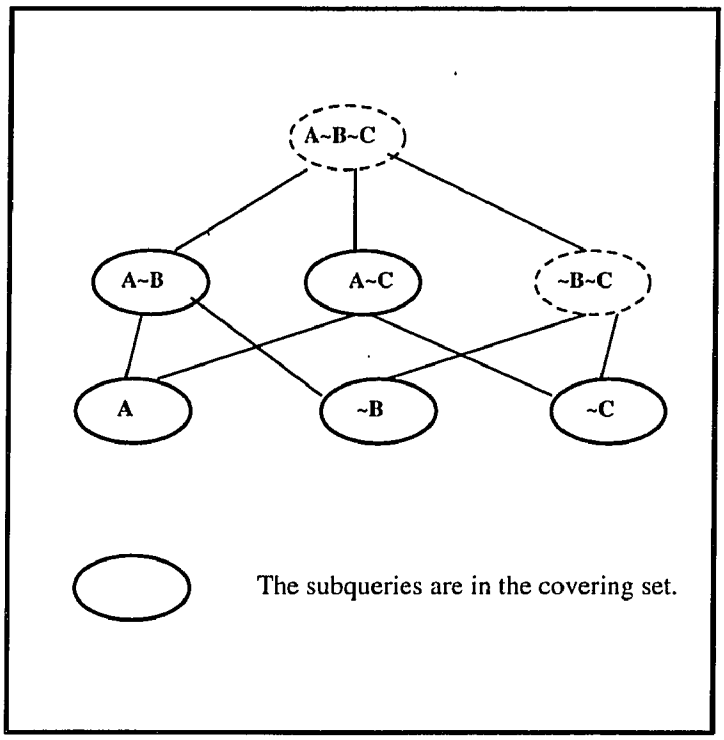

Figure 8.3 An Example of Conjunctive Compatible Subqueries

The DNF terms are the maximal elements of the covering set.[14]

Rule 8.4: The generalized subqueries are restricted to the *covering set* of a disjunctive query. The subqueries not in the *covering set* of the query are considered to be irrelevant.

When a disjunctive query gives an empty answer, each one of its DNF terms also gives an empty answer. Given a disjunctive query with an empty answer, the **Algorithm 8.1** for constructing a conjunctive query graph begins from selecting all the compatible index terms and primitive predicate terms from the query and terminates as reaching the nodes containing subqueries which are the DNF terms of the query.

[14]A subquery X is included in a DNF term Y if every index term or primitive predicate term in X is appeared in Y. Some nodes are included in more than one DNF terms.

For deriving the meaningful and cooperative response of the query, **Algorithm 8.2** traverses all the nodes of the covering set, starting from the nodes containing the DNF terms of the query.

8.3 Example

The following example demonstrates our approach. Consider a query: "Find all Ph.D. students who were not enrolled in courses CIS792 and ENG543." The information can be searched through the *Full_Transcript* (denoted as T) of each student in the *Ph.D.Students* folder (F) which contains no $Course\_No$ = "$CIS792$" (C) and $Course\_No$ = "$ENG543$" (E). The query can be represented as $FT\neg(CE)$.

- The system first transforms the query into one which is in DNF using the rules given in Section 8.2.1.
  $FT\neg(CE) = FT(\neg C + \neg E) = FT\neg C + FT\neg E$.

- For the query $FT\neg C + FT\neg E$, only the index terms F and T and the primitive predicate terms $\neg C$ and $\neg E$ are taken into consideration for constructing a conjunctive query graph. The graph contains only conjunctive compatible subqueries and is depicted in Figure 8.4.

- Every node of the graph is associated with a subquery. Then the covering set of the original query, which is shown in Figure 8.4, contains all the nodes, each of whose subqueries is included in a DNF term of the given query, and every index term and primitive predicate term in the given query must be in one of these subqueries.

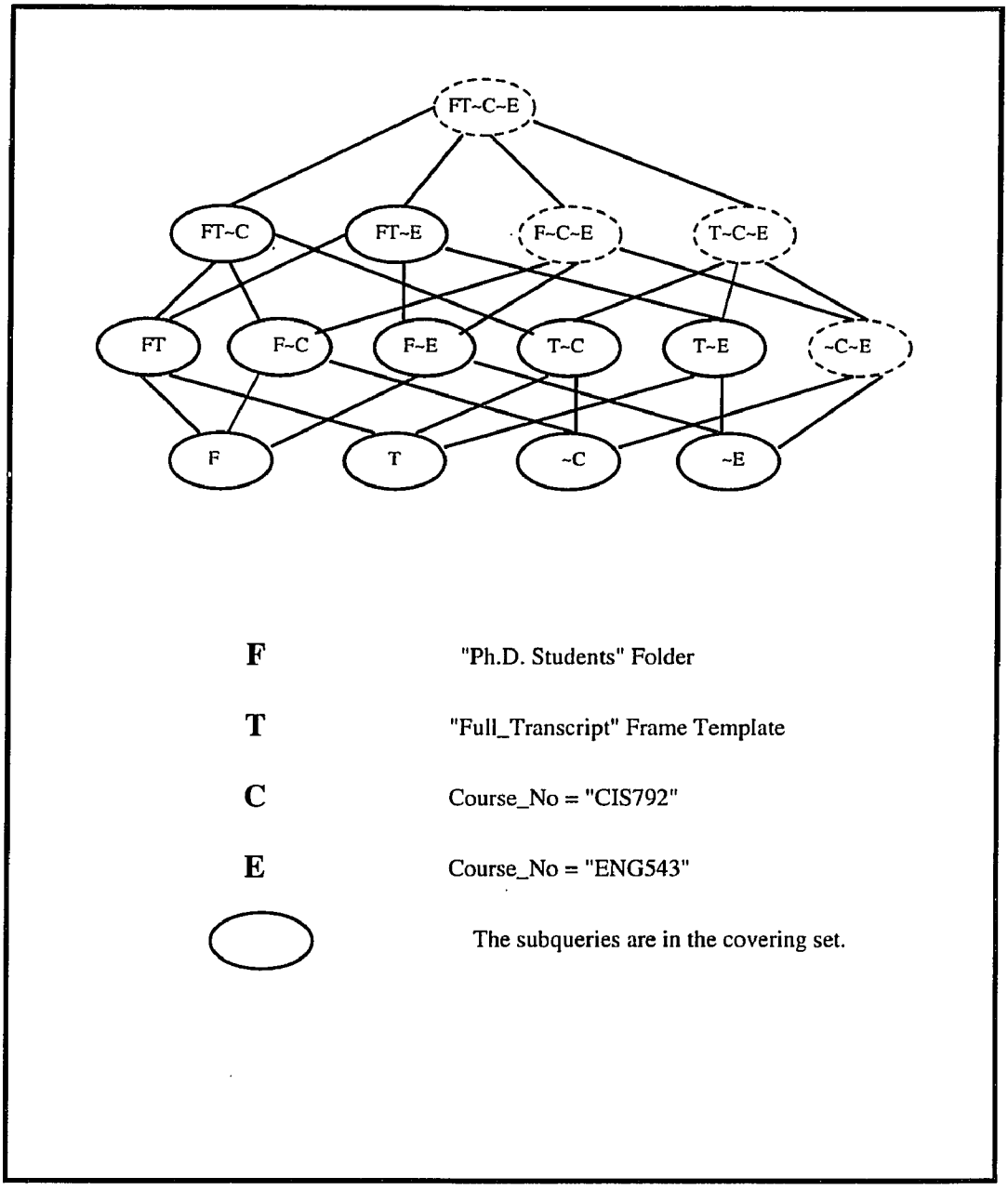

Figure 8.4 Conjunctive Compatible Subqueries

8.4 Remarks

The main objective of implementing the generalization algorithm is for generating the relevant, generalized subqueries for a given query. Each of the subqueries, which is called a DNF term, is in conjunctive normal form. The generation of the subqueries is based on the following observations. If a conjunctive subquery Q_1 which is included in a conjunctive subquery Q_2, gives an empty answer, then Q_2 will give an empty answer. It is important to avoid to process subquery Q_2. Similarly, if Q_1 is not in the covering set of a query, then Q_2 is not in the covering set either.

Given a failed query, the algorithm can be used to construct its *covering set*, from which the *minimal* subqueries with empty answers and *maximal* subqueries with non-empty results can be obtained. The evaluation of these *minimal* subqueries with empty answers derives a more precise result, which explicates why the original query yields an empty answer. The evaluation of these *maximal* subqueries with non-empty results can determine the follow-up queries to be evaluated next.

Returning the cardinalities of these result sets instead of these result sets themselves[15] prevents the user flooded with information in these large result sets, since the cardinalities of these sets can give enough clues to help determine the reason of empty answers produced and the appropriate follow-up queries.

[15] i.e., returning the number of frame instances which qualify a subquery instead of their contents.

CHAPTER 9

SUBSTITUTION RULES

In Chapter 8, we present the generalization mechanisms to distinguish the fake empty answer from the genuine empty answer. In this chapter, we will present a methodical approach to analyzing the results of executing generalization which is discussed in Chapter 8, and propose a strategic scheme of various substitutions that may need to produce a meaningful and cooperative response according to the different situations. A rule execution scheme is designed for efficiently applying the possible substitutions to generate subqueries when a rule is executed.

We use rules, in first order logic, to define the orderly sequences of the folders and frame templates, which are used to replace the folders and the frame templates in the original query.

9.1 Determining Various Substitutions

In Chapter 8, we presented the transformation of query into one in a disjunctive normal form, which contains compatible conjunctive subqueries, called the DNF terms of the query. The covering set of the query is the set of subqueries such that each of the subqueries is included in at least one of the DNF terms of the query, and every index term and primitive predicate term of the query must be in one of these subqueries. Then, the minimal subqueries with empty answers in the covering set can be used to explain why the original query yields an empty answer. And the maximal subqueries with non-empty results in the covering set, together with the number of frame instances involved, can be used to determine which appropriate subqueries to be considered next.

Let Min and Max be the sets of minimal subqueries and maximal subqueries, respectively. In this section, we will derive various criteria of different ways of substi-

tution, which may take place in the process of further generalization, by taking these two sets of subqueries into consideration.

Given a disjunctive original query Q_0, if every DNF term $FTp_1p_2 \ldots p_m$ in Q_0 has a genuine empty answer, then the empty answer of Q_0 is genuine. **Algorithm 9.1** is used to determine whether $FTp_1p_2 \ldots p_m$ has a genuine empty answer.

Algorithm 9.1:

$A = \{p_1, p_2, \ldots, p_m\}$, where ($p_i$ for $i = 1, \ldots, m$ is a primitive predicate term which includes a comparison between the *attributes* or between an *attribute* and a *value*);

F denotes a folder; T denotes a frame template;

$FTp_1p_2 \ldots p_m \in Empty$;

Min denotes the minimal query set in which each subquery has an empty answer;

Max denotes the maximal query set in which each subquery has a non-empty answer;

BEGIN

if $Min = \{FTp_1p_2 \ldots p_m\}$ **then**{ the empty answer of the original query is genuine}

/* *case1.1* : only the original query is in the Min.*/

else{ /* *case1.2* : the empty answer of the original query is fake.*/

if $Fp_1p_2 \ldots p_m \in Max$ **then**{ do frame template substitution in folder F }

/* *case1.3* : there is information in folder F but other types of frame templates.*/

else{ /* *case1.4* : there is no information in folder F (with different reasons).*/

if $Tp_1p_2 \ldots p_m \in Max$ **then**{ do folder substitution over frame template T }

/* *case1.5* : there is information with type of frame template T but not in the folder F.*/

else{ /* *case1.6* : there is no information with type of frame template T.*/

if $p_1p_2 \ldots p_m \in Max$ **then**{ *case1.7* : do folder substitution and frame template substitution }

```
            else{ /* case1.8 : there is no information satisfying all predicates
                        in the system.*/
                  Return{ there is no such information in the system }
            }
        }
    }
}
END
```

9.2 Characterization of Returned Information

A logical folder organization (as shown in Figure 7.1) mimics the filing organization perceived by the user. A document type hierarchy represents the document classification in terms of a structural organization of the frame templates in which each of the templates describing the properties of a class of documents. We will proceed the folder and frame template substitutions based on the logical folder organization and document type hierarchy, respectively. In **Algorithm 9.1**, we check $Fp_1p_2 \ldots p_m$ prior to $Tp_1p_2 \ldots p_m$, because the folders have more semantic characteristics than the frame templates.

Proposition 9.1: Let $\mathbf{S} = (F|T)(p_1p_2 \ldots p_m)$.[1]

(i) If $\mathbf{S} \notin Max$, then $\mathbf{S} \in Empty$.

(ii) If $\mathbf{S} \notin Empty$, then $\mathbf{S} \in Max$.

The reason for checking only the Max set in *case1.3* and *case1.5* is based on the **Proposition 9.1**. Furthermore, **Proposition 9.1**(i) gives the explanation for *case1.4* and *case1.6*. In *case1.4*, the subquery $Fp_1p_2 \ldots p_m$ returns an empty

[1] $(F|T)(p_1p_2 \ldots p_m)$ reads as $F(p_1p_2 \ldots p_m)$ or $T(p_1p_2 \ldots p_m)$.

answer, so there is an indication of no information satisfying all predicates in folder F. In *case1.6*. the subquery $Tp_1p_2 \ldots p_m$ returns an empty answer, so there is no such frame instances of the frame template type T satisfying all predicates.

Proposition 9.2: If $p_1p_2 \ldots p_m \in Max$, then $(F|T)(p_1p_2 \ldots p_m) \in Empty$.

Proposition 9.2 states that the subquery $Fp_1p_2 \ldots p_m$ and the subquery $Tp_1p_2 \ldots p_m$ must have empty answers when $p_1p_2 \ldots p_m$ is in the Max set. So we need both folder and frame template substitutions in *case1.7*.

Proposition 9.3: Let $\mathbf{S} = (F|T)(p_1p_2 \ldots p_m)$. If $\mathbf{S} \notin Max$ and $p_1p_2 \ldots p_m \notin Max$, then $p_1p_2 \ldots p_m \in Empty$.

Proposition 9.3 supports *case1.8*: when the subquery $Fp_1p_2 \ldots p_m$ and the subquery $Tp_1p_2 \ldots p_m$ return the empty answers, the subquery $p_1p_2 \ldots p_m$ must be in the $Empty$ set if it is not in the Max set. So it concludes that there is no information satisfying all the predicates, $p_1, p_2, \ldots, p_m$, in the system.

9.3 Informal Specification of Substitutions

In **Algorithm 9.1**, there are three ways of folder and frame template substitutions. In this section, various strategies for accomplishing these substitutions at different situations are described.

9.3.1 Do Folder Substitution over a Specific Frame Template T

From the results of the subquery $Fp_1p_2 \ldots p_m$ having an empty answer and the subquery $Tp_1p_2 \ldots p_m$ being in the Max set, in *case1.5*, the system concludes that there are frame instances of type T in the file organization, which satisfy all the primitive predicate terms $p_1, p_2, \ldots, p_m$, but there is no frame instance in the folder F satisfying these predicates. Thus, the folder F in the original query will be replaced by a sequence of folders, which are associated with T, in the logical folder organization. The order of folders in the sequence to be used for substitutions is determined in terms of the *similarities*, and the *semantic* and *structural interdependencies* defined in Chapter 7:

1. From the logical folder organization, obtain an orderly sequence of folders which are the candidates of folder substitution. The folders in the sequence are in the order of the following:

   - The folders having higher similarities with F are prior to the folders having lower similarities.

   - For the folders which have the same similarities with F, the priorities of taking folders into consideration are:
     - the folders which are *partial_joint* with F to be first,
     - the folders which are not *coverings* of F next, and
     - the folders which are *coverings* of F last.

- For the folders, which have the same similarities and same semantic inter-dependency with F, the folders having more common frame templates with F is prior to the others having less common frame templates.[2]

2. From the obtained sequence folders, substitute the folders, which are *joint* with F over frame template T, for F in the original query.

