Spring 5-31-1998

Knowledge-based document filing for texpros

Xien Fan
New Jersey Institute of Technology

ABSTRACT

KNOWLEDGE-BASED DOCUMENT FILING FOR TEXPROS

by
Xien Fan

This dissertation presents a knowledge-based document filing system for TEXPROS. The requirements of a personal document processing system are investigated. In order for the system to be used in various application domains, a flexible, dynamic modeling approach is employed by getting the user involved in document modeling. The office documents are described using a dual-model which consists of a document type hierarchy and a folder organization. The document type hierarchy is used to capture the layout, logical and conceptual structures of documents. The folder organization, which is defined by the user, emulates the real world structure for organizing and storing documents in an office environment.

The document filing and retrieval are predicate-driven. The user can specify filing criteria and queries in terms of predicates. The predicate specification and folder organization specification are described. It is shown that the new specifications can prevent falsedrops which happen in the previous approach.

The dual models are incorporated by a three-level storage architecture. This storage architecture supports efficient document and information retrieval by limiting the searches to those frame instances of a document type within those folders which appear to be the most similar to the corresponding queries. Specifically, a three-level retrieval strategy is used in document and information retrieval. Firstly, a knowledge-based query preprocess is applied for efficiently reducing the search space to a small set of frame instances, using the information in the query formula. Secondly, the knowledge and content-based retrieval on the small set of frame instances is applied.

Finally, the third level storage provides a platform for adopting potential content-based multimedia document retrieval techniques.

A knowledge-based predicate evaluation engine is described for automating document filing. The dissertation presents a knowledge representation model. The knowledge base is dynamicly created by a learning agent, which demonstrates that the notion of flexible and dynamic modeling is applicable.

The folder organization is implemented using an agent-based architecture. Each folder is monitored by a filing agent. The basic operations for constructing and reorganizing a folder organization are defined. The dissertation also discusses the cooperation among the filing agents, which is needed for implementing the folder organization.

KNOWLEDGE-BASED DOCUMENT FILING FOR TEXPROS

by
Xien Fan

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer and Information Science

May 1998

APPROVAL PAGE

Knowledge-Based Document Filing for TEXPROS

Xien Fan

4/18/98

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

4/18/98

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

4/18/98

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

4/18/98

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

4/18/98

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

BIOGRAPHICAL SKETCH

Author: Xien Fan

Degree: Doctor of Philosophy

Date: May 1998

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, New Jersey, 1998
- Master of Computer Science and Engineering,
  Tsinghua University, Beijing, China, 1989
- Bachelor of Computer Science and Engineering,
  Tsinghua University, Beijing, China, 1986

Major: Computer Science

Publications:

Xien Fan, Peter A. Ng, "Personal Document Management and Retrieval: A Knowledge-Based Approach", To appear in *Journal of Systems Integration*, vol. 8, no. 3, 1998.

Xien Fan, Peter A. Ng, "A Dual Model Approach For Modeling Office Documents", To appear in *Proceedings of the Third World Conference on Integrated Design & Process Technology*, 1998.

Xien Fan, Simon Doong, Peter A. Ng, Ching-Song Don Wei, "A Process for Constructing a Personal Folder Organization", To appear in *Processings of the IEEE International Workshop on Multi-Media Database Management System*, 1998.

Xien Fan, Qianhong Liu, Peter A. Ng, "A Multimedia Document Filing System", In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pp.492-499, June, 1997.

Xien Fan, Qianhong Liu, Peter A. Ng, "Knowledge-based Document Filing: TEXPROS Approach", In *Proceedings of the 13th International Conference on Advanced Science and Technology in conjunction with the 2nd International Conference on Multimedia Information Systems*, pp.58-67, Apr. 1997.

X. Li, J. Hu, X. Fan, C.Y. wang and P. A. Ng, "Automated Document Filing and Retrieval", To appear in *Proceedings of the Third World Conference on Integrated Design & Process Technology*, 1998.

S. Doong, C. Wei, X. Fan, D.C Hung, P. A. Ng, "A Folder Organization Model in the Office Environment", To appear in *Processings of the 4th International Conference on Information Systems Analysis and Synthesis*, 1998.

This dissertation is dedicated to
my son
David J. Fan
and
my wife
Fang Sheng

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my advisor, Professor Peter A. Ng, for his guidance, support, and constant encouragement through this work. Special thanks are given to Dr. Murat M. Tanik, Dr. D.C. Douglas Hung, Dr. Ronald S. Curtis and Dr. Tina Taiming Chu for actively participating in my committee.

I wish to express my appreciation to Dr. Ronald S. Curtis for his helpful comments and crucial feedbacks to this dissertation.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Large quantity of documents, either in hard-copy or electronic format, is being processed in today's office environment. The objective of TEXPROS (TEXt PROcessing System) is to provide a computerized environment for users to manipulate their personal documents. TEXPROS provides functional capabilities of classifying, filing, storing, retrieving, and reproducing documents, as well as extracting, browsing, retrieving and synthesizing information from a variety of documents. As more and more multimedia documents are being produced in the office environment, the efficiency and effectiveness of document and information retrieval is becoming more critical. Many efforts have been made to narrow down the search space based on user provided information about the retrieved documents. Document organization plays an important role in doing so by trying to group the documents in such a way that the retrieval can be carried out based on a small amount of documents.