Example 9.1: Given the query in the Figure 7.5, from the results of evaluating its corresponding conjunctive query graph as shown in Figure 8.1, we conclude that there are frame instances of type "Course_Grade_Report" in the entire system which satisfy predicates C and A, but there is no frame instance satisfying these predicates in the folder "M.S. Students" and other folders associated with frame template "Course_Grade_Report". That, "Financial Assistantship", "M.S. Program", "Ph.D.Students", "Ph.D Program", "Academic Affairs", etc, is a sequence of folders which are the candidates for folder substitution. The folder "Financial Assistantship" should be eliminated from the sequence because it does not joint with "M.S. Students" over "Course_Grade_Report". And the remaining folders of the sequence which are joint with "M.S. Students" over "Course_Grade_Report" are used to substitute for the folder "M.S. Students" in the query of Figure 7.5.

[2]We use the concept of *structural similarity*, which means that a folder containing more instances of the same frame template type is considered as more similar. For simplicity, the degree of structural similarity can be computed by dividing the total number of frame instances in the folder by the number of their distinct frame template types. Thus, a folder of highest degree of structural similarity is first taken into consideration. If two folders have the same degree of structural similarity, then the folder having the smaller number of frame template types will be considered first. Otherwise, one of these folders can be selected arbitrarily as the tie-breaker.

9.3.2 Do Frame Template Substitution in a Specific Folder F

For *case1.3*, since the subquery $Fp_1p_2 \ldots p_m$ is in the Max set, there are frame instances in the folder F satisfying all the primitive predicate terms. The system will proceed frame template substitutions in the folder F disregarding whether there are frame instances of type T, which are satisfying all the predicates. A sequence of frame templates, which are associated with F, in the document type hierarchy is used to substitute for the frame template T in the original query.

- The frame templates in the document type hierarchy, which are used to substitute for the frame template T in the original query, must satisfy the following conditions:
  - The frame templates are associated with the folder F.
  - The frame templates include all the attributes of the primitive predicate terms, $p_1, p_2, \ldots, p_m$.
- The system assigns the order of the templates for substitutions based on the *Type Substitution Rules* specified in Section 7.5.

Example 9.2: Given the following formal query:

SELECT Ph.D.Students(Grade_Report).Student_Name
FROM Ph.D.Students(Grade_Report)
WHERE
Ph.D.Students(Grade_Report).Course_No = "$ENG543$" **AND**
Ph.D.Students(Grade_Report).Grade = "A";

The conjunctive query graph for this query is depicted in Figure 9.1, which yields the following results of evaluating the subqueries:

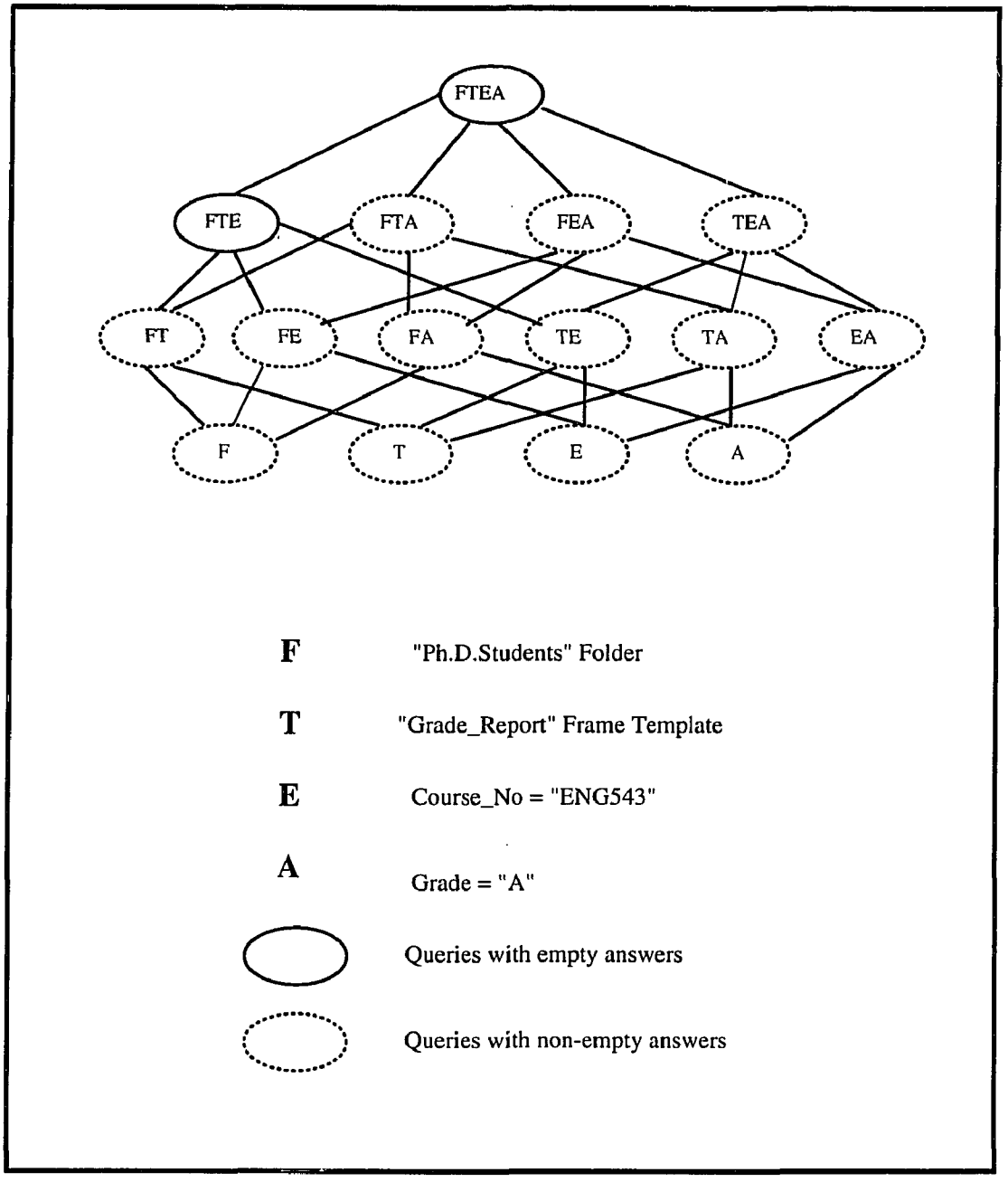

Figure 9.1 Conjunctive Query Graph of Example 9.2

(*i*) $Max = \{FTA, FEA, TEA\}$.

(*ii*) $Min = \{FTE\}$.

In analyzing the Max and Min, the system can conclude that:

1. There are frame instances satisfying predicates E and A in the folder F. That is, there is at lease one Ph.D. student who received a grade A for the course ENG543 (from FEA in the Max set).

2. There is no frame instance of the frame template type T, satisfying the primitive predicate term E in the folder F (from FTE in the Min set).

The system needs to find the appropriate frame template in the document type hierarchy to replace the frame template "Grade_Report" in the "Ph.D. Students" folder. A possible sequence of substitutions can be "Course_Grade_Report", "Full_Transcript", "Transcript", etc, according to the substitution rules defined in section 7.5. Since the frame template "Full_Transcript" contains all the attributes appeared in the primitive predicate terms E and A, and is associated with the "Ph.D. Students" folder, it substitutes for the frame template "Grade_Report" in the original query. If the query still returns an empty answer after the substitution, the system needs to find one of the other frame templates to be a substitute for T such that the query returns non-empty answer.

From the result of TEA in the Max set through evaluating the conjunctive query graph, we conclude that there are frame instances with type "Grade_Report" in the system, which satisfy all the predicates, but they are not in the folder "Ph.D. Students". Although the frame template "Grade_Report" is associated with the folder "Ph.D.Students" since the subquery FT returns non-empty answer, and the fact that FTE is in the Min set, we know that there is no frame instance of

type "Grade_Report" in the folder "Ph.D.Students", which satisfies the primitive predicate term "Course_No = ENG543".

9.3.3 Do Folder and Frame Template Substitution at the Same Time

The evaluating results of the subqueries $Fp_1p_2 \ldots p_m$ and $Tp_1p_2 \ldots p_m$ having empty answers, lead us to conclude that there is no frame instance of type T, which satisfies all the predicates, and there is no frame instance satisfying all the predicates in the folder F. For *case1.7*, since the subquery $p_1p_2 \ldots p_m$ is in the Max set, there are frame instances in the system satisfying all predicates. We try to find the folders containing these frame instances with the unknown frame templates satisfying all the predicates in the system using the folder and frame template substitutions.
The system proceeds substitutions as follows:

1. Do frame template substitution in the entire system.

   We get the appropriate frame templates in the document type hierarchy to substitute for the frame template T in the original query. Each of the frame templates contains all the attributes of the primitive predicate terms $p_1, p_2, \ldots, p_m$. The system assigns the order of templates for substitutions based on the *Type Substitution Rules.*

2. Do folder substitution.

   The folder substitutions over these frame templates can be executed as in section 9.3.1.

Example 9.3: Given the following formal query:

SELECT M.S. Students(Grade_Report).Student_Name

FROM M.S. Students(Grade_Report)

WHERE

M.S. Students(Grade_Report).Course_No = "*CIS*792" **AND**

M.S. Students(Grade_Report).Grade = "*A*";

From the conjunctive query graph shown in Figure 9.2, we conclude that there is no frame instance with the type "Grade_Report" in the system satisfying the predicates C and A (from TC in the Min set), and there is no frame instance satisfying these predicates in the folder "M.S. Students" either (from FC in the Min set). Then a possible sequence of frame template substitutions can be "Course_Grade_Report", "Full_Transcript", "Transcript", etc. Each of these frame templates contains the attributes "Course_No" and "Grade". From the previous Example 9.1, the sequence of folder substitutions consists of "M.S. Program", "Ph.D. Students", "Ph.D. Program", "Academic Affairs", etc. Thus, the sequence of folder over template substitutions can be "M.S. Program" over "Course_Grade_Report", " M.S. Program" over "Full_Transcript", " M.S. Program" over "Transcript", ..., "Ph.D. Students" over "Course_Grade_Report", "Ph.D. Students " over "Full_Transcript", "Ph.D. Students " over "Transcript", etc. The process stops with a meaningful response.

As a matter of fact, these is another sequence of folder over frame template substitutions, in which, for each template substitute, such as "Course_Grade_Report", we look into the folders "M.S. Program", "Ph.D. Students", etc.

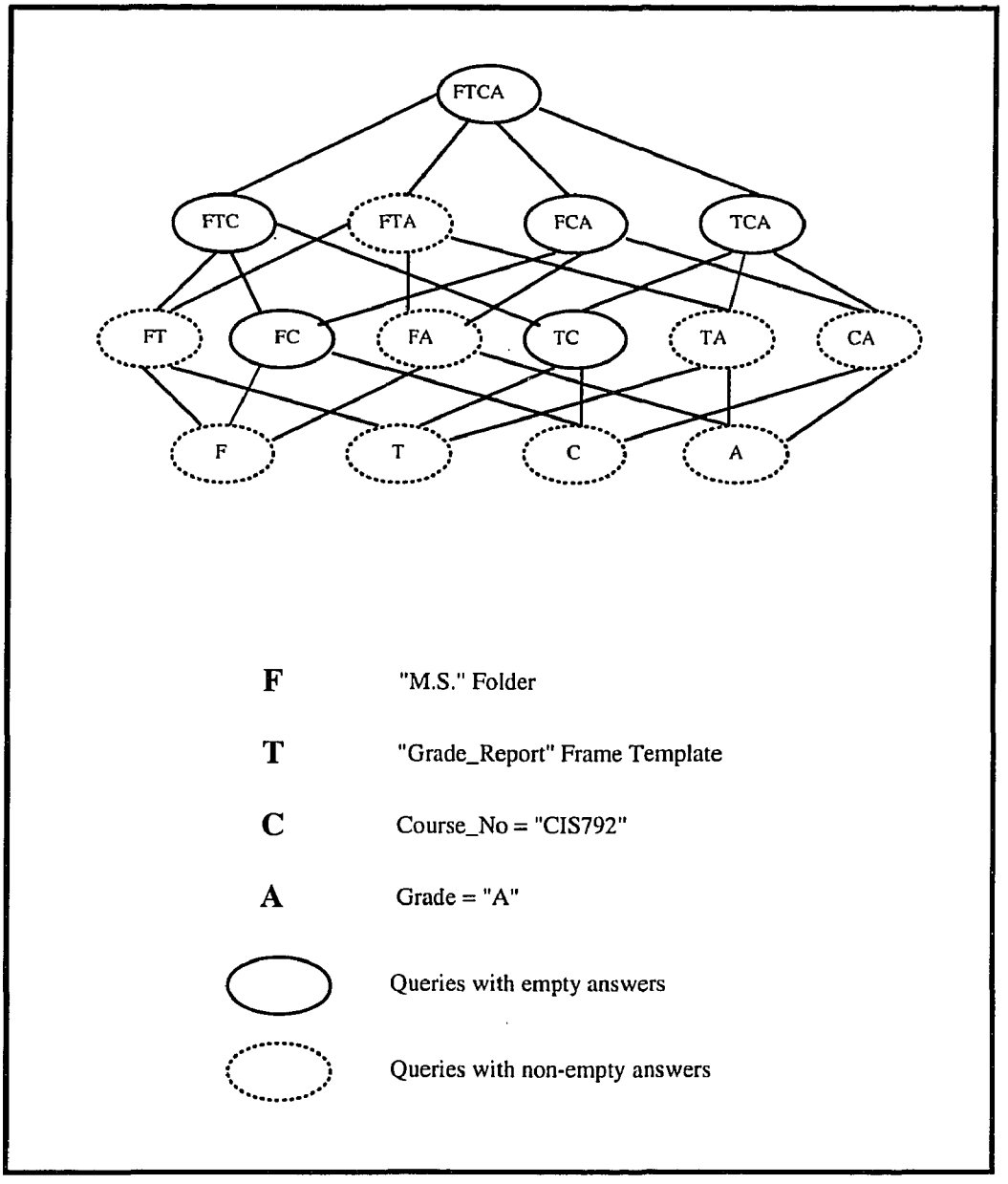

Figure 9.2 Conjunctive Query Graph of Example 9.3

9.4 Formal Representation of Substitutions

We described the strategies of various folder and frame template substitutions in the previous section. In this section, a formal representation of substitutions is given in terms of substitution rules, which are defined in first order logic.

9.4.1 Database Structure Representation

The following meta predicates are used to define the substitution rules:

- **Folder**(f): f is a folder.
- **FrameTm**(ft): ft is a frame template.
- **FolderQy**(q, f): a folder f appears in the query q.
- **FrameTmQy**(q, ft): a frame template ft appears in the query q.
- **IndexTmQy**(q, T): T is an index term part of the query q, which is of the form $Folder(FrameTemplate)$.
- **PredicateQy**(q, p): p is a primitive predicate term in the query q.
- **ISA**(x, y): x is a subtype of y in the document type hierarchy.
- **Sibling**(ft_1, ft_2): ft_1 and ft_2 are siblings in the document type hierarchy.
- **Associate**(f, ft): a folder f is associated with a frame template ft.
- **Att_Predicate**(p, a): an attribute a appears in the predicate p.
- **Att_FrameTm**(ft, a): the frame template ft contains an attribute a.
- **PriorFolder**(f, f_1, f_2): a folder f_1 is prior to a folder f_2 in the sequence of folder substitutions for the folder f.

- **PriorFrameTm**(ft, ft_1, ft_2): a frame template ft_1 is prior to a frame template ft_2 in the sequence of frame template substitutions for the frame template ft.

- **Prior_to_All**(f, f'): f' has the highest priority in the current sequence of folder substitutions for f.

- **Prior_to_All**(ft, ft'): ft' has the highest priority in the current sequence of frame template substitutions for ft.

- **EmptyAnswer**(q): the result of evaluating query q is an empty answer.

- **Similarity**(f_1, f_2, s): the similarity between a folder f_1 and a folder f_2 is s.

- **PartialJoint**(f_1, f_2, f): the semantic interdependency between a folder f_1 and a folder f_2 is a *Partial_Jointness* with respect to the folder f (f_1 and f_2 are partially joint with respect to f).

- **Covering**(f_1, f_2): the semantic interdependency between a folder f_1 and a folder f_2 is a *Covering* (f_1 covers f_2).

- **Disjoint**(f_1, f_2): the structural interdependency between a folder f_1 and a folder f_2 is a *disjointness* (f_1 and f_2 are disjoint).