1.1 Related Work on Document Processing

A considerable amount of research has focused on document filing and retrieval [3, 4, 5, 6, 8, 12, 19, 40, 45, 46, 47, 48, 52, 54, 57]. Many document processing systems have been developed in the past. Basically, they fall into four categories.

1. The first group deals with multimedia information including text, image and voice data. Diamond [58], for example, allows users to create, edit, and transmit multimedia documents. However, it does not explore in depth the retrieval methods. MULTOS [57], on the other hand, supports a well-defined query language and many query processing techniques. MINOS [7] provides integrated facilities for creating complex document

objects, and for extracting and formulating new information from existing documents.

2. The second group deals with text-based or bibliographic information retrieval. A recent work is Kabiria's [5] distributed client/server architecture which supports the classification, filing, and retrieval of documents and the maintenance of system knowledge. The retrieval systems RUBRIC [38], Intelligent Interface for Information Retrieval (I^3R) [11] and Grant [9] use knowledge-based techniques to support query processing, natural language understanding, text understanding, and classification. RUBRIC aims to provide more automated access to unformatted textual databases. I^3R uses diverse kinds of knowledge to provide intelligent assistance to help users find information in a textual database. Grant addresses the set of documents needed to submit a grant request for supporting scientific research, by assuming that a specific application domain can be precisely defined using a rule-based knowledge system.

3. The third group is concerned with message exchanging and filtering. The goal of such systems is to help users filter, sort and prioritize messages that are already addressed to them, and also help them find useful messages they would not otherwise have received. Relevant work includes INFORMATION LENS system [36], MAFIA [35], and many others. Because of the versatility of the electronic recording media, automatic information-handling applications, such as the automatic teleconferencing systems [2, 55], electronic mail and messages [43, 49, 63], electronic information services [37, 62], electronic publications and the electronic library [61, 74], and many others, are available and utilized. While electronic information processing methods have many attractions, the long-predicted demise of conventional paper-handling systems has not yet occurred [52].

Many interesting questions are raised in the literature about paperless information systems for achieving the eventual abandonment of paper documents [13, 51, 61, 73].

4. With the advent of the World-Wide Web (WWW), networked information systems with advanced methods for browsing, searching and accessing the document collection in repositories become one of the central issues that needs to be resolved. Emerging from the explosive growth in networked connectivity and rapid advances in computer technology, the notion of standalone information utilities are replaced with interconnected digital libraries, which require effective means for the amplification of information-intensive work [50]. Libraries exist in many forms and types, such as image libraries, audio libraries, and digital video, as well as distributed text-based information systems [10]. Effective information access involves rich interactions between users and information residing in digital libraries at diverse locations. These lead to a new perspective towards information retrieval, the notion of documents, and publishing in general [25]. A selection of supporting technologies for digital libraries includes electronic publishing, hypermedia, data and information management, and education [18].

1.2 Document Organization and Modeling

Text-based documents can be described and stored by sets of keywords. Indexing techniques are used to reduce the search space to a collection of documents which contains the given keywords [45, 46]. However, there are two major limitations of these models. Firstly, it is difficult to verify whether a document is related to a given keyword which does not appear in it. Secondly, keywords may not be the only information that can be used to retrieve documents. For example, a user is trying

to find a letter from a professor. The user will get a set of non-relevant documents if he/she issues the query using "letter" and "professor" as keyterms. This is because the terms "letter" and "professor" are conceptual information, not the content of the letter.

Another approach is using structured or conceptual models to describe documents [3, 4, 5, 6, 8, 19, 40, 45, 46, 47, 48, 57, 60]. However, the systems using this approach are designed for special purpose since their models are domain dependent. Kabiria, for example, is a knowledge-based filing and retrieval system [4, 5, 12, 48] designed using the notion of documents as objects embedded in a rich procedural and domain context. The context describes a document's semantics by taking into account the activities in which it is used and the domain rules that justify its existence in the office environment. The Kabiria document model consists of a conceptual model for describing and classifying documents, and a retrieval model for retrieving documents. Documents are partitioned into different classes. Documents of the same class have the same structure, meaning and roles in the office. Document classes are organized as an is-a hierarchy using generalization and specialization mechanisms. During filing, documents are organized based on their classes. Original documents are stored in a document base, while classes and conceptual structures (instances of the document classes) are stored in the model base as nodes of the model's semantic network. The document retrieval model uses referencing and linking to describe the relationships among documents and their roles in the office. The reference mechanism describes logical relationships between two conceptual structures of specific documents, whereas documents as a whole are related to each other via document links. Document retrieval is based on content, conceptual structure, role and domain dependencies, while users are allowed to browse through the semantic network at either the class or instance level. The semantic network is composed of two layers: the class layer shows the relationships

between different types of documents, and the instance layer describes, for each instance, the real environment where real documents are embedded. To maintain model consistency, two instances can be connected by an instance of a link only if the corresponding classes are connected by the corresponding link type. Constructing the instance layer and maintaining model consistency become cumbersome.

The MULTOS (MULTimedia Office Server) [19, 57, 60], aims to support multimedia document filing and retrieval according to various search parameters. Besides logical and layout structures, defined according to the Office Document Architecture (ODA) standard [24], it introduces a conceptual structure to capture the semantics of document contents. Documents with similar conceptual structures are grouped into classes and are referred to as conceptual types, which are arranged in an is-a hierarchy. Based on retrieval requirements and hardware capabilities, the MULTOS (MULTimedia Office Server) divides document filing systems into three categories: dynamic document filing systems, current document filing systems and archive document filing systems. The dynamic document filing systems are used essentially as buffers where a small number of documents are stored to be accessed and manipulated by a single user. Retrieval of the documents can be done simply by file names or superimposed codes assigned by the user. The current document filing systems are used for storing documents that are of current interest to the office, and that are frequently accessed. They may be shared and updated. Retrieval of the documents can be done by content, set of documents and document identifiers. The archive document filing systems are used for storing documents that have reached a stable state in which modification is infrequent. Document retrieval is again by content, set of documents and document identifiers. The three filing system categories are related to the document life-cycle. Documents would migrate over time from a dynamic filing system towards an archive filing system through a current filing system. From the viewpoints of storage capacities, the archive filing

systems have the greatest capacity, followed by the current filing systems, then the dynamic filing systems. However, the dynamic filing systems give the best response in comparison with the other two filing systems. One would expect a longer response time from the archive filing system because of its large size.

1.3 Requirements of Personal Document Processing System

As mentioned before, various document models are under investigation [3, 5, 6, 8, 40, 45, 46, 47, 48]. Full text systems store and retrieve textual documents based on subject content. Keyword-based systems store and retrieve documents based on manually or automatically created document records. To retrieve documents and information from these systems, users are required to express what the documents contain. This limits the efficiency and effectiveness of document and information retrieval because the systems do not take into account what users know about the documents. Structured models can capture the layout and logical structures of documents. Users are allowed to specify queries in terms of what the documents are like. Conceptual models can capture the conceptual structure and other domain knowledge such as how the documents are used, what organizational activities are related to the documents. These models, which are domain dependent, allow users to express the request in terms of what the documents are about.

However, personal information systems have not drained enough attention. The current document processing systems have limitations when they are used as personal document systems. We identified three basic requirements for personal document processing systems from document filing point of view. The first one is the need for a flexible and dynamic document model. It is impractical to build a personal document processing system for a single user. In other words, a personal document system will be used by different users. Different users understand and organize their documents in different ways. For better retrieval efficiency and effectiveness, the

system should be able to use all the knowledge each user has about the documents to be retrieved. But it is impossible for a single model to capture all the information that can be derived from the documents. Therefore, documents should be stored based on the user's knowledge about them. This allows the system to match the user's knowledge about the documents to be retrieved against the descriptions (from the user's point of view) of the documents in the document base. This matching will become more difficult if the models are predefined, like most of the systems have. As a result, it will be difficult for the user to specify queries because he/she understands documents in a different way. This causes a lot of vague queries to be issued, and in turn reduces the efficiency and effectiveness of document and information retrieval. A predefined document model will also cause the system to be domain-dependent. It is hard for the system to be used by different users.

The second requirement is the need for a document filing model. Document organization plays an important role in reducing the search space based on user-provided information about the retrieved documents, rather than searching through the whole document space. Obviously, higher efficiency and effectiveness of document search can be achieved when the user knows how the documents are organized. Therefore, a document filing model is useful in capturing the user's domain knowledge of document organization. If we allow the user to define the document filing model, the system can organize the documents the same way that the user would in the real world. As a result, the user is familiar with the document organization, and can provide helpful information for reducing the search space.

The third requirement is the capability of supporting various information retrieval techniques. Today's office documents have various media types. And the automatic semantic interpretation of some media types such as image and video is far from applicable given the state-of-the-art of computer intelligence. A personal document processing system should provide a platform for various IR

techniques. Meanwhile the possibility of using various text-based IR techniques for providing semantic content-based retrieval of multimedia documents should also be investigated.

1.4 TEXPROS Approach

TEXPROS (TEXt PROcessing System) [32, 69] is an automatic document filing and retrieval system. The system provides functional capabilities for classifying [20, 21, 22, 23, 70, 71, 72], categorizing [14, 15, 41, 42, 76, 77], storing [14, 15, 32, 68, 76], retrieving [30, 31, 32, 33, 34] and reproducing [68, 69] documents, as well as extracting[20, 22], browsing [31, 65], retrieving [30], and synthesizing [30, 68, 69] information from a variety of documents.

The intent of our research is to develop a personal document processing system based on existing technologies. There are two major contributions of this work. The first one is that it uses a flexible dual-model approach to modeling office documents. The dual-model consists of a document type hierarchy and a folder organization, both of which can be defined by the user. The document type hierarchy describes the conceptual structure of the documents. And the folder organization, used as the filing model, emulates the real world structure for organizing and storing documents in an office environment. The dual-model provides a flexible and dynamic modeling method since it is defined by the user. This gives the system the capability to be used in various application domains. Secondly, the system can support precise queries. Different users understand the documents from different point of view. Since the dual-model is flexible and dynamic, the documents can be described and stored as the user expects. Therefore, the user is allowed and able to specify queries based on whatever he/she knows such as the content (keywords), layout and logical structure, conceptual structure, and domain knowledge of the retrieved documents.