- **Joint**(f_1, f_2, ft): the structural interdependency between a folder f_1 and a folder f_2 is a *jointness* with respect to a common frame template ft (f_1 and f_2 are joint with respect to ft). [3]

---

[3]The relationships among **Disjointness**, **Jointness**, **Partial_Jointness** and **Covering** are:
Disjoint$(f_1, f_2) \Leftrightarrow (\forall ft)(\neg$**Joint**$(f_1, f_2, ft))$
Covering$(f_1, f_2) \Rightarrow (\exists ft)($**Joint**$(f_1, f_2, ft))$
Covering$(f_j, f_{j1}) \wedge$ **Covering**(f_j, f_{j2}), where $f_j \subseteq f_{j1} \cup f_{j2}$ and $f_{jk} \neq empty, (k = 1, 2)$
$\Rightarrow$ **PartialJoint**(f_{j1}, f_{j2}, f_j)

- **SubstitutedFolder**(f, f_1): f_1 has been used to replace the folder f in the query.

- **SubstitutedFrameTm**(ft, ft_1): ft_1 has been used to replace the frame template ft in the query.

- **FrameTm_Rel_Predicate**(p, ft): the frame template ft contains all the attributes appearing in the primitive predicate term p of an original query.

- **Folder_Substitution**(T, T', f, f'): the index term part T in the original query is transformed into T' by substituting the folder f' for f.

- **FrameTm_Substitution**(T, T', ft, ft'): the index term part T in the original query is transformed into T' by substituting the frame template ft' for ft.

- **Generalize_Query**(q, q', f, f'): the original query q is transformed into the query q' by substituting the folder f' for the folder f.

- **Generalize_Query**(q, q', ft, ft'): the original query q is transformed into the query q' by substituting the frame template ft' for ft.

9.4.2 Rules for Specifying the Substitution Priority

The following rules define an orderly sequence of folders and frame templates to accomplish the substitutions. The order of folder substitutions is defined in **Rule 9.1**, **Rule 9.2**, and **Rule 9.3**, and the order of frame template substitutions is defined in **Rule 9.4**.

Rule 9.1: (For the folders having different similarities with a specific folder f)

For $(q, \mathbf{Folder}(f_1), \mathbf{Folder}(f_2), \mathbf{FolderQy}(q, f))$

$\mathbf{Similarity}(f, f_1, s_1) \wedge \mathbf{Similarity}(f, f_2, s_2) \wedge s_1 > s_2 \wedge$

$\neg\mathbf{SubstitutedFolder}(f, f_1) \wedge \neg\mathbf{SubstitutedFolder}(f, f_2)$

$\rightarrow \mathbf{PriorFolder}(f, f_1, f_2)$

Rule 9.2: (For the folders having same similarities with a specific folder f)

For $(q, \mathbf{Folder}(f_1), \mathbf{Folder}(f_2), \mathbf{FolderQy}(q, f))$

$\mathbf{Similarity}(f, f_1, s_1) \wedge \mathbf{Similarity}(f, f_2, s_2) \wedge s_1 = s_2 \wedge$

$((\exists f')(Folder(f') \wedge \mathbf{PartialJoint}(f, f_1, f')) \wedge$

$((\nexists f'')(Folder(f'') \wedge \mathbf{PartialJoint}(f, f_2, f'')) \wedge$

$\neg\mathbf{SubstitutedFolder}(f, f_1) \wedge \neg\mathbf{SubstitutedFolder}(f, f_2)$

$\rightarrow \mathbf{PriorFolder}(f, f_1, f_2)$

Rule 9.3: (For the folders having same similarities with a specific folder f)

For $(q, \mathbf{Folder}(f_1), \mathbf{Folder}(f_2), \mathbf{FolderQy}(q, f))$

$\mathbf{Similarity}(f, f_1, s_1) \wedge \mathbf{Similarity}(f, f_2, s_2) \wedge s_1 = s_2 \wedge$

$\neg\mathbf{Covering}(f, f_1) \wedge \mathbf{Covering}(f, f_2) \wedge$

$\neg\mathbf{SubstitutedFolder}(f, f_1) \wedge \neg\mathbf{SubstitutedFolder}(f, f_2)$

$\rightarrow \mathbf{PriorFolder}(f, f_1, f_2)$

Rule 9.4: (For the frame templates in the document type hierarchy)

For $(q, \mathbf{FrameTm}(ft'), \mathbf{FrameTm}(ft''), \mathbf{FrameTmQy}(q, ft))$

$\neg\mathbf{SubstitutedFrameTm}(ft, ft') \wedge \neg\mathbf{SubstitutedFrameTm}(ft, ft'') \wedge$

$\mathbf{Sibling}(ft, ft') \wedge \mathbf{ISA}(ft, ft'')$

$\rightarrow \mathbf{PriorFrameTm}(ft, ft', ft'')$

9.4.3 Substitution Rules

Definition 9.1 defines the *current* folder f', which is prior to any folders in the current sequence of folder substitutions for the folder f, and the *current* frame template ft', which is prior to any frame templates in the current sequence of frame template substitutions for the frame template ft.

Definition 9.1: (Prior_to_All)

Let $S_f = \{f_i | Folder(f_i)(1 \leq i \leq n)\}$.

Let $S_{ft} = \{ft_j | FrameTm(ft_j)(1 \leq j \leq m)\}$.

- For $(q, \mathbf{Folder}(f'), S_f, \mathbf{FolderQy}(q, f))$

  $\mathbf{Prior\_to\_All}(f, f') \leftrightarrow \forall(f_i \in S_f)(1 \leq i \leq n)\mathbf{PriorFolder}(f, f', f_i)$

- For $(q, \mathbf{FrameTm}(ft'), S_{ft}, \mathbf{FrameTmQy}(q, ft))$

  $\mathbf{Prior\_to\_All}(ft, ft') \leftrightarrow \forall(ft_j \in S_{ft})(1 \leq j \leq m)\mathbf{PriorFrameTm}(ft, ft', ft_j)$

Rule 9.5 defines the folder substitution over a specific frame template ft. The folder f' is a subsitute for the folder f in the original query, such that the index term part T of the original query is transformed into T'.

Rule 9.5: (For the folder substitution over a specific frame template)

For $(q, \mathbf{Folder}(f'), \mathbf{FolderQy}(q, f), \mathbf{FrameTmQy}(q, ft), \mathbf{IndexTmQy}(q, T), T')$

$\mathbf{Prior\_to\_All}(f, f') \wedge \mathbf{Joint}(f, f', ft)$

$\rightarrow \mathbf{Folder\_Substitution}(T, T', f, f')$

Definition 9.2 defines the concept of a frame template related to a predicate. That is, the frame template ft contains all the attributes which appear in the predicate p of the query q.

Definition 9.2: (**FrameTm_Rel_Predicate**)

For $(q, \mathbf{PredicateQy}(q,p), \mathbf{FrameTm}(ft))$

$\mathbf{FrameTm\_Rel\_Predicate}(p, ft)$

$\leftrightarrow \forall(a)(\mathbf{Att\_Predicate}(p,a) \rightarrow \mathbf{Att\_FrameTm}(ft,a))$

Rule 9.6 defines the frame template substitution associated with a specific folder f. The frame template ft' is a subsitute for the frame template ft in the original query q, such that the index term part T of the original query is transformed into T'.

Rule 9.6: (For the frame template substitutions associated with a specific folder)

For $(q, \mathbf{Folder}(f), \mathbf{FrameTm}(ft'), \mathbf{FrameTmQy}(q,ft), \mathbf{PredicateQy}(q,p),$

$\mathbf{IndexTmQy}(q,T), T')$

$\mathbf{Prior\_to\_All}(ft,ft') \wedge \mathbf{FrameTm\_Rel\_Predicate}(p,ft') \wedge \mathbf{Associate}(f,ft)$

$\rightarrow \mathbf{FrameTm\_Substitution}(T,T',ft,ft')$

Rule 9.7 defines the frame template substitutions applied in the entire system, if $\mathbf{Associate}(f,ft)$ is relaxed from **Rule 9.6**.

Rule 9.7: (For the frame template substitutions in the system)

For $(q, \mathbf{FrameTm}(ft'), \mathbf{FrameTmQy}(q,ft), \mathbf{PredicateQy}(q,p),$

$\mathbf{IndexTmQy}(q,T), T')$

$\mathbf{Prior\_to\_All}(ft,ft') \wedge \mathbf{FrameTm\_Rel\_Predicate}(p,ft')$

$\rightarrow \mathbf{FrameTm\_Substitution}(T,T',ft,ft')$

Rule 9.8: (The original query q is transformed into q' by substituting the folder f' for f or the frame template ft' for ft.)

- For $(q, \mathbf{Folder}(f'), \mathbf{FolderQy}(q, f), \mathbf{IndexTmQy}(q, T), T', q')$

  $\mathbf{Prior\_to\_All}(f, f') \wedge \mathbf{Folder\_Substitution}(T, T', f, f')$

  $\rightarrow \mathbf{Generalize\_Query}(q, q', f, f')$

- For $(q, \mathbf{FrameTm}(ft'), \mathbf{FrameTmQy}(q, ft), \mathbf{IndexTmQy}(q, T), T', q')$

  $\mathbf{Prior\_to\_All}(ft, ft') \wedge \mathbf{FrameTm\_Substitution}(T, T', ft, ft')$

  $\rightarrow \mathbf{Generalize\_Query}(q, q', ft, ft')$

Rule 9.9: (In the case of the generalized query q' still having an empty answer, f' needs to be identified as **SubstitutedFolder** in the current sequence of folder substitutions. Similarly, ft' needs to be identified as **SubstitutedFrameTm** in the current sequence of frame template substitutions.)

- For $(q, \mathbf{Folder}(f'), \mathbf{FolderQy}(q, f), q')$

  $\mathbf{Prior\_to\_All}(f, f') \wedge \mathbf{Generalize\_Query}(q, q', f, f') \wedge \mathbf{EmptyAnswer}(q')$

  $\rightarrow \mathbf{SubstitutedFolder}(f, f')$

- For $(q, \mathbf{FrameTm}(ft'), \mathbf{FrameTmQy}(q, ft), q')$

  $\mathbf{Prior\_to\_All}(ft, ft') \wedge \mathbf{Generalize\_Query}(q, q', ft, ft') \wedge \mathbf{EmptyAnswer}(q')$

  $\rightarrow \mathbf{SubstitutedFrameTm}(ft, ft')$

CHAPTER 10
CONCLUDING REMARKS

In this dissertation, we give a full description of an office document retrieval system with the capabilities of processing incomplete and vague queries and providing meaningful responses to the users when empty answers arise. It has four major components, namely, the system catalog, query transformation, browser and generalizer.

An *unified* system catalog is proposed for storing meta-data and domain knowledge of the document filing organization, and a thesaurus at both the system and operational levels. These provides a centralized retrieval facility for processing complete, incomplete and vague queries and retrieving the meaningful information (pertaining to the users) about the entities of the database.

Upon receiving it, a complete query is transformed into a set of algebraic queries with complete and precise information regarding to the folders (where the documents reposited) and frame templates (the document types) from which the frame instances (i.e., the synopses of documents) are to be retrieved or synthesized. The query processor executes the set of algebraic queries after its formulation.

For any incomplete or vague queries, the browser provides a mechanism for *guiding systematically* the user to gain sufficient knowledge about the entities stored in the database, by representing dynamically the snapshots of the dual model and data elements of the document filing organization in terms of object networks. Such information is obtained by looking up the system catalog. Thus, this allows the user to construct a complete query from his own request.

In attempt to provide the user with meaningful and cooperative responses as interpretations to any given failed query (i.e., with an empty answer), the generalizer is employed to formulate the generalizations of the given failed query, which are derived by *methodically* analyzing the results of executing generalizations and by

strategically and *efficiently* applying the possible folder and frame template substitutions and weakening the search criteria.

10.1 Summary

In the following subsections, we shall summarize the significant features of the system catalog, the query transformation and browser, and the query generalization mechanism.

10.1.1 System Catalog

In TEXPROS, the system catalog is shared by different components of the system. It is desired to use an uniform representation, such as frames, for describing the meta-data and domain knowledge, and the contents of documents. This unified approach allows to use the same methods for retrieving and managing of the knowledge at system and operational levels and eliminates problems of duplicate knowledge and translation between different knowledge representations. The system catalog has the following features:

- The uniform representation of the system catalog and database itself provides a natural and consistent operational approach.
- It includes not only the meta-data knowledge, but also the domain knowledge to increase the effectiveness of the document retrieval system.
- It supports not only the procedure of query processing as a traditional system catalog does, but also the query transformation, browser and generalizer mechanisms.
  - It provides significant support for refining the incomplete queries and formulating the complete queries.

- It supports for deriving dynamically the object network pertaining to a vague query, helps the browser recognizing *synonyms*, and supports *access by value.*

- It provides the similarities, semantic and structural interdependencies, and other meta-data knowledge (i.e., the document type hierarchy and folder organization) to be used by the folder and frame template substitutions, during the process of generalizing any failed queries for achieving cooperative responses.

10.1.2 Query Transformation and Browser

When the user has the knowledge of the database, he can specify his request in a formal query. However, it is difficult for the user to utilize such knowledge to formulate precise and complete queries. The retrieval system, as a Search Computerized Intermediary System [72], is designed in such a way that it allows a user to issue an incomplete query and can help him formulate a complete one. The system has the following features:

- The user can specify only part of the index terms he knows, and the context construction mechanism can find the other missing index terms.

- The user can specify the subject of an index term, and then the context construction mechanism can find all possible relevant index terms.

- The context construction mechanism can find the precise index terms as the correct substitutes for the imprecise terms in the user query.

- The ambiguity of interpreting the query is reduced by having the user to specify as much information as he knows.

- When the multiple index terms are found, the system tries to approach the user for clarification, which usually is a simple and inexpensive way to avoid presenting any irrelevant outcome to the user.

Browsing is used to be a complementary method when the systematic retrieval[1] is difficult or impossible to apply. When a vague query is issued as a topic, the system presents the user an object network, which creates an intuitive environment for browsing, such that an incremental enhancement of user knowledge can be achieved.

- The object network, which is composed of the schema elements and data elements, is depicted as a two dimensional network. In the vertical level, the relationships between the objects of different types (i.e., between the folders and frame templates, the frame templates and attributes, the attributes and values) are described; in the horizontal level, the relationships among the objects of same type (i.e., the folders or frame templates) are presented.

- The topics connected by operator AND and OR comprise quite a simple query interface. However, the very rich functionalities to achieve the user's browsing target are provided. The user does not have to follow the limited guiding facility to perform retrieval tasks, and therefore he has more flexible access to the database.

- The object network is presented to the user at any instant during the browsing session. The instantaneous feedback of the resultant object network and descriptions provides the user with a clear view for analyzing its information and then leading into the further browsing directions. Therefore, the object network providing with needed information gives the user substantial help for constructing a formal query.

[1]The user presents the request in a formal query, and the system retrieves the data promptly [63].

- The browsing process is a "long-sighted" navigation, since it is possible to reach not only the objects adjacent to the current one, but the distantly related objects without navigating through all intermediate objects. The user can select any object from the object network or outside the network as a next browsing topic. The system attempts to find the possible connections of topics or the object networks.

The browsing can be interleaved with formal querying. The combination of the browser with the formal query results in a very effective retrieval environment.

10.1.3 Query Generalization Mechanism

In TEXPROS, since the query entered by the user is less restrictive, the response given to the user by the system may be less cooperative. Our retrieval system is designed to accomplish the requirements, such as the one described in [45], for achieving cooperative responses in the situation when empty answers arise.

- In order to detect the erroneous presuppositions, the system evaluates the results of the subqueries (the generalizations of a given failed query) which are formed using the *Conjunctive Query Graph.* And a set of rules is applied to reduce the space of generalized subqueries by excluding the redundant and irrelevant subqueries. Therefore, only a small subset of query generalizations, based on a constant propagation strategy, is taken into consideration in the generalization procedure.

- To generate precise and meaningful responses for a given failed query, the generalizer is developed by incorporating both the folder substitution and type substitution.