Using generalization and inheritance, the powerful abstractions for sharing similarities among document classes while preserving their differences, the document classes are organized as the document type hierarchy. Each document is divided into textual and nontextual parts. (The latter includes essay, logos, drawings, graphics, pictures, and images.) The system permits the user to capture the meaning and synopsis of both parts and organize them into a semistructured form (which is called a frame instance). A folder organization is used as the repositories for the frame instances based on whether they meet the criteria of the folders. When retrieval occurs, the system often returns the frame instances, rather than the original documents, to the user. (In most cases, the user should be contented with the information in the frame instances.) Information search and retrieval can be speeded up by focusing on frame instances of a particular type (frame template) within a particular folder by combining the document type hierarchy and the folder organization. The document classification process is to find the best fitting type for a given document and then to instantiate a frame instance of its type (represented by a frame template) by filling significant information of the document pertinent to the user into the underlying frame template. This reduces the document size considerably without any severe loss of content. Storage space and processing time are saved in many applications by using frame instances instead of full documents. By doing so, it speeds up the information retrieval considerably. In addition, the system exploits many AI techniques (e.g., organizing the knowledge base as a three dimensional semantic network) to support intelligent retrieval, which is absent in the previous multimedia document systems.

This dissertation is focused on document filing. We first introduce the document model in Chapter 2, and then introduce the previous work on document filing in Chapter 3. Chapter 4 describes the predicate-based representation of documents. Predicates are used by the user to specify filing criteria and queries.

In Chapter 5, we extend the folder organization by introducing a new link between folders for simplifying local predicates. In Chapter 6, a system repository architecture is described which takes into account the dual document models. By incorporating the document type hierarchy and folder organization into the storage system, it facilitates the document search and retrieval. A knowledge-based query preprocessing algorithm is given for reducing the search space using the information contained in the query formula. It is shown that this architecture can also support various text-based information retrieval techniques and content-based multimedia IR techniques. In Chapter 7, an evaluation mechanism is proposed, which can evaluate predicates using the knowledge base. The knowledge base consists of an object base and a domain knowledge base. A knowledge representation model is given to maintain the knowledge. Chapter 7 also presents a learning agent for acquiring domain knowledge needed by the predicate evaluation engine. An agent-based architecture for implementing the folder organization is proposed in Chapter 8. Chapter 9 describes the overall architecture and the implementation of the filing system. Future work is also discussed.

CHAPTER 2

THE DUAL MODELS

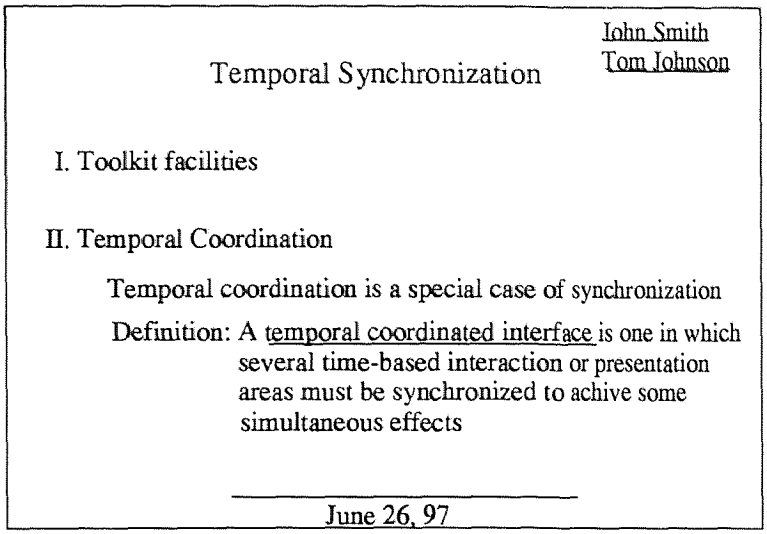
TEXPROS employs a dual modeling approach for describing, classifying, categorizing, filing and retrieving documents. This document model consists of two hierarchies: a document type hierarchy which depicts the structural organization of the documents, and a folder organization which describes the user's real-world document organization structure.

2.1 The Document Type Hierarchy

In a user's office environment, by identifying common properties for each document class, documents are partitioned into different classes. Each document class is represented by a frame template which describes the common properties in terms of attributes [44] of the class and is referred to as the document type (or simply type). As a powerful abstraction for sharing similarities among document classes while preserving their differences, the frame templates are related by specialization and generalization and are organized as a document type hierarchy whose members are related by an is-a relationship. This is-a relationship and the mechanism of inheritance help to reduce the complexity of models and redundancy in specifications [56]. After its classification [20, 21, 23, 70, 71, 72], a particular office document, summarized from the viewpoint of its frame templates, yields a synopsis of the document which is called a frame instance. Figure 2.1 shows the frame template and frame instance of a viewgraph of slide.

2.2 The Folder Organization

The frame instances of different types are deposited into folders over time. Folders are heterogeneous repositories of frame instances and are organized to form a folder



(a)

Type			
Title			
Author			
Date			
[Description]			
<Topic>	Topic Name		
	[Description]		
	<Topic>		
	<Definition>	Concept	
		Description	

(b)

Type	Slide		
Title	Temporal Synchronization		
Author	John Smith, Tom Johnson		
Date	June 26, 97		
Topic	Topic name	Toolkit facilities	
Topic	Topic Name	Temporal Coordination	
	Description	.....	
	Definition	Concept	Temporal Coordinated Interface
		Description	.....

(c)

Figure 2.1 (a) An original slide (b) The frame template for the slide type (c) The frame instance of the slide in (a)

organization [16], which is one of the common ways of organizing and storing documents for their retrieval in an office environment. The folder organization is defined by a user corresponding to the user's view of the document organization, which is obtained by repeatedly dividing documents for particular areas of discourse into groups until the user believes the groups are descriptive. Each folder has a user defined criterion for governing the automatic document filing. A predicate based representation of documents is used to specify criteria for the folder organization.

One of the disadvantages of text retrieval using inverted index files is that the information pertaining to a document is scattered among many different inverted term lists. Information relating to different documents with similar term assignments is not in close proximity within the file system. The inverted file strategy always remains faster than cluster searches, but it does not provide easy access to the complete frame instances in the folders, nor does it have a browsing capability. Browsing is to be permitted in document collection when data about related items are close together [52]. In a cluster file organization [27, 64], records that have similar frequency distributions for attribute values are grouped together, without taking into account the degree of closeness between two specific records. This clustered structure can, in fact, be used for both search and browsing, since similar documents are collected in a common group in a clustered file. Clustered file searches can be effective in retrieving the wanted items when similarity associations between documents convey information about the joint relevance of documents to the queries. In this organization, once clusters are formed, it would be possible to determine in which cluster a new document would fall, and to find which cluster of records a query best fits [39]. However, many documents, which may have little in common with the query, must be compared with the query formulation [52]. Furthermore, the use of clustering strategies in information retrieval has been limited in practice, because of the expense of generating the cluster structures for large collections of documents

[53]. Analogous to a cluster file organization, the folder organization provides efficient frame instance access by limiting the searches to those frame instances of a specific document type in a folder (analogous to document clusters) which appear to be most relevant to queries in a collection. This search strategy filters out first any frame instances of irrelevant document types and then compares the contents of a small collection of frame instances of an identified document type against query formulations, this is quite analogous to the global-local text similarity comparison [53], and should produce a high degree of retrieval precision. In addition, it is convenient to distinguish the folder-generation process from folder-search strategies. In normal circumstances, the DAG structured folder organization is generated only once, and its maintenance including folder reorganization can be carried out at relatively infrequent intervals. Document search operations [14, 15, 30, 31, 32, 33, 34, 65], on the other hand, may have to be performed continually and efficiently.

CHAPTER 3
PREVIOUS WORK

An agent-based filing architecture was first proposed to implement TEXPROS's document filing subsystem [32, 68, 69, 76, 77] for automating document filing (i.e., placing an incoming frame instance in appropriate folders) and for coping with folder reorganization. There are two types of agents in the system: filing agent and storage agent. Each agent has its own private data structures and operations for manipulating these data structures. The agents communicate with each other through message passing. As it is monitored by a filing agent, each folder is associated with a criterion. All frame instances in the folder and its descendants must satisfy the criterion of that folder. An agent receives copies of the frame instances from its parent and distributes them to its children if their contents satisfy the criteria of the children. Each agent stores frame instances in two places: repository and output_buffer. However, a storage agent is a special type of filing agents without output_buffer, for it does not send frame instances to any other agents. All the frame instances in the storage agent will be kept in the repository, and may be collected and redistributed by the parents of a storage agent.

In [32, 76, 77], a DAG (Directed Acyclic Graph) structured folder organization is presented, in which each node represents a folder and each edge (f_1, f_2) denotes that the folder f_2 is a child folder of f_1 (or f_1 is a parent folder of f_2). As its criterion, each folder has a user-defined predicate, called the local predicate, which governs the filing of frame instances into it. To be filed in the folder f_2 from a folder f_1, a frame instance must satisfy the local predicate of the folder f_1, and, in turn, it must satisfy the local predicate of the folder f_2. A filing path from the root folder f_0 to folder f_1 in the folder organization is a path from f_0 to f_1 in the DAG; and therefore, to be filed in the folder f_1 from the root folder f_0 through the filing path, a frame instance must satisfy every local predicate on the filing path by evaluating these predicates

against the contents of the frame instance. The global predicate for a folder f_1 in a folder organization is the ORing of the predicates of all the possible filing paths from f_0 to f_1; and the global predicate of each filing path from f_0 to f_1 is an ANDing of the local predicates of all the folders on the path. Thus, a folder groups together those frame instances, regardless of their document types, which satisfy its global predicate.