- The *similarity* between folders in the logical file organization based on their semantics is defined. The *semantic and structural interdependency* are introduced to stress the semantic meaning of the relative similarity.
- The various strategies, which are defined in first order logic, are explored for accomplishing substitutions at different context such that the similarity comparison is *context-sensitive*[1]. Therefore, the resultant queries, generated by the application of various substitution strategies to the original query, are more relevant and meaningful. [2]

10.2 Potential Research Directions

In this section, we will discuss several important issues left to be resolved that emanate from the work described in this dissertation.

10.2.1 Knowledge Representation

- With integration of the knowledge representation of retrieval system and other subsystems, such as, document classification, filing, etc., create a centralized document classification, extraction, filing and retrieval environment to achieve an intelligent information system. [77]
- Investigate the automatic processes of generating the frameworks for the various subsystems, from the system catalog, to support the document classification, filing and retrieval in the entire system.[14]
- For the sake of effectiveness and efficiency, the overall structure of system catalog may change in a variety of ways. One likely enhancement will be to add a "server" which maintains the system frame instances in the system

[2] A query is a kind of specification of a context. Disregarding to the specific query, the substitution based only on the logical folder organization and document type hierarchy would lead to irrelevant and meaningless outcome.

catalog and allows the subsystems to access only the portion of system catalog under its authorization. Therefore, based on a client-server architecture, the system can support three basic activities on documents classification, filing and retrieval [9, 11, 12].

10.2.2 Intelligent Database Assistant System

In TEXPROS, each frame template, which describes the properties (or attributes) for a class of documents, is divided into structured and unstructured parts. The contents of an unstructured part can be free text, as opposed to that the attribute values of the structured part are fixed length character strings. By keeping the synopses for both textual and nontextual parts of a document in a frame instance, a user may describe the document in a very succinct manner, without capturing all the information from it. Retrieving and browsing such a small piece of information require much less time than retrieving the original document. However, the information contained in frame instances governs the scope of querying. In performing concept-based and keyword-based retrievals or access by value querying, it is necessary for the system to guide or assist the user to refine gradually his queries [107].

Considerable research has been discussed in the area of free text retrieval [18, 55, 56]. In our system, extending the browser mechanism to the unstructured part of the frame instance can be developed as follows:

- Creating the links between the unstructured fields and the subjects.

  Using WITH clause of the query interface as shown in Figure 3.2, a user can specify a subject for determining its related index terms. In system catalog, we specify the *subjects* which related to the index terms, including folders and frame templates. We can identify the unstructured fields according to the subjects.

- Constructing a subject network.

  A subject network is a graph whose vertices correspond to *subjects*, and edges correspond to relationships between those subjects.

- Augmenting the subject network onto the object network.

  The subject network can be incorporated into the existing object network by connecting the subjects to the index terms.

- Browsing through the connections.

  The connections between a unstructured field and its related frame instances can be discovered dynamically by traversing the paths.

10.2.3 An Information Sharing Environment

When using TEXPROS in a multi-user or distributed environment [6, 17, 59, 97], it requires to share information contained in frame instances. When data communication and sharing are necessary, the system must provide mechanisms for users to specify protocols for extracting, transmitting and exchanging information. Basically, there are two approaches of storing documents, namely, the centralized and distributed ones. For the distributed one, each user has his own document type hierarchy and document filing organization created at his disposal in his own personal TEXPROS. The other approach is to create a centralized database consisting of a unified document type hierarchy and a document filing organization sharing by a group of users, who have limited functional capabilities of adding (and deleting) folders and frame instances into (from) the document filing organization, and of extracting information from documents. Then, one must specify the protocols for governing cooperatively the frame templates definitions, and the document classification and categorization.

For both cases, the system catalog, as a group communication and coordination system, must reflect the contents, extracting from documents by a user, in such a way

that the other users are able to retrieve these documents by specifying formal queries, or to browse through any information that are not created by themselves. For the distributed (centralized) case, the system catalog must be extended to one which has capability of unifying (providing) multiple versions of document type hierarchy and document filing organization from (to) each individual system. The query facility for multiple databases includes the following features:

- An uniform interface is created using an uniform representation of the schema descriptions and the query specification for retrieving data from the multiple databases.

- For global applications, the browing mechanism can be extended to apply on multiple versions of the document type hierarchy and filing organization. The browser may unify the different models visually for a standard presentation, such as, the object network.

- In distributed environment, the coexistence of different document type hierarchies and filing organizations is allowed. Therefore, the system needs to assist users in identifying semantically equivalent data elements and reduce the user's effort of creating a query.

- Coexistence of the different models preserves the autonomy of individual database, and thus, all the existing functions for local applications would not be changed.

10.3 Ongoing Research Topics

Finally, we will briefly describe a number of significant ongoing research in the area of document classification, categorization, management, and many others, which are closely related to the document retrieval system. It is desirable to bring them together to form a complete, workable system.

10.3.1 Document Classification

we classify documents that are similar in properties into a document class. Each class is associated with a type (called a frame template) which describes the properties for the class of documents. The document type hierarchy exploits structural commonalities between frame templates, which are related by specialization and generalization [60, 61, 107, 106]. In general, the type or class to which the document belongs can be identified automatically by analyzing the contents, the layout structure or the conceptual structure of any document [10, 34, 35, 52, 108]. The document classification has laid a solid foundation for the information extraction from documents. In TEXPROS, a knowledge-based document classification subsystem is investigated for classifying documents based upon the layout structure with brief information extracted from the content of a document [34, 35, 108]. The subsystem employs the knowledge acquisition tool to generate the document format trees (each of which describes the layout structure and the content of a document) for each type of documents. This allows to identify the type of a document by matching its layout structure with simple content description against a small set of document format trees.

10.3.2 Document Categorization

A frame instance represents the synopsis of a document. TEXPROS provides facilities to define folders which are repositories of frame instances. And folders are connected to another via the depends-on relationship, thus forming a folder organization. Such an organization mimics the user's real-world document filing system. Given a frame instance, TEXPROS needs to identify a folder and place it in that folder. This procedure is called document categorization. Similarly, in reorganizing files, the system needs to place all the involving frame instances in appropriate folders. To automate these operations, we adopt an agent-based archi-

tecture to implement TEXPROS's categorization subsystem [104, 105, 107]. The criteria used to categorize documents are defined in terms of attribute values and rules. Each filing agent (or folders) is associated with a criterion (a predicate), data structures and operations for handling the frame instances. By comparing the contents of a frame instance against the criterion, the agent is able to distribute the instance into its descendent folders. If the frame instance satisfies categorization rules (i.e., a categorization rule is a well-formed formula consisting of criteria) for many descendent folders, copies are made and sent to each of these folders. By doing so repetitively, the frame instance will be placed in appropriate folders. Given an agent-based architecture of a folder organization, any newly created filing agent (i.e., a folder) for the organization requires to specify its associated criterion. This criterion must be "well-defined" to ensure that every frame instance to be inserted in this folder is distributed and placed exactly in it according to the categorization rules.

The file reorganization, which may occur frequently, may render frame instances accumulated in buffers due to poor categorization criteria. It may also cause duplicate frame instances to be placed in the same folder. Given a collection of folders with their criteria of an existing agent-based architecture, the file reorganization must ensure that the desired categorization rules for the newly-formed architecture are "well-defined" (that is, all frame instances are redistributed and placed in appropriate folders based on the new rules) [117].

10.3.3 Document Management through Hypertext

The concept of hypertext concerns information management and access. Research work is conducted which focuses on integrating hypertext functionalities into TEXPROS for developing a direct manipulation interface that provides access

to all the implicit relationships among documents and the information they contain [103].

Among many others, a visual programming environment, DocFlow VPE, is also investigated for the purpose of specifying and automating structured office procedures including the handling of office documents [15]. The DocFlow VPE provides a programming interface that allows end-users doing their own programming in the office environment.

APPENDIX A

THE STRUCTURE OF SYSTEM CATALOG

A.1 Thesaurus

In TEXPROS(an acronym for Text Processing System, which is an integrated system for processing office documents), an approach to assist in the efficient information retrieval is to provide the system with the knowledge of synonyms. This is usually accomplished by using a thesaurus. In the system catalog, there are three major types of components, SYSSYNONYMS(a component containing synonymous keyterms), SYSNARROWER(a component describing the terms that have semantic associations with the keyterms), and SYSTERMASSOC(a component describing the associations of keyterms in terms of the names of folders, frame templates and attributes) to form the thesaurus as shown in Figure A.1.

- The set of system frame instances in $\mathcal{SYSCATALOG}$(SYSSYNONYMS), whose type is specified by the system frame template SYSSYNONYMS, contains information about synonymous terms that are relevant to the user. The **KeyTerm** contains a system reserved keyterm, which is synonymous to the set of terms that are denoted by **SynKeyTerms** which may exist in the user's queries.
  Let $sfi = \{< KeyTerm, KT >, < SynKeyTerms, \{SKT_1, SKT_2, \ldots, SKT_k\} >\}$ be a system frame instance. Then $sfi \in$SYSCATALOG(SYSSYNONYMS) iff SKT_i is a synonym of $KT, 1 \leq i \leq k$.
  For example, Peter A.Ng can be referred to by one of many different terms such as Peter Ng, Ng, Peter A. Ng and P.A.Ng as shown in Figure A.1.

- The set of system frame instances in $\mathcal{SYSCATALOG}$(SYSNARROWER), whose type is specified by the system frame template SYSNARROWER, contains a set of narrower key terms, NKT_i $(1 \leq i \leq n)$ in a user's query that are semantically associated with a system reserved keyterm, KT. Let

The corresponding frame instance for SYSSYNONYMS

KeyTerm	*Peter A. Ng*
SynKeyTerms	*Peter Ng, Ng, Peter A.Ng, P.A.Ng*

The corresponding frame instance for SYSNARROWER

KeyTerm	*Student Assistant*
NarrKeyterms	*Teaching Assistant, Graduate Assistant, Research Assistant, Student Assistant*

The corresponding frame instances for SYSTERMASSOC

KeyTerm	*Student Assistant*
IndexTm	*Assistants*
IndexTmType	*folder*

KeyTerm	*Q.E.Application*
IndexTm	*Q.E.Application Form*
IndexTmType	*frame template*

Figure A.1 Examples in a Thesaurus

$sfi = \{< KeyTerm, KT >, < NarrKeyTerms, \{NKT_1, NKT_2, \ldots, NKT_k\} >\}$ be a system frame instance. Then $sfi \in$SYSCATALOG(SYSNARROWER) iff NKT_i is a narrower term of $KT, 1 \leq i \leq k$. To a certain extent, NKT_i is a specialization of the KT.

For example, in Figure A.1, Teaching Assistant, Graduate Assistant and Research Assistant are referred to as Student Assistant.

- The frame template SYSTERMASSOC provides a mechanism for associating each keyterm that may appear in a user's query to an index term that is actually residing in the database. The associated index term is classified by an index term type, **IndexTmType**, which may be a folder name, a frame template name or a attribute name. Therefore, the frame instances of the type SYSTERMASSOC specify index terms to be the names of folders, frame templates or attribute names which are associated with the keyterms. Let *sfi* be a system frame instance over SYSTERMASSOC. If *sfi*[**IndexTm**] is the name of a folder, then *sfi*[**IndexTmType**] = '*folder*'. If *sfi*[**IndexTm**] is the name of a frame template, then *sfi*[**IndexTmType**] = '*frametemplate*'. If *sfi*[**IndexTm**] is the name of an attribute, then *sfi*[**IndexTmType**] = '*attribute*'.

  In the example of the system frame instances for SYSTERMASSOC shown in Figure A.1, Q.E.Application Form and Assistants are index terms residing in the database, which represent a frame template name and folder name, respectively.

A.2 Meta-Data

The last five components, SYSFOLDERS (a component for describing the folder characteristics in a logical file structure), SYSFRINSTCOUNT (a component for counting the number of frame instances associated with the frame templates in each folder), SYSFRTEMPLATES (a component for describing the schemas of

frame templates), SYSATTRIBUTES and SYSATTRTYPES (components for defining attributes used in the frame templates) are meta-data, which describe the organizational description of the database. Detailed descriptions of each of these components are given as follows:

- The frame template SYSFOLDERS provides a mechanism to describe not only the frame templates associated with each folder but also the logical file structure. The latter information is represented by the **Depends_On** and **Parent_Of** attributes.

  For example, in Figure A.2, *Ph.D. Students* folder may contain frame instances of the types specified by the frame templates, *Admission_Acc_Letter*, *Updated_Transcript*, etc. This folder depends on another folder named *Ph.D. Program*. This folder has two subordinate folders, and therefore, it is the parent of two folders *Q.E.* and *Publication*. The frame templates represented by the **FTNames** are the local frame templates in the the folder **FolderName** for the purpose of filing reorganization. All the frame templates associated with the folder **FolderName** include not only these local frame templates but also all the frame templates in the descendant folders of **FolderName**.

- The frame template SYSFRINSTCOUNT specifies the number of frame instances whose type is **FTName** in the folder **FolderName**.

  For example, in Figure A.3, there are 20 frame instances of the *Q.E.Result* type and 22 frame instances of the *Q.E.Application* type in the folder *Q.E.*.

- The frame template SYSFRTEMPLATES specifies the attributes within a frame template. The **Is_A** attribute describes the document type hierarchy.

  For example, in Figure A.2, the schema of a frame template, *Q.E.Result* contains the attributes, *Sender*, *Receiver*, *Date*, *Student_Name*, *Date_Taken*

The corresponding frame instances for SYSFOLDERS

FolderName	*Q.E.*
FTNames	*Q.E.Application Form, Q.E.Result, Q.E.Question*
Depends_On	*Ph.D Students*
Parent_Of	*NIL*

FolderName	*Ph.D Students*
FTNames	*Admission_Acc_Letter, Updated_Transcript*
Depends_On	*Ph.D Program*
Parent_Of	*Q.E., Publication*

The corresponding frame instances for SYSFRTEMPLATES

FTName	*Q.E.Result*
AttrNames	*Sender, Receiver, Date, Student_Name, Date_Taken, Outcome*
Is_A	*Memo*

FTName	*Q.E.Application Form*
AttrNames	*Student_Name, Date_Taken, Courses*
Is_A	*Exam Application Form*

Figure A.2 Examples of Meta-data

and *Outcome*. In the document type hierarchy, the *Q.E.Result* is a subtype of *Memo* type.

- The frame template SYSATTRIBUTES is used to describe the information about each attribute in the system. Each attribute, denoted by **AttrName** is associated with an attribute type denoted by **AttrType** in the frame template **FTName**, and is bounded to a set of values, called **ActiveDomain**. The attributes with the same name may have different attribute types in different frame templates.

- The frame template SYSATTRTYPES is to describe the information about each attribute type denoted by **AttrType**, its degree denoted by **Degree**, and its domain denoted by Domain.

Figure A.2, Figure A.3 and Figure A.4 are examples of the frame instances for these five components.

A.3 Attributes Corresponding to the System Catalog

Table A.1 lists the finite set of attributes corresponding to the system catalog.