3.1 Problems and Limitation

However, there are four major limitations of the previous work. The first one is that the predicate specification is ambiguous. Previous work uses a simple form for specifying predicates, such as, in terms of attribute-value pairs. This ambiguous predicate specification causes the user and the system to misunderstand each other. Therefore, the filing system cannot guarantee that a frame instance is filed in appropriate folders. Consider an example: a folder organization shown in Figure 3.1; each of the folders is associated with its local predicate as stated beside the folder. Take the slide in Figure 2.1 as a sample document. Suppose John Smith is a Ph.D. student in the CIS department, and also a faculty member of the EE department, while Tom Johnson is a staff member in the CIS department. According to [32, 76, 77], the slide will be deposited into the folder NJIT because both authors are affiliated to NJIT. Then the slide will be stored in the CIS and EE folders because John Smith is in the EE department and Tom Johnson is in the CIS department. And then it will be deposited into the CISStudent, CISPHD, EEEMP and EESFAC folders because John Smith is a Ph.D. student in the CIS department and a faculty member in the EE department. However, the folder CISFAC also receives a copy of the slide because John Smith is a faculty member, although not in the CIS department. The slide will also be deposited into the folder EEStudent because John Smith is a Ph.D. student, although not in the EE department. And the folder EESTAFF has a copy of the

slide because Tom Johnson is a staff member, but not in the EE department. The filings of the slide into the folders CISFAC, EEStudent and EESTAFF are examples of *falsedrops*; that is, a frame instance is deposited in a folder where it should not be. These falsedrops are caused by inconsistent interpretation the folders' predicates. One solution is evaluating predicates of the folders on a filing path against the same attribute-value pair, but this limits the process of creating folders' predicates at a user's disposal. And the problem can only be partially solved: only the falsedrop of filing the slide into the folder EESTAFF can be prevented. Another solution is adding a restriction that the attributes used in specifying predicates of the folders must be the same as the attributes used in frame templates, but then the predicate specification cannot be used for specifying high level information and domain knowledge about documents. The next two sections will discuss how the previous predicate specification causes falsedrops. Based on the analyses, a new predicate specification will be proposed in the next chapter that allows users to specify precise predicates, and in turn, to prevent falsedrops.

The second limitation is that the domain knowledge, used by the filing system for determining whether a frame instance satisfies a predicate, is predefined. As a personal document processing system, TEXPROS allows the user to get involved in document modeling for meeting different users' needs. A predefined domain knowledge base cannot support such a flexible and dynamic modeling approach. The system must have self-learning capability for enriching the domain knowledge base.

The third limitation is the lack of a system storage architecture which can incorporate the folder organization and the document type hierarchy. TEXPROS employs a dual modeling approach. The dual models can support efficient and effective document search and retrieval. However, a system storage architecture

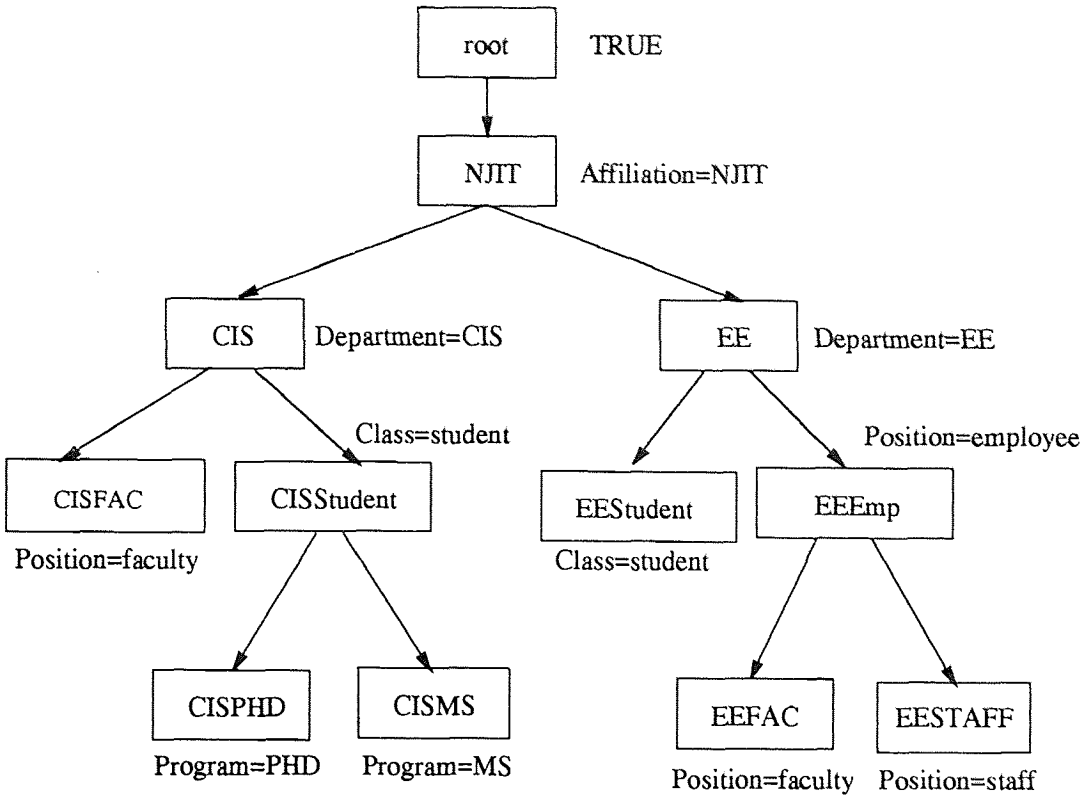


Figure 3.1 An old folder organization

is needed to combine the dual models together for speeding up the document and information retrieval.