The corresponding frame instance for SYSFRINSTCOUNT

FTName	*Q.E.Result*
FolderName	*Q.E*
Count	*20*

FTName	*Q.E.Application*
FolderName	*Q.E.*
Count	*22*

Figure A.3 Examples of Meta-data(continued)

The corresponding frame instances for SYSATTRIBUTES

AttrName	*Receiver*
FTName	*Q.E.Result*
Attrtype	*Name*
ActiveDomain	*Fortune, Liu*

AttrName	*Date_Taken*
FTName	*Q.E.Result*
AttrType	*Date*
ActiveDomain	*May 5 1992, May 26 1992, June 13 1992*

The corresponding frame instances for SYSATTRTYPES

AttrType	*Name*
Degree	*3*
Domain	*dom(FName) X dom(LName) X dom(MName)*

AttrType	*Date*
Degree	*3*
Domain	*dom(Month) X dom(Day) X dom(Year)*

Figure A.4 Examples of Meta-data(continued)

Table A.1 Attributes Corresponding to the System Catalog

Attribute A	*dom*(A)	Description
AttrName	*SetOfCharString*	the name of an attribute belonging to some frame template
AttrType	*SetOfCharString*	the name of an attribute type
Domain	*SetOfCharString* × *Integer*	a total function which associates a domain to each attribute
ActiveDomain	*SetOfCharString* × *Integer*	the set of values an attribute has in the DB
FolderName	*SetOfCharString*	the name of a folder in the filing organization
FTName	*SetOfCharString*	the name of a frame template that exists in the document type hierarchy
FTNames	*dom*(FTName)	the name of a frame templates associated with a folder
Depends_On	*SetOfCharString*	a set of predecessor folder names
Parent_Of	*SetOfCharString*	a set of successor folder names
Is_A	*SetOfCharString*	a set of frame template names in superclass
Degree	*Integer*	the number of component attribute types comprising some attribute type T
KeyTerm	*SetOfCharString*	a term that may appear in a user's query or associated with a term in user's query
IndexTm	*SetOfCharString*	a term that exists in the database
IndexTmType	*folder, frame template*	the type of IndexTm
SynKeyTerms	*SetOfCharString*	a set of keyterms that appear in a user's query and are synonymous to KeyTerm
NarrKeyTerms	*SetOfCharString*	a set of keyterms that appear in a user's query and are semantically associated with KeyTerm

APPENDIX B

RETRIEVAL ON SYSTEM CATALOG

Recall that the system catalog is considered to be a folder of several frame templates. Each of these frame templates is a representative of a subset of system frame instances of the system catalog. In this chapter, we restrict the following discussion to the system catalog. We investigate the use of **algebra** to query the system catalog, and we present the methods of retrieval on the system catalog using algebraic query language.

B.1 Retrieval on $\mathcal{SYSCATALOG}$(SYSSYNONYMS)

The $\mathcal{SYSCATALOG}$(SYSSYNONYMS) component allows the user to use different synonyms for a standardized keyterm. For example, in the system, Peter A. Ng is a standardized keyterm to refer to a person. The SYSSYNOMYMS allows the user to use different terms, such as Peter Ng, P.A.Ng, etc. to refer to the same person and TA's or TA to refer to a teaching assistant. Such standardized keyterms can be obtained through the application of algebraic operators, such as projection(π), selection(σ) and unnest(μ). For example, a query can be given as follows:

Get the *keyterm* whose synonymous set includes x (Equivalently, get the *keyterm* for x from SYSSYNONYMS). Its equivalent algebraic query is as follows. y is the *keyterm* yielded from a given synonymous keyterm x.

$f1 = \sigma_{SynKeyTerms \supseteq x}(\mathcal{SYSCATALOG}(\text{SYSSYNONYMS}))$, which is equivalent to

$f1 = \sigma_{SynKeyTerms = x}(\mu_{SynKeyTerms}(\mathcal{SYSCATALOG}(\text{SYSSYNONYMS})))$;

if $f1 \neq empty$ then

$y = sfi[KeyTerm]$ where $sfi \in f1$;

B.2 Retrieval on $\mathcal{SYSCATALOG}$(SYSNARROWER)

The $\mathcal{SYSCATALOG}$(SYSNARROWER) component provides a mechanism which allows the user to derive a system standardized keyterm by given terms whose semantics are closely related to it. For example, the terms *Teaching Assistant*, *Graduate Assistant* and *Research Assistant* are referred to the keyterm *Student Assistant*. To a certain extent, the student assistant has a broader function than the others and they are semantically related.

An example of a query and its algebraic query is given as follows.

Get the *KeyTerm* whose narrow term set includes x.

$f1 = \sigma_{NarrKeyTerms \supseteq x}(\mathcal{SYSCATALOG}(\text{SYSNARROWER}))$;

if $f1 \neq empty$ then

$y = sfi[KeyTerm]$ where $sfi \in f1$;

B.3 Retrieval on $\mathcal{SYSCATALOG}$(SYSTERMASSOC)

In an application, the system standardized keyterms can refer to the names of folders in which the frame instances of documents are located, or to the names of frame templates from which the frame instances of documents are created in the filing organization. In the process of retrieving frame instances of documents, the retrieval process can be eased by providing the information about the folder which contains a frame instance to be retrieved, or the frame template corresponding to the type of the frame instance to be retrieved. However the exact names of the folder and frame template may not necessarily be quoted by the user. The $\mathcal{SYSCATALOG}$(SYSTERMASSOC) provides a capability for the system to identify the exact name of a folder and the exact name of a frame template, if a standardized keyterm is used.

In the following, examples of queries and their algebraic queries are given.

- Get the index term z and its type zt , which is associated with given *keyterm* y.

  $f1 = \sigma_{KeyTerm=y}(\mathcal{SYSCATALOG}(\text{SYSTERMASSOC}))$;

  if $f1 \neq empty$ then

  $(z, zt) = \{sfi[IndexTm], sfi[IndexTmType] | sfi \in f1\}$;

- Get the folder z which is associated with *Keyterm* y.

  $f1 = \sigma_{KeyTerm=y \wedge IndexTmType=folder}(\mathcal{SYSCATALOG}(\text{SYSTERMASSOC}))$;

  if $f1 \neq empty$ then

  $y = sfi[IndexTm]$ where $sfi \in f1$;

- Get the frame template z which is associated with *Keyterm* y.

  $f1 = \sigma_{KeyTerm=y \wedge IndexTmType=frametemplate}(\mathcal{SYSCATALOG}(\text{SYSTERMASSOC}))$;

  if $f1 \neq empty$ then

  $y = sfi[IndexTm]$ where $sfi \in f1$;

In addition to the capabilities of describing synonyms of keyterms, the semantic associations of terms and the exact terms used as names of the folders and frame templates, the system catalog also contains five additional components, SYSFOLDERS, SYSFRINSTCOUNT, SYSFRTEMPLATES, SYSATTRIBUTES and SYSATTRTYPES, for describing the document type and logical file structures, the folder characteristics, the schemas of frame templates and the characteristics of the attributes appeared in the frame templates, which give significant support and help to the user during the process of extracting information from documents, and storing and retrieving frame instances of documents.

B.4 Retrieval on $\mathcal{SYSCATALOG}$(SYSFOLDERS)

The $\mathcal{SYSCATALOG}$(SYSFOLDERS) contains frame instances, each of which describes a folder in terms of its name, ancestor(s) and descendant(s), and the types of synopses of documents contained in the folder. This provides the user with the capabilities of finding the number of folders being checked for determining whether a folder is in the system(TEXPROS), the types of frame instances contained in a folder, the folders which are its predecessor(s) (Depend_On) and successor(s) (Parent_Of), and all the folders that are associated with a given frame templates.

Following are examples of queries and their algebraic queries.

- Given ef, the number of folders, which are checked for determining whether the folder z is in the system.

  $ef = \text{count}_{FolderName}(\sigma_{FolderName=z}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS})))$;

- Get all the children folders of z.

  $f1 = \sigma_{FolderName=z}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;

  if $f1 \neq empty$ then

  $fdc = \{sfi[Parent\_Of]|sfi \in f1\}$;

- Get all the frame templates *fts* associated with folder z.

  GetFt(z)

  Begin

  $f1 = \sigma_{FolderName=z}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;

  $fts = \{sfi[FTNames]|sfi \in f1\}$;

  $fcs = \{sfi[Parent\_Of]|sfi \in f1\}$;

  if $fcs \neq empty$ **then**

  For each $fc \in fcs$ **Do**

  $fts = fts \cup$ **GetFt**(fc);

return(fts)

end

- Get all the parents folders of z.

  $f1 = \sigma_{FolderName=z}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;

  if $f1 \neq empty$ **then**

  $fdp = \{sfi[Depends\_On] | sfi \in f1\}$;

- Get all the folders *FolderNames* associated with frame template ft.

  GetFolder(ft)

  Begin

  $f1 = \sigma_{FTNames \supseteq ft}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;

  $fds = \{sfi[FolderName] | sfi \in f1\}$;

  For each $fd \in fds$ **Do**

  $FolderNames = fds \cup$ **GetPredecessor**(fd);

  $f2 = \sigma_{Is\_A \supseteq ft}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES}))$;

  if $f2 \neq empty$ **then**

  Begin

  $fts = \{sfi[FTName] | sfi \in f2\}$;

  For each $ft \in fts$ **Do**

  $FolderNames = FolderNames \cup$ **GetFolder**(ft);

  end;

  return(FolderNames)

  end

GetPredecessor(fd)

Begin

$f1 = \sigma_{FolderName=fd}(\mathcal{SYSCATALOG}(\text{SYSFOLDERS}))$;

$fps = \{sfi[Depends\_On] | sfi \in f1\}$;

if $fps \neq empty$ **then**

$fd = fd \cup$ **GetPredecessor**(fps);

return(fd)

end

B.5 Retrieval on $\mathcal{SYSCATALOG}$(SYSFRTEMPLATES)

During the process of extracting information from documents and retrieving frame instances of the documents, there needs a frame template z to govern the information extraction and the retrieval based on a query by attributes. Then the existence of such a frame template in the system, the information about its superclasses and its attributes, and the frame templates containing the given attributes can be in question. Given the ($\mathcal{SYSCATALOG}$(SYSFRTEMPLATES)), this information can be obtained as follows.

- Given eft,the number of frame templates, which are checked for determining whether a frame template z is in the system.

  $eft = \text{count}_{FTName}(\sigma_{FTName=z}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES})))$;

- Get all the attributes in the frame template z.

  $f1 = \sigma_{FTName=z}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES}))$;

  if $f1 \neq empty$ then

  $attrs = \{sfi[AttrNames] | sfi \in f1\}$;

- Get frame templates which are the superclass of frame template z.

  $f1 = \sigma_{FTName=z}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES}))$;

  if $f1 \neq empty$ then

  $fts = \{sfi[Is\_A] | sfi \in f1\}$;

- Get all the frame templates which include any subset of attributes att.

  $f1 = \sigma_{AttrNames \supseteq att}(\mathcal{SYSCATALOG}(\text{SYSFRTEMPLATES}))$;

  if $f1 \neq empty$ then

  $fts = \{sfi[FTName] | sfi \in f1\}$;

B.6 Retrieval on $\mathcal{SYSCATALOG}$(SYSATTRIBUTES)

The $\mathcal{SYSCATALOG}$(SYSATTRIBUTES) and $\mathcal{SYSCATALOG}$(SYSATTRTYPES) provide the user with a detailed description of the attributes of the frame templates and the capabilities to manipulate the attributes.

Following are examples of queries and their algebraic queries.

- Given ac, the number of attributes, which are checked for determining whether the attribute att is in the system.

  $ac = \text{count}_{AttrName}(\sigma_{AttrName=att}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES})))$;

- Get all the frame templates which include the attribute att of type $attype$.

  $f1 = \sigma_{AttrName=att \wedge AttrType=attype}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES})))$;

  $ft = \{sfi[FTName] | sfi \in f1\}$;

- Get all the attributes whose active domain include any subset of v.

  $f1 = \sigma_{ActiveDomain \supseteq v}(\mathcal{SYSCATALOG}(\text{SYSATTRIBUTES}))$;

  if $f1 \neq empty$ then

  $attrs = \{sfi[AttrName] | sfi \in f1\}$;

APPENDIX C

SYSTEM CATALOG MANAGEMENT

In this chapter we describe how the system catalog is managed dynamically during document classification and filing(categorization). We define the functions that manage the system catalog as triggers.

C.1 System Catalog Management during Document Classification

During document classification, if a user selects a frame template which does not exist in the system catalog, the following triggers are invoked:

1. **InsertFrTemplate**(FTName, AttrName, Is_A):

   This function will append a new frame template containing relevant information about name of the frame template, its attribute names, and its Is_A relationship in the document type hierarchy as a system frame instance of $\mathcal{SYSCATALOG}$(SYSFRTEMPLATES).

2. **InsertAttributes**(AttrName, FTName, AttrType, ActiveDomain):

   Information about any attributes of this frame template with their attribute types and active domains that do not exist in the system must be appended as system frame instances of $\mathcal{SYSCATALOG}$(SYSATTRIBUTES).

3. **InsertAttrTypes**(AttrType, Degree, Domain):

   Information about any attribute types that do not exist in the system must be appended as system frame instances of $\mathcal{SYSCATALOG}$(SYSATTRTYPES).

4. **InsertAssocTerms**(KeyTerm, FTName, IndexTmType):

   This function will update the subfolder $\mathcal{SYSCATALOG}$(SYSTERMASSOC). It appends the frame template name, *FTName*, as a value of **IndexTerm** in the frame instance associated with the **KeyTerm** *KeyTerm*.

C.2 System Catalog Management during Document Filing

The primitive functions are defined in section C.2.1. In section C.2.2 various algorithms to update the system catalog using these primitive function are described.

C.2.1 Primitive Functions

The following primitive functions are employed for manipulating system frame instances of SYSFOLDERS type in $\mathcal{SYSCATALOG}$(SYSFOLDERS) during document filing.

1. **InsertFolderName**(*folder*):

   This function will create a system frame instance *sfi* of SYSFOLDERS type in the $\mathcal{SYSCATALOG}$(SYSFOLDERS), in which *sfi*[**FolderName**] is the name of a folder *folder*, and the values for the other attributes are *NIL*.

2. **DeleteFolderName**(*folder*):

   This function will remove a system frame instance *sfi* of SYSFOLDERS type from the $\mathcal{SYSCATALOG}$(SYSFOLDERS), in which *sfi*[**FolderName**] is *folder*.

3. **InsertFTName**(*folder, frametemplate*):

   This function will append *frametemplate* as an element of the *sfi*[**FTNames**] in the system frame instance *sfi* without duplicate, where *sfi* $\in$ SYSFOLDERS, and *sfi*[**FolderName**]= *folder*.

4. **DeleteFTName**(*folder, frametemplate*):

   This function will remove *frametemplate* from the set *sfi*[**FTNames**], where *sfi* $\in$ SYSFOLDERS, and *sfi*[**FolderName**]= *folder*.

5. **CheckFICount**(*frametemplate, folder*):

   This function will check the number of frame instances *sfi*[**Count**], where *sfi* $\in$ SYSFRINSTCOUNT, *sfi*[**FolderName**]= *folder* and *sfi*[**FTName**]= *frametemplate*.

6. **InsertFRINST**(*frametemplate, folder,num*):

   This function will add the value *num* to the *sfi*[**Count**], where *sfi* ∈ SYSFRINSTCOUNT, *sfi*[**FolderName**]= *folder* and *sfi*[**FTName**]= *frametemplate*. If ¬∃*sfi* ∈ SYSFRINSTCOUNT, *sfi*[**FolderName**]= *folder* and *sfi*[**FTName**]= *frametemplate*, then this function will insert a system frame instance *sfi* of type SYSFRINSTCOUNT, in which *sfi*[**FolderName**]= *folder*, *sfi*[**FTName**]= *frametemplate* and *sfi*[**Count**]= *num*.

7. **DeleteFRINST**(*frametemplate, folder, num*):

   This function will subtract the value *num* from the *sfi*[**Count**], where *sfi* ∈ SYSFRINSTCOUNT, *sfi*[**FolderName**]= *folder* and *sfi*[**FTName**]= *frametemplate*. If *sfi*[**Count**]= 0 after subtraction, this function will delete the system frame instance *sfi*.

8. **InsertDepend**(*childfolder, parentfolder*):

   This function will append *parentfolder* as an element of the *sfi*[**Depends_On**] in the system frame instance *sfi* without duplicate, where *sfi* ∈ SYSFOLDERS, *sfi*[**FolderName**]= *childfolder*.

9. **DeleteDepend**(*childfolder, parentfolder*):

   This function will remove *sfi*[**Depends_On**] = *parentfolder* from the set *sfi*[**Depends_On**], where *sfi* ∈ SYSFOLDERS, *sfi*[**FolderName**]= *childfolder*.

10. **InsertParent**(*parentfolder, childfolder*)

    This function will append *childfolder* as an element of the *sfi*[**Parent_Of**] in the system frame instance *sfi* without duplicate, where *sfi* ∈ SYSFOLDERS, *sfi*[**FolderName**]= *parentfolder*.

11. **DeleteParent**(*parentfolder, childfolder*):

    This function will remove *sfi*[**Parent_Of**] = *childfolder* **from the set** *sfi*[**Parent_Of**], **where** *sfi* ∈ SYSFOLDERS, *sfi*[**FolderName**]= *parentfolder*.

C.2.2 Algorithms for Modifying SYSFOLDERS

In TEXPROS, an agent-based approach to automating document filing is employed [104, 105]. Associated with each folder in the filing organization, there is a filing agent which specifies its private data structures (called attributes) and operations (or methods) for manipulating the data structures. The attributes specify the linkages among folders, and the criteria for accepting frame instances reposited in folders at the locations called output and collection. The methods include distributing and collecting frame instances from folders to folders, modifying criteria, and so forth.

Based on these operations at the level of implementation, there are two groups of operations at the user's level for manipulating folders and the frame instances of documents reposited in the folders. For the frame instances, two major operations are the insertion of a frame instance into a folder and the deletion of a frame instance from a folder. In the process of automating document filing, the insertion of frame instances into proper folders can be done by distributing each of the frame instances from a folder into one of its descendants. In dealing with folders, the operations include the insertion of a new folder, the relocation of a folder with its contents, the deletion of a folder with or without its contents and the merge of folders with their contents. This section discusses operations that arise during document filing and which require updating the subfolder $\mathcal{SYSCATALOG}$(SYSFOLDERS).

1. The process of automatically inserting frame instances fi_s into the proper folders in the filing organization requires the distribution of each frame instance *fi* of a document from a folder fd_p into a folder fd_c, a descendant of fd_p, as shown in Figure C.1. This invokes **DetermineFT**(*fi*) to determine the type (a frame

template) *ft* of *fi*, and then **InsertFTName**(fd_c, ft) will be invoked to append the *ft* as a value of the **FTNames** of the frame instance (of SYSFOLDERS type) whose FolderName is fd_c, if *ft* is not a value of the **FTNames**. The function **CheckFICount**(ft, fd_p) is invoked to check the number of frame instances of type *ft* in folder fd_p. The function **DeleteFTName**(fd_p, ft) is invoked to remove *ft* from **FTNames** in the frame instance associated with folder fd_p if no more frame instance of *ft* type are in the folder fd_p. The function **DeleteFRINST**(ft, fd_p, 1) is invoked to reduce the number of frame instances of type *ft* in folder fd_p.

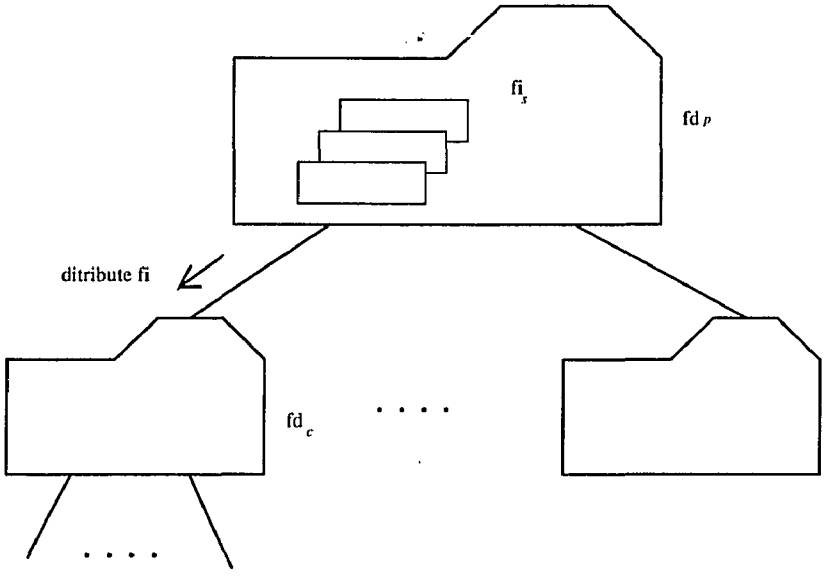

Figure C.1 Distribution of Frame Instances fi_s

In the filing organization, it may be desirable to distribute a set of frame instances fi_s from a folder fd_p into a folder fd_c. Then the sequence of actions activated is as follows:

For each fi in fi_s

Do ft := **DetermineFT**(fi);

InsertFICount(ft, fd_c, 1);

If ft does not appear in the **FTNames** of the frame

instance of SYSFOLDERS type associated with fd_c

then **InsertFTName**(fd_c, ft);

If **CheckFICount**(ft, fd_p)= 1

then **DeleteFTName**(fd_p, ft);

DeleteFICount(ft, fd_p, 1)

end

A special case is that, in the filing organization, it may be desirable to insert a frame instance fi of a document into a folder fd_c, whose predecessor is fd_p. Then, in $\mathcal{SYSCATALOG}$, the sequence of actions activated is as follows:

Do ft := **DetermineFT**(fi);

If ft does not appear in the **FTNames** of the frame

instance of SYSFOLDERS type associated with fd_c

then **InsertFTName**(fd_c, ft);

InsertFICount(ft, fd_p, 1)

end

In another case is that it may be desirable to delete(or remove) a frame instance fi of a document from a folder fd_c, whose predecessor is fd_p. Then, in $\mathcal{SYSCATALOG}$, the sequence of actions activated is as follows:

Do ft := **DetermineFT**(fi);

If **CheckFICount**(ft, fd_c)= 1

then **DeleteFTName**(fd_c, ft);

DeleteFICount(ft, fd_c, 1)

end

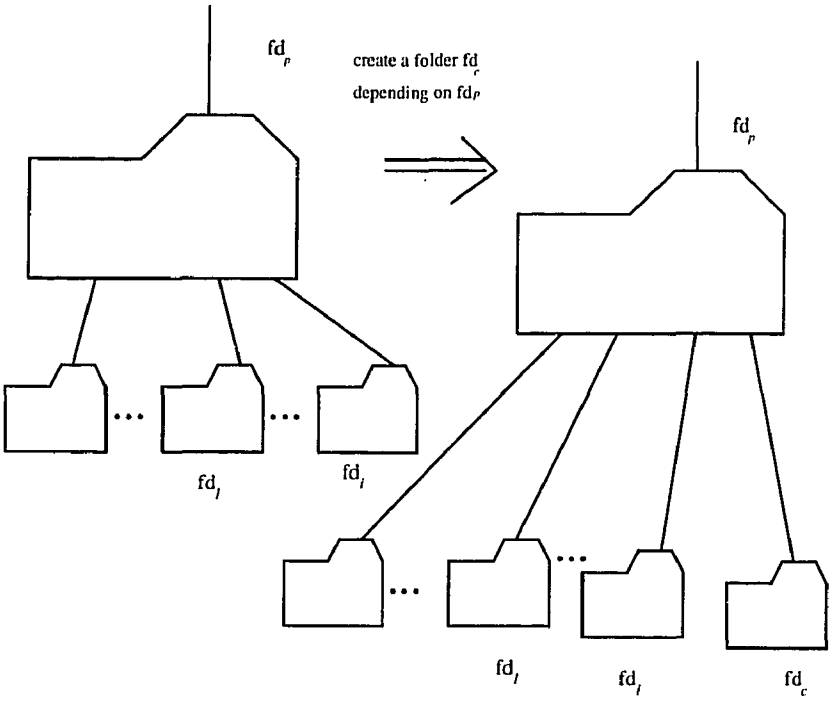

Figure C.2 Insertion of a Folder fd_c

2. In the filing organization of the system(TEXPROS), a folder fd_p may have several descendants, fd_s's. For inserting a new folder fd_c to be a child of a folder fd_p within the filing organization, as shown in Figure C.2, the system will invoke the function **InsertFolderName**(fd_c) for inserting a system frame instance of SYSFOLDERS type, containing fd_c as the **FolderName**, into the $\mathcal{SYSCATALOG}$(SYSFOLDERS), and then **InsertDepend**(fd_c, fd_p) for

inserting fd_p as its **Depend_On**. Finally, the function **InsertParent**(fd_p, fd_c) is invoked to append fd_c as a value of **Parent_Of** in the frame instance associated with the folder whose name is fd_p in the $\mathcal{SYSCATALOG}$ (SYSFOLDERS). Thus, the following actions are applied.

Do

InsertFolderName(fd_c);

InsertDepend(fd_c, fd_p);

InsertParent(fd_p, fd_c)

end

However, it may be desirable to insert a new folder fd_c to be a child of fd_p and to be a parent of fd_i's, as shown in Figure C.3. Then after inserting a new folder fd_c to be a child of fd_p, the following sequence of actions must be taken to change fd_i's as the descendants from fd_p to fd_c.

For each fd_j in fd_i's

Do

DeleteParent(fd_p, fd_j);

InsertParent(fd_c, fd_j);

InsertDepend(fd_j, fd_c);

DeleteDepend(fd_j, fd_p)

end

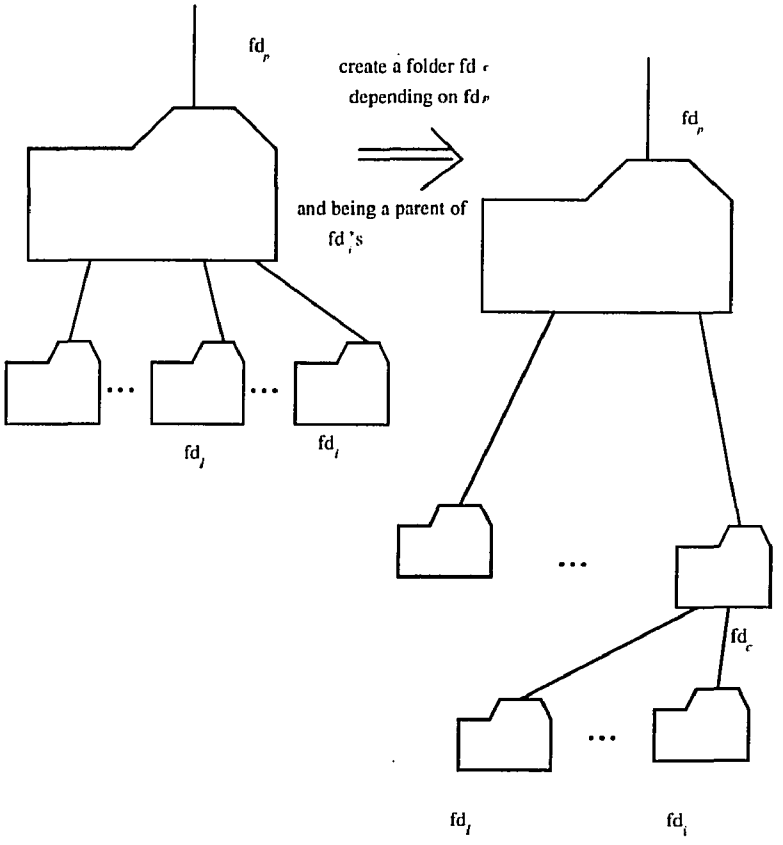

Figure C.3 Insertion of a Folder fd_c

3. Within the filing organization of TEXPROS, it may be desirable to disassociate the folder fd_1 as the predecessor of the folder fd_c and to designate the folder fd_2 as the predecessor of the folder fd_c, which may have several folders as its descendants. To change the predecessor of the folder fd_c with its contents from fd_1 to fd_2, as shown in Figure C.4, the function **DeleteParent**(fd_1, fd_c) is invoked to remove the fd_c from **Parent_Of** associated with fd_1 and **InsertParent**(fd_2, fd_c) to append fd_c as a value of **Parent_Of** associated with fd_2. Then the function **InsertDepend**(fd_c, fd_2) and **DeleteDepend**(fd_c, fd_1) will be invoked for replacing fd_1, one of the values of **Depends_On** associated with fd_c by the new value fd_2 in the $\mathcal{SYSCATALOG}$(SYSFOLDERS).

   In summary, the sequence of actions activated is as follows:

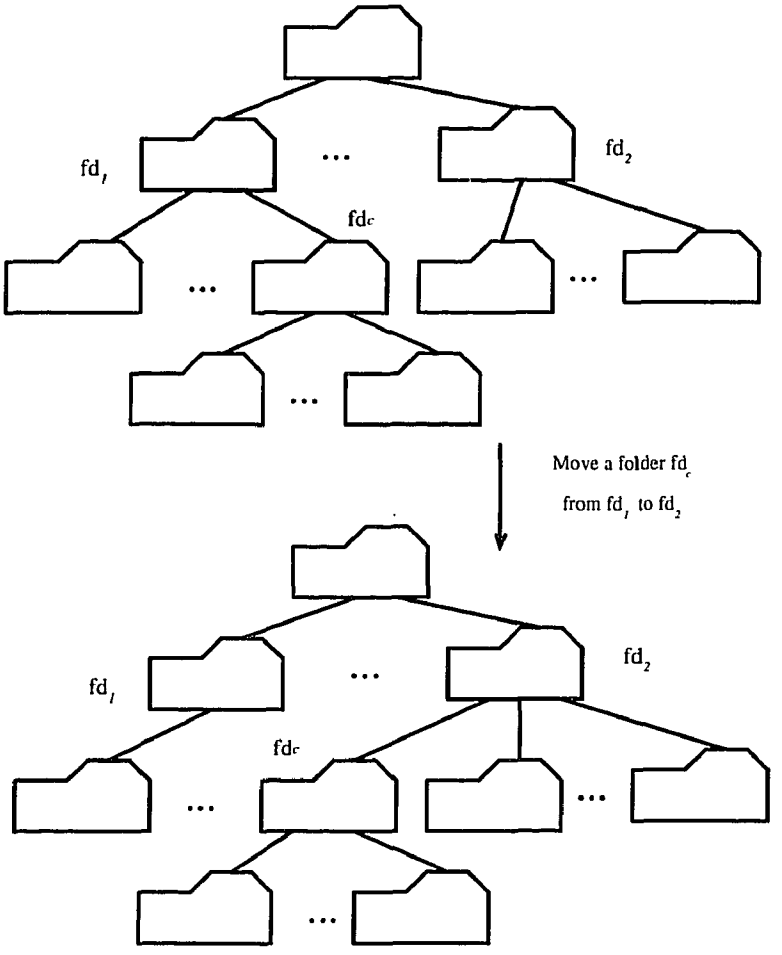

Figure C.4 Relocation of a Folder fd_c

Do

DeleteParent(fd_1, fd_c);

InsertParent(fd_2, fd_c);

InsertDepend(fd_c, fd_2);

DeleteDepend(fd_c, fd_1)

end

4. In filing organization, it may be desirable to collect all the frame instances fi_s from folder fd_c by its parent folder fd_p and then delete the folder fd_c and its descendants, as shown in Figure C.5. All the frame template names from the frame instances in the subtree of folder fd_c are appended as the values of **FTNames** of the folder fd_p by invoking the function **InsertFTNames**(fd_p, ft). Then the function **DeleteParent**(fd_p, fd_c) is invoked to remove the fd_c from the **Parent_Of** associated with fd_p. Finally, the function **DeleteFolderName**(fd) is invoked to remove the relevant information about folder fd_c and its descendants, which are the frame instances in $\mathcal{SYSCATALOG}$ (SYSFOLDERS). In summary, the sequence of actions activated is as follows:

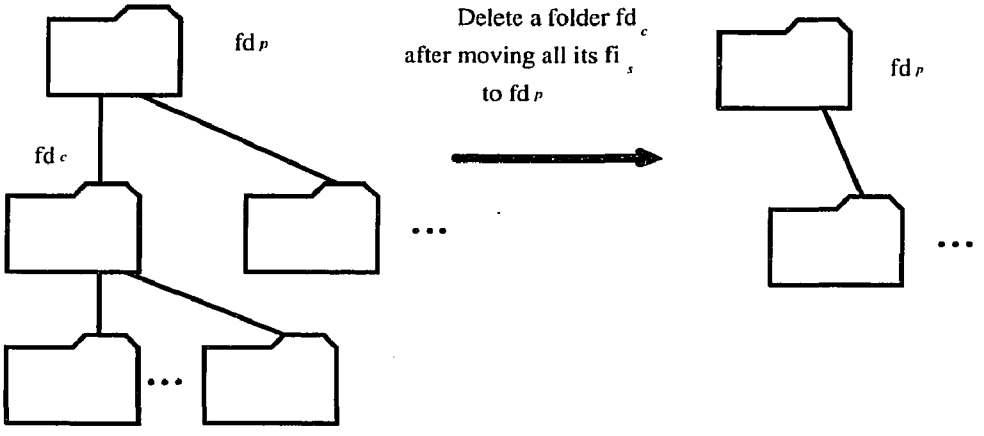

Figure C.5 Deletion of a Folder fd_c

For each *ft* appeared in **FTNames** of the frame instance of SYSFOLDERS type associated with *fd* which is either fd_c or its descendants

Do

InsertFTNames(fd_p, ft);

Number= CheckFICount(ft, fd);

InsertFICount(ft, fd_p, **Number**);

DeleteFICount(ft, fd_c, **Number**);

DeleteParent(fd_p, fd_c);

For each folder *fd* as a value of the **FolderName** of the frame instances of SYSFOLDERS type associated with fd_c and its descendants

Do

DeleteFolderName(fd)

end

In filing organization, it may be desirable to collect all the frame instances fi_s from folder fd_c by its parent folder fd_p without deleting the folder fd_c. After processing **InsertFTNames**(fd_p, ft), the function **DeleteFTName**(fd, ft) is invoked for removing *ft* from **FTNames** in the frame instances associated with a folder *fd* which is either fd_c or its descendants, if no more frame instance of *ft* type is in the folder *fd*. The sequence of actions activated is as follows:

For each *ft* appeared in **FTNames** of the frame instances of
of SYSFOLDERS type associated with *fd* which is either fd_c
or its descendants
Do **InsertFTNames**(fd_p, ft);
Number= CheckFICount(ft, fd);
InsertFICount(ft, fd_p, **Number**);
DeleteFICount(ft, fd, **Number**);
DeleteFTName(fd, ft)
end

5. In filing, it may be desirable to remove a folder fd_c with its contents from the filing organization. The contents include all the frame instances and its descendants. Assume that the folder fd_p is the parent of fd_c. This can be done by using a special operation called **KillFolder**.