Finally, the folder organization is implemented using an agent-based architecture in [32, 68, 69]. Each folder is monitored by a filing agent. A set of filing agents is needed for monitoring a folder organization. However, the cooperation among these filing agents is not defined. The previous work employs a filing approach that only one copy of each frame instance exists in the folder organization. As we will see in Chapter 8, this approach requires an agent to collect frame instances from its descendant agents during document filing and folder reorganization, which is time consuming since the collection has to go through the whole branch of the folder organization.

3.2 Analysis: Level of Abstraction

The users' knowledge about a document can be specified in terms of a low level abstraction, such as keywords, or expressed in terms of a high level conceptual information, such as "the author of an article is a student". The information carried by a predicate can be considered as part of the abstraction of a document. The abstraction can be expressed at different levels. For example, attributes of a frame template, such as sender and receiver of letters, subject and authors of articles are among the abstractions of the first level. Very often, a user may specify a predicate which characterizes the properties of an object. For example, consider a predicate specification that the author of an article is a faculty member. In this case, author is an attribute of the article and faculty is considered to be the property of the author of the article which cannot be obtained directly from the frame instance. The property, which specifies the semantic and behavioral information of an object, can be derived from a deeper content analysis, looking it up in the thesaurus tailored to a particular subject area, or a preconstructured knowledge base for a particular area of discourse which describes specific relationships between entities in terms of synonym relations, whole-part relations, cause-effect relations, and so on. The properties of attributes are among the abstractions of the second level.

However, the previous predicate specification does not take the level of abstraction into account. Predicates are statements of objects. The previous predicate specification omits the objects in predicates by assuming that the object in a predicate is the frame instance and the filing system files only one frame instance each time. Since the abstractions of a frame instance can be at different levels, the objects in predicates are not always the frame instances. Take the folder EESTAFF and its filing path (from root to the folder EESTAFF) in Figure 3.1 as an example, the filing of the slide into this folder is considered as a falsedrop under the assumption that the meanings of the local predicates of folders along the filing path of the

folder EESTAFF, "Affiliation=NJIT", "Department=EE", "Position=employee", "Position=staff", are one of the authors is Affiliated to NJIT, one of the authors is in the EE department, one of the authors is an employee, one of the authors is a staff member, respectively. And the global predicate of the folder EESTAFF, which is "Affiliation=NJIT & Department=EE & Position=employee & Position=staff", means that one of the authors is a staff member in the EE department at NJIT. So these predicates are statements about the authors of the slide, not the slide itself. Obviously, the slide does not satisfy the global predicate of the folder EESTAFF although it satisfies all the local predicates of folders along the filing path of the folder EESTAFF. In other words, $\exists x.A(x) \wedge \exists x.B(x) \not\Rightarrow \exists x.A(x) \wedge B(x)$.

3.3 Analysis: Value Integrity

Another factor that may cause falsedrop is the imprecise meanings of values (or symbols) in a predicate specification. Some values are ambiguous without giving the domain. Consider the term "Ph.D. Student". It is ambiguous to say someone is a "Ph.D. Student" because it is not clear in which department at which school. Ambiguous values in predicates may cause falsedrops. As an example, the filing of the slide into the folder EEStudent in the previous example is due to the fact that John Smith is affiliated to the EE department of NJIT (a staff member of the EE department) , and a Ph.D. student (although not in the EE department). Note that the global predicate of the folder EEStudent, **Affiliation=NJIT** $\wedge$ **Department=EE** $\wedge$ **Program=student**, has ambiguous meanings. One is that John Smith is affiliated to NJIT, is in the EE department, and is a student. Another is that John Smith is a student in the EE department at NJIT. If the user takes the later one, then the global predicate of the folder EEStudent is not true, although all the local predicates of folders along the filing path of the folder EEStudent are true. To the user's understanding, the filing of the slide into the folder EEStudent is a

falsedrop. In order to understand and interpret clearly and precisely (and therefore to prevent the falsedrop of frame instances), it is necessary to disambiguate and formalize the values in a predicate specification by specifying the domain in which a value is defined. We use path-notation [75] to reference values of particular components of the aggregate hierarchy. For this case, in order to refer a Ph.D. Student of CIS department at NJIT, where "CIS" is the domain where "Ph.D. Student" (denoted as PHD) is defined, and "NJIT" is the domain where "CIS" is defined, the path-notation is "PhD.CIS.NJIT". Interchangeably, we also write PHD.(Department=CIS).(University=NJIT).