   In $\mathcal{SYSCATALOG}$, **DeleteParent**(fd_p, fd_c) is invoked for removing fd_c from the **Parent_Of** associated with fd_p. Then **DeleteFolderName**(fd) is invoked to remove the folder *fd* which is either fd_c or its descendants from the $\mathcal{SYSCATALOG}$(SYSFOLDERS). A special case is that if, in the filing organization, the last frame instance of a document type *ft* has been removed from a folder *fd*, then in SYSFOLDERS, the function **DeleteFTName** is invoked to delete *ft* from the **FTNames** in the frame instance associated with the folder *fd*. The sequence of actions activated is as follows:

For each *ft* appeared in **FTNames** of the frame instances of
of SYSFOLDERS type associated with *fd* which is either fd_c
or its descendants
Do **Number= CheckFICount**(ft, fd);
DeleteFICount(ft, fd, **Number**);
DeleteFolderName(fd);
DeleteParent(fd_p, fd_c)
end

6. Let the folders $fd1_p$ and $fd2_p$ be the predecessors of the folders fd_1 and fd_2 respectively. In the filing process, it may be desirable to merge the folder fd_1 and fd_2, to rename the resultant folder as fd_c, and to move fd_c as a descendant of fd_p, as shown in Figure C.6 and Figure C.7.

   Corresponding to the folder fd_c created in the filing organization, in SYSFOLDERS, **InsertFolderName**(fd_c) is invoked to create a frame instance of SYSFOLDERS type with fd_c as a value of **FolderName**. Then **InsertDepend**(fd_c, fd_p) and **InsertParent**(fd_p, fd_c) are invoked to append fd_p in the **Depend_On** associated with fd_c, and fd_c in the **Parent_Of** associated with fd_p, respectively. The function **InsertFTNames**(fd_c, ft) is invoked repeatedly for inserting all the *ft*'s appearing in the **FTNames** of the frame instances associated with fd_1 and fd_2, into the **FTNames** of the frame instance associated with fd_c. The function **InsertParent**(fd_c, childfolder) is invoked repeatedly for inserting all the childfolders appeared in the **Parent_Of** of the frame instances associated with fd_1 and fd_2, into the **Parent_Of** of the frame instances associated with fd_c. While doing this, **InsertDepend**(childfolder, fd_c) and **DeleteDepend**(childfolder, fd_k) are invoked for replacing fd_1 and fd_2 by fd_c as the value of **Depend_On** in the frame instances of SYSFOLDERS type associated

with all the childfolders of fd_1 and fd_2 by fd_c. Finally, **DeleteParent**($fd1_p$, fd_1) and **DeleteParent**($fd2_p$, fd_2) are invoked to disassociate fd_1 and fd_2 from their parent $fd1_p$ and $fd2_p$. In summary, the sequence of actions activated is as follows:

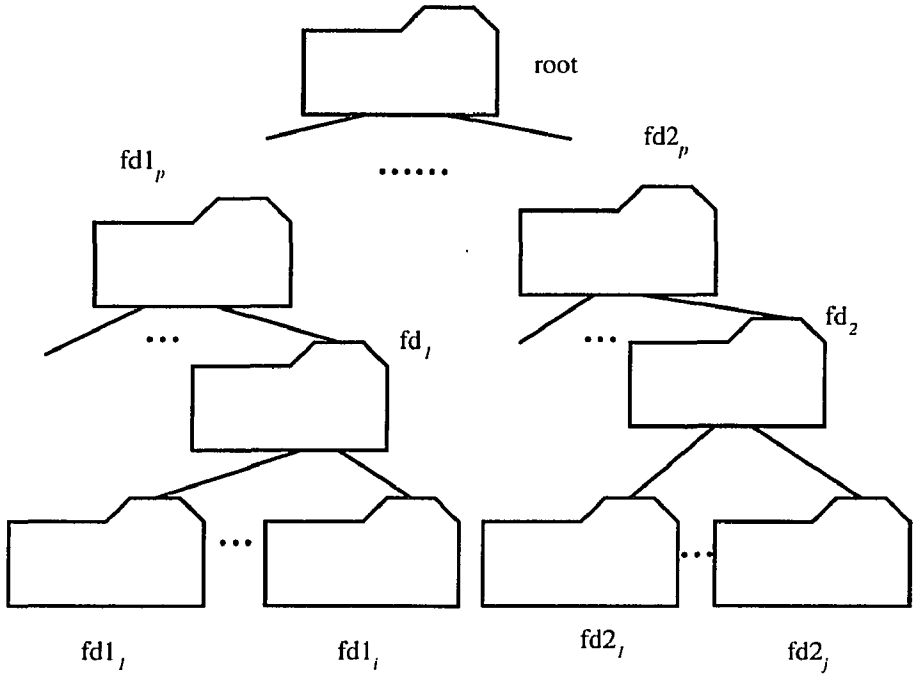

Figure C.6 Before Merging Two Folders fd_1 and fd_2.

InsertFolderName(fd_c);

InsertDepend(fd_c, fd_p);

InsertParent(fd_p, fd_c);

For each folder fd_k, $(1 \leq k \leq n)$

Do

For each ft appearing in **FTNames** of the frame instances of SYSFOLDERS type associated with fd_k

Do

InsertFTNames(fd_c, ft);

For each childfolder of the **Parent_Of** associated with fd_k

Do

InsertParent(fd_c, childfolder);

InsertDepend(childfolder, fd_c);

deleteDepend(childfolder, fd_k);

DeleteParent($fd1_p$, fd_1);

DeleteParent($fd2_p$, fd_2);

DeleteFolderName(fd_1);

DeleteFolderNamefd$_2$);

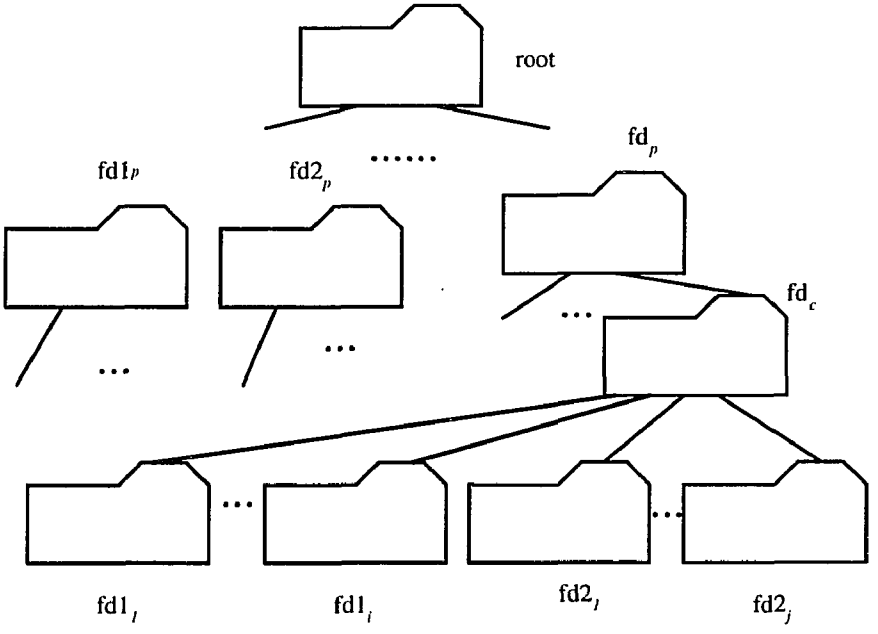

Figure C.7 After Merging Two Folders fd_1 and fd_2.

Note that, in the filing organization, merging folder fd_1 and fd_2, which have the same parent fd_p, and then renaming the resultant folder as fd_c, which is a descendant of fd_p, is to be considered as a special case.

C.2.3 Algorithms for Modifying SYSTERMASSOC

During document filling, the system also needs to update the subfolder $\mathcal{SYSCATALOG}$(SYSTERMASSOC) by invoking the following functions:

- **UpdateAssocTerms**(KeyTerm, OldFolderName, NewFolderName, IndexTmType):
  This function replaces *OldFolderName*, one of the values of **IndexTerm** associated with *KeyTerm* by the *NewFolderName*.

- **InsertAssocTerms**(KeyTerm, FolderName, IndexTmType):
  This function will append *FolderName* as a value of **IndexTerm** in the frame instance associated with KeyTerm *KeyTerm*.

- **DeleteAssocTerms**(KeyTerm, FolderName, IndexTmType):
  This function will remove *FolderName* from **IndexTerm** of the frame instance associated with KeyTerm *KeyTerm*.

REFERENCES

1. T. Anwar and H. Beck, "Knowledge Mining by Imprecise Querying: A Classification-Based Approach," in *Proceedings of the 8th International Conference on Data Engineering*, Tempe, Arizona, pp. 622–630, February 1992.

2. E. Bertino, F. Rabitti, and S. Gibbs, "Query Processing in a Multimedia Document System," *ACM Transactions on Office Information Systems*, vol. 6, no. 1, pp. 1–41, January 1988.

3. P. Bose and M. Rajinikanth, "KARMA: A Knowledge-Based Assistant to a Database System," in *Proceedings of the 2nd. Conference of AI Applications*, pp. 462–472, October 1985.

4. M. Bouzeghoub and E. Metais, "SECSI: An Expert System Approach for Database Design," *Information Processing*, pp. 251–257, 1986.

5. B. Buckles and F. Petry, "A Fuzzy Representation of Data for Relational Databases," *Fuzzy Sets and Systems*, vol. 5, pp. 213–226, 1982.

6. O. Bukhres, J. Chen, A. Elmagarmid, X. Liu, and J. Mullen, "InterBase: A Multidatabase Prototype System," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Washington, D.C., pp. 534–539, June 1993.

7. D. Campbell, D. Embley, and B. Czejdo, "Graphical Query Formulation for an ER Model," *Data and Knowledge Engineering*, vol. 2, pp. 89–121, 1987.

8. R. Cattell, "An Entity-Based Database Interface," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Santa Monica, CA, pp. 144–150, May 1980.

9. A. Celentano, M. Fugini, and S. Pozzi, "Knowledge-Based Retrieval of Office Documents," in *Proceedings of the 13th ACM SIGIR International Conference on Research and Development in Information Retrieval*, Brussels, Belgium, September 1990.

10. A. Celentano, M. Fugini, and S. Pozzi, "Classification and Retrieval of Documents Using Office Organization Knowledge," in *Proceedings ACM Conference on Organizational Computing Systems*, Atlanta, Georgia, pp. 159–164, November 1991.

11. A. Celentano, M. Fugini, and S. Pozzi, "Querying Office Systems about Document Roles," in *Proceedings of the 14th ACM SIGIR International Conference on Research and Development in Information Retrieval*, Chicago, Illinois, pp. 183–189, October 1991.

12. A. Celentano, M. Fugini, and S. Pozzi, "Conceptual Document Browsing and Retrieval in Kabiria," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, San Diego, CA, pp. 3, June 1992.

13. S. Chang and L. Leung, "A Knowledge-Based Message Management System," *ACM Transactions on Office Information Systems*, vol. 5, no. 3, pp. 213–236, 1987.

14. H. Chen and V. Dhar, "A Knowledge-Based Approach to the Design of Document-Based Retrieval Systems," in *Proceedings of ACM Conference on Office Information System*, Cambridge, MA, pp. 281–290, April 1990.

15. S. Chiang, J. Wang, M. Bieber, and P. Ng, "An Event-Driven Visual Programming Environment for Office Automation through Document Processing," in *Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering*, Jurmala, Latvia, pp. 454–461, June 1994.

16. S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa, and A. Pathria, "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and System," *ACM Transactions on Office Information Systems*, vol. 4, no. 4, pp. 345–383, 1986.

17. W. Chu, M. Merzbacher, and L. Berkovich, "The Design and Implementation of CoBase," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Washington, D.C., pp. 517–522, June 1993.

18. M. Consens and A. Mendelzon, "Hy^{+}: A Hygraph-Based Query and Visualization System," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Washington, D.C., pp. 511–516, June 1993.

19. W. Croft and R. Rovetz, "Interactive Retrieval of Office Documents," in *Proceedings of ACM Conference on Office Information Systems*, New York, 1988.

20. W. Croft, "User-Specified Domain Knowledge for Document Retrieval," in *Proceedings of the 9th ACM SIGIR International Conference on Research and Development in Information Retrieval*, Pisa, Italy, pp. 201–206, 1986.

21. W. Croft and D. Stemple, "Supporting Office Document Architecture with Constrained Type," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, San Francisco, CA, pp. 504–509, May 1987.

22. C. Date, "The Data Dictionary," *A Guide to INGRES*, Addison-Wesley Publishing Company, 1989.

23. A. D'Atri and L. Tarantino, "From Browsing to Querying," *IEEE Data Engineering*, vol. 12, no. 2, pp. 46–53, June 1989.

24. J. Davis and R. Bonnell, "EDICT - An Enhanced Relational Data Dictionary: Architecture and Example," in *Proceedings of the 4th International Conference on Data Engineering*, Los Angeles, California, pp. 184–191, February 1988.

25. B. Defude, *Knowledge-Base System versus Thesaurus: An Architecture Problem about Expert System Design*, Cambridge University Press, New York, 1984.

26. R. Fiker and T. Kehler, "The Role of Frame-Based Representation in Reasoning," *Communications of ACM*, vol. 28, pp. 904–920, 1985.

27. N. Fuhr, "A Probabilistic Framework for Vague Queries and Imprecise Information in Database," in *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia, pp. 696–707, 1990.

28. N. Fuhr, "Integration of Probabilistic Fact and Text Retrieval," in *Proceedings of the 15th ACM SIGIR International Conference on Research and Development in Information Retrieval*, Copenhagen, Denmark, pp. 211–222, June 1992.

29. N. Fuhr and C. Buckley, "A Probabilistic Learning Approach for Document Indexing," *ACM Transactions on Information Systems*, vol. 9, no. 3, pp. 223–248, July 1991.

30. S. Gadia, S. Nair, and Y. Poon, "Incomplete Information in Relational Temporal Databases," in *Proceedings of 18th International Conference on Very Large Data Bases*, Vancouver, Canada, pp. 395–407, August 1992.

31. S. Gauch and J. Smith, "Search Improvement via Automatic Query Reformulation," *ACM Transactions on Information Systems*, vol. 9, no. 3, pp. 249–280, July 1991.

32. S. Gibbs and D. Tsichritzis, "A Data Modeling Approach for Office Information Systems," *ACM Transactions on Office Information Systems*, vol. 1, no. 4, pp. 299–319, 1983.

33. S. Gibbs and D. Tsichritzis, "Document Presentation and Query Formulation in Muse," in *Proceedings of the 9th ACM SIGIR International Conference on Research and Development in Information Retrieval*, Pisa, Italy, pp. 23–29, 1986.

34. X. Hao, J. Wang, M. Bieber, and P. Ng, "A Tool for Classifying Office Documents," in *Proceedings of the 5th International Conference on Tools with Artificial Intelligence*, Boston, MA, pp. 427–434, November 1993.

35. X. Hao, J. Wang, and P. Ng, "Nested Segmentation: An Approach for Layout Analysis in Document Classification," in *Proceedings of the 2nd IAPR Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, pp. 319–322, October 1993.

36. P. Hayes and S. Weinstein, "CONSTRUE/TIS: A System for Content-Based Indexing of a Database of News Stories," in *AAAI Proceedings of the 2nd Annual Conference on Innovative Applications of Artificial Intelligence*, Washington, D.C., pp. 1–5, 1990.

37. C. Herot, "Spatial Management of Data," *ACM Transactions on Database Systems*, vol. 5, no. 4, pp. 493–513, December 1980.

38. W. Horak, "Office Document Architecture and Office Document Interchange Formats-Current Status of International Standardization," *IEEE Computer*, vol. 18, no. 10, pp. 50–60, 1985.

39. E. Horowitz and R. Williamson, "SODOS: A Software Documentation Support Environment-its Use," *IEEE Transactions on Software Engineering*, vol. 12, no. 11, pp. 1076–1087, 1986.

40. W. Howden, "Contemporary Software Development Environments," *Communications of ACM*, vol. 25, no. 5, pp. 318–329, 1982.

41. T. Ichikawa and M. Hirakawa, "ARES: A Relational Database with the Capability of Performing Flexible Interpretation of Queries," *IEEE Transactions on Software Engineering*, vol. 12, no. 5, pp. 624–634, May 1986.

42. T. Imielinski, "Incomplete Information in Logical Databases," *IEEE Data Engineering*, vol. 12, no. 2, pp. 29–40, June 1989.

43. G. Jakobson, G. Lafond, E. Nyberg, and G. Piatetsky, "An Intelligent Database Assistant," *IEEE Expert*, vol. 1, pp. 65–78, 1986.

44. J. Kalita, "Generating Summary Responses to Natural Language Database Queries," Tech. Rep. TR84-9, University of Saskatchewan, 1984.

45. M. Kao, N. Cercone, and W. Luk, "What Do You Mean "Null"? Turning Null Responses into Quality Responses," in *Proceedings of the 3th International Conference on Data Engineering*, Los Angeles, California, pp. 356–362, February 1987.

46. J. Kaplan, "Cooperative Responses from a Portable Natural Language Query System," *Artificial Intelligence*, vol. 19, no. 2, pp. 165–187, 1982.

47. W. Kent, "Consequences of Assuming a Universal Relation," *ACM Transactions on Database Systems*, vol. 6, no. 4, pp. 539–556, 1981.

48. H. Korth, G. Kuper, J. Feigenbaum, A. Gelder, and J. Ullman, "System/U: A Database System Based on the Universal Relation Assumption," *ACM Transactions on Database Systems*, vol. 9, no. 3, pp. 331–347, 1984.

49. S. Lee, "An Extended Relational Database Model for Uncertain and Imprecise Information," in *Proceedings of the 18th International Conference on Very Large Data Bases*, Vancouver, Canada, pp. 395–407, August 1992.

50. Q. Liu, "An Office Document Retrieval System with the Capability of Processing Incomplete and Vague Queries," Ph.D. Thesis Proposal, Department of Computer and Information Science, New Jersey Institute of Technology, January 1993.

51. Q. Liu, J. Wang, and P. Ng, "On Research Issues Regarding Uncertain Query Processing in an Office Document Retrieval System," *Journal of Systems Integration*, vol. 3, no. 3, pp. 163–194, 1993.

52. E. Lutz, H. Kleist-Retzow, and K. Hoernig, "MAFIA –An Active Mail-Filter-Agent for an Intelligent Document Processing Support," *Multi-user Interface and Applications*, Elsevier, pp. 16–32, 1990.

53. D. Maier, J. Ullman, and M. Vardi, "On the Foundations of the Universal Relation Model," *ACM Transactions on Database Systems*, vol. 9, no. 2, pp. 283–308, June 1984.

54. T. Malone, K. Grantm, K. Lai, R. Rao, and D. Rosenblitt, "Semistructured Messages are Surprisingly Useful for Computer-Supported Coordination," *ACM Transactions on Office Information Systems*, vol. 5, no. 2, pp. 115–131, 1987.

55. P. Martin, I. MacLeod, and B. Nordia, "A Design of a Distributed Full Text Retrieval System," in *Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 131–137, 1986.

56. J. Mayfield and C. Nicholas, "SNITCH: Augmenting Hypertext Documents with a Semantic Net," *International Journal of Intelligent and Cooperative Information Systems*, vol. 2, no. 3, pp. 335–351, 1993.

57. K. McCoy, "Augmenting a Database Knowledge Representation for Natural Language Generation," in *Proceedings of the 20th ACL*, Toronto, Ontario, pp. 121–128, 1982.

58. K. McKeown, "The TEXT System for Natural Language Generation: An Overview.," in *Proceedings of 20th ACL*, Toronto, Ontario, pp. 113–120, 1982.

59. U. Merz and R. King, "DIRECT: A Query Facility for Multiple Database," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, San Diego, CA, pp. 2, June 1992.

60. F. Mhlanga, "$\mathcal{D}$-Model and $\mathcal{D}$-Algebra: A Data Model and Algebra for Office Documents," Ph.D. Thesis, Department of Computer and Information Science, New Jersey Institute of Technology, May 1993.

61. F. Mhlanga, J. Wang, T. Shiau, and P. Ng, "A Query Algebra for Office Documents," in *Proceedings of the 2nd International Conference on Systems Integration*, Morristown, NJ, pp. 458–467, June 1992.

62. M. Morgenstern, "The Role of Constraints in Database, Expert Systems, and Knowledge Representation," in *Proceedings of the 1st International Workshop on Expert Database Systems*, Kiawah Island, South Carolina, pp. 351–368, October 1984.

63. A. Motro, "BAROQUE: A Browser for Relational Databases," *ACM Transactions on Office Information Systems*, vol. 4, no. 2, pp. 164–181, April 1986.

64. A. Motro, "Construction Queries from Tokens," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington D.C., pp. 120–131, May 1986.

65. A. Motro, "Extending the Relational Database Model to Support Goal Queries," in *Proceedings of 1st International Conference on Expert Database Systems*, Charleston, SC, pp. 129–150, April 1986.

66. A. Motro, "SEAVE: A Mechanism for Verifying User Presuppositions in Query Systems," *ACM Transactions on Office Information Systems*, vol. 4, no. 4, pp. 312–330, October 1986.

67. A. Motro, "VAGUE: A User Interface to Relational Database that Permits Vague Queries," *ACM Transactions on Office Information Systems*, vol. 6, no. 3, pp. 187–214, July 1988.

68. A. Motro, "A Trio of Database User Interfaces for Handling Vague Retrieval Requests," *IEEE Data Engineering*, vol. 12, no. 2, pp. 54–63, June 1989.

69. A. Motro, "FLEX: A Tolerant and Cooperative User Interface to Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 2, pp. 231–246, June 1990.

70. R. Oddy, "Information Retrieval through Development Environment," *Journal of Documentation*, vol. 33, pp. 1–14, 1977.

71. C. Parent and S. Spaccapietra, "An Algebra for a General Entity-Relationship Model," *IEEE Transactions on Software Engineering*, vol. 11, no. 7, pp. 634–643, 1985.

72. K. Parsaye, M. Chignell, S. Khoshafian, and H. Wong, *Intelligent Databases*, John Wiley and Sons, Inc., U.S.A., 1989.

73. S. Pollitt, "CANSEARCH: An Expert Systems Approach to Document Retrieval," *Information Processing and Management*, vol. 23, no. 2, pp. 119–138, 1987.

74. S. Pollock, "A Rule-Based Massage Filtering System," *ACM Transactions on Office Information Systems*, vol. 6, no. 3, pp. 232–254, 1988.

75. H. Prade and C. Testemale, "Generalizing Database Relational Algebra for the Treatment of Incomplete or Uncertain Information and Vague Queries," *Information Science*, vol. 34, pp. 115–143, 1984.

76. F. Rabitti, "A Model for Multimedia Documents," *Office Automation*, pp. 227–250, 1985.

77. R. Rada and B. Martin, "Augmenting Thesauri for Information Systems," *ACM Transactions on Office Information Systems*, vol. 5, no. 4, pp. 378–392, October 1987.

78. C. Rolland and C. Proix, "An Expert System Approach to Information System Design," *Information Processing*, pp. 251–257, 1986.

79. G. Salton, *Automatic Text Processing*, Addison-Wesley, 1989.

80. L. Saxton and V. Raghavan, "Design of an Integrated Information Retrieval/ Database Management System," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 2, pp. 231–246, June 1990.

81. R. Schank and W. Lehnert, "The Concept Content of Conversation," in *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, Tokyo, Japan, pp. 769–771, 1979.

82. P. Schauble, "Thesaurus Based Concept Spaces," in *Proceedings of the 10th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 254–262, 1987.

83. F. Shih, S. Chen, D. Hung, and P. Ng, "A Document Segmentation Classification and Recognition System," in *Proceedings of the 2nd International Conference on Systems Integration*, Morristown, NJ, pp. 258–267, June 1992.

84. P. Shoval, "Expert/Consultation System for the Retrieval Database with Semantic Network of Concepts," in *Proceedings of the 4th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pp. 145–149, 1981.

85. E. Sibley, "An Expert Database System Architecture Based on an Active and Extensible Dictionary System," in *Proceedings of the 1st International Workshop on Expert Database Systems*, Kiawah Island, South Carolina, pp. 401–422, 1986.

86. M. Siegel and S. Madnick, "A Metadata Approach to Resolving Semantic Conflicts," in *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, Spain, pp. 133–145, September 1991.

87. P. Smith, S. Shute, and D. Galdes, "Knowledge-Based Search Tactics for an Intelligent Intermediary System," *ACM Transactions on Information Systems*, vol. 7, no. 3, pp. 246–270, July 1989.

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

89. M. Stonebraker, H. Stettner, N. Lynn, J. Kalash, and A. Guttman, "Document Processing in a Relational Database System," *ACM Transactions on Office Information Systems*, vol. 1, no. 2, pp. 143–158, April 1983.

90. V. Tahani, "A Conceptual Framework for Fuzzy Query Processing - A Step Toward Very Intelligent Database Systems," *Information Processing & Management*, vol. 13, pp. 289–303, 1977.

91. R. Thomas, H. Forsdick, T. Crowley, R. Schaaf, R. Thomlinson, V. Travers, and G. Robertson, "Diamond: A Multimedia Message System Build on a Distributed Architecture," *IEEE Computing*, vol. 18, no. 12, pp. 65–78, 1985.

92. L. Tokuda, "Computer Assist Humans in Human Resources," in *AAAI Proceedings of the 2nd Annual Conference on Innovative Applications of Artificial Intelligence*, Washington, D. C, pp. 31–35, 1990.

93. R. Tong, "RUBRIC: An Environment for Full Text Information Retrieval," in *Proceedings of the 8th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pp. 243–251, 1985.

94. F. Tou, M. Williams, R. Fikes, A. Henderson, and T. Malone, "RABBIT: An Intelligent Database Assistant," in *Proceedings of the National Conference of AI*, Pittsburgh, PA, pp. 314–318, August 1982.

95. D. Tsichritzis, "Form Management," *Communications of ACM*, vol. 25, no. 7, pp. 453–477, 1982.

96. D. Tsichritzis, S. Christodoulakis, A. Lee, and J. Vandenbroek, "A Multimedia Office Filing System," *Office Automation*, Springer-Verlag, Berlin, pp. 43-65, 1985.

97. F. Tuijnman and H. Afsarmanesh, "Management of Shared Data in Federated Cooperative Peer Environment," *International Journal of Intelligent and Cooperative Information Systems*, vol. 2, no. 4, pp. 451–473, 1993.

98. A. Tzvieli, "Representation and Access of Uncertain Relational Data," *IEEE Data Engineering*, vol. 12, no. 2, pp. 21–28, June 1989.

99. J. Ullman, *Principles of Database Systems*, Computer Science Press, 2nd ed., 1982.

100. J. Ullman, " On Kent's "Consequences of Assuming a Universal Relation"," *ACM Transactions on Database Systems*, vol. 8, no. 4, pp. 637–643, 1983.

101. L. Vieile, "Recursive Axioms in Deductive Database: The Query/Subquery Approach," in *Proceedings of the 1st International Conference on Expert Database Systems*, Charleston, South Carolina, April 1986, Benjamin/Cummings, 1987.

102. R. Wall, "Intelligent Indexing and Retrieval: A Man–Machine Partnership," *Information Processing and Management*, vol. 16, pp. 73–90, 1980.

103. J. Wan, M. Bieber, J. Wang, and P. Ng, "Document Management through Hypertext: A Logic Modeling Approach," in *Proceedings of the 27th International Conference on System Sciences*, Kauai, Hawaii, pp. 558–568, January 1994.

104. J. Wang, F. Mhlanga, Q. Liu, W. Shang, and P. Ng, "Intelligent Documentation Support Environment," in *Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, pp. 429–436, June 1993.

105. J. Wang, F. Mhlanga, Q. Liu, W. Shang, and P. Ng, "Database Support for Software Documentation: The TEXPROS Project," To appear as a book chapter in *Software Automation and Productivity Improvement*, 1995.

106. J. Wang, F. Mhlanga, and P. Ng, "A New Approach to Modeling Office Documents," *ACM SIGOIS Bulletin*, vol. 14, no. 2, pp. 46–55, December 1993.

107. J. Wang and P. Ng, "TEXPROS: An Intelligent Document Processing System," *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, no. 2, pp. 171–196, June 1992.

108. C. Wei, J. Wang, and P. Ng, "A Knowledge Based Document Classification Tool," in *Proceedings of the 3rd International Conference on Systems Integration*, Sao Paulo, Brazil, pp. 1166–1175, August 1994.

109. K. Whang, A. Ammann, A. Bolmarcich, M. Hanrahan, G. Hochgesang, K. Huang, A. Khorasani, R. Krishnamurthy, G. Sockut, P. Sweeney, V. Waddle, and M. Zloof, "Office-by-Example: An Integrated Office System and Database Manager," *ACM Transactions on Office Information Systems*, vol. 5, no. 4, pp. 393–427, 1987.

110. D. Woelk, W. Kim, and W. Luther, "An Object-Oriented Approach to Multimedia Databases," in *Proceedings of ACM SIGMOD International Conference Mangement Data*, Washington, D. C, pp. 311–325, 1986.

111. H. Wong and I. Kuo, "GUIDE: Graphical User Interface for Database Exploration," in *Proceedings of the 8th International Conference on Very Large Data Bases*, Mexico City, Mexico, pp. 1–10, September 1982.

112. X. Wu and T. Ichikawa, "KDA: A Knowledge-Based Database Assistant with a Query Guiding Facility," *IEEE Transactions on Knowledge and Date Engineering*, vol. 4, no. 5, pp. 443–453, October 1992.

113. L. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 3, pp. 177–200, 1965.

114. L. Zadeh, "Fuzzy Sets as a Basis for a Theory of Possibility," *Fuzzy Sets and Systems*, vol. 1, no. 1, pp. 3–28, 1978.

115. M. Zemankova, "FIIS: A Fuzzy Intelligent Information System," *IEEE Data Engineering*, vol. 12, no. 2, pp. 11–20, June 1989.

116. M. Zemankova and A. Kandel, "Implementing Imprecision in Information Systems," *Information Science*, vol. 37, no. 2, pp. 107–141, December 1985.

117. Z. Zhu, J. McHugh, J. Wang, and P. Ng, "A Formal Approach to Modeling Office Information Systems," To appear in *Journal of Systems Integration*, vol. 4, no. 4, 1994.

118. M. Zloof, "Query-by-Example: A Database Language," *IBM System Journal*, vol. 16, no. 4, pp. 324–343, 1977.