CHAPTER 4

PREDICATE-BASED REPRESENTATION OF DOCUMENTS

The major objective of this dissertation is to automatically organize the documents in the same way that the user expects, for supporting efficient and effective document search and retrieval. Specifically, the system provides the user with an intelligent GUI (Graphical User Interface) for specifying the document filing model, which is the folder organization. Through the interaction with the user, the system understands the folder organization and stores the corresponding knowledge into the knowledge base. The knowledge base is then used for supporting the document filing and retrieval. This makes the folder organization a flexible and dynamic document filing model. In order to support this filing model, a language is needed for supporting the interaction between the user and the system. The language is used for specifying knowledge about documents. The user can use it to specify criteria for governing the document filing, or to specify queries for retrieval. The language has the following properties:

- Expressive power: Users can easily use the language for specifying their filing criteria. Users can easily use the language to specify what they know about the documents being retrieved.

- Understandability: Users can understand how the documents are organized and what information is important for document search and retrieval by browsing the filing criteria. This will help users to specify precise queries.

4.1 Predicate Specification

In last chapter, we have discussed why the previous predicate specification causes falsedrops. Based on that discussion, in this section, we shall formalize the new predicate specification. Predicates are specified based on frame instances, which

contains conceptual information of the original documents. Attributes are used as identifiers in accessing information from frame instances. Since frame instances have different underlying structures (frame templates), sometimes it would be helpful in simplifying predicate specification to give a common name to a set of attributes of different frame templates. For example, Author is an attribute of the article type, Sender and Receiver are attributes of the letter type, From and To are attributes of the memo type. These attributes are the same type and all specify the persons who creates or received the documents. However, each of these attributes can only be used for one document type. In order to simplify the predicate specification, we can define a common name (say Owner) to represent all of them. This common name is called an *abstract attribute*. *Abstract attribute* is an attribute which represents a set of attributes of the similar type, that appear in some frame templates. For notion simplicity, abstract attributes are still called attributes.

Predicates are statements about objects. We allow two kinds of objects to appear in predicates. One is the frame instances. The other is objects which are related to the frame instances, i.e appearing in the frame instances as values of some attributes. This limitation is made based on the fact that we only use predicates to represent information of frame instances. Objects are represented by object identifiers. For attributes that are multi-valued, objects can be given in form of a :?i, where a is a multi-valued attribute, ? is the first letter of a and i is to denote the ith value of attribute a. For example, suppose Author is a multi-valued attribute of the article type, then Author:A1 and Author:Ai denote the first author and one of the authors, respectively. An object can also be represented simply by the attribute, which means the object appears in the values of the attribute. For example, suppose Sender is an attribute of the letter type, then "Sender" represents any of the senders of the letter if it is used as an object identifier.

Definition 4.1 *(Pattern)* A *pattern* defines a filter which can be used to convert a string to another. It may contain the special symbols ?, *, # and -, where ? stands for one character; * stands for any number of characters, # stands for one character that will be ignored, and - stands for any number of characters that will be ignored.

Definition 4.2 *(First Level Predicate Clause)* A *first level predicate clause* has the form $g(\omega, b[, r])$ where

1. g is the name of the predicate clause, and can be an attribute of the frame instance ω;
2. ω is a frame instance;
3. b is either a value or a variable; and
4. if b is a variable, r can be given as a pattern.

Definition 4.3 *(Second Level Predicate Clause)* A *second level predicate clause* has the form $g(a, b[, r])$ where

1. g is the name of the predicate clause, and can be a property name of the object a;
2. a is an object;
3. b is either a value or a variable; and
4. if b is a variable, r can be given as a pattern.

Intuitively, both the first level predicate clause and the second level predicate clause have the same syntax. The difference is that the first level predicate clauses are used to specify characteristics of frame instances, whereas the second level predicate clauses are used to specify the properties of objects which are related to the frame instances. For example, Date(ω, 4/25/96) denotes that the attribute Date of the

frame instance ω has value 4/25/96. Position(Sender, Employee.CIS.NJIT) denotes that one of the senders of the frame instance is an Employee of the CIS department at NJIT.

A predicate clause, either first level predicate clause or second level predicate clause, is called a *goal predicate clause* if its second parameter is a value. An *assignment predicate clause* is the one whose second parameter is a variable. A goal predicate clause is a statement whose evaluation is either true or false. An assignment predicate clause is to assign a value, which makes the predicate clause true, to its second parameter. For example, Program(Sender, PhD.CIS) represents that one of the senders is a Ph.D. student of the CIS department. Age(Sender, x) will assign the age of the Sender to the variable x. It should be noted that a variable can be multivalued.

In some cases, users may not be interested in the whole value of an attribute. A pattern can be used to access part of a value. The pattern given in an assignment predicate clause will be matched with the value of the attribute. The characters that are represented by the special symbols # and - will be ignored. The rest of the value will be assigned to the variable. For example, the predicate clause ContractID(ω, x, *??-*) will assign the first two digits of the value of the attribute ContractID to the variable x by using a pattern *??-* as the third parameter of g. As an another example, the predicate clause ContractID(ω, x, *##??-*) will assigned the third and the fourth digits of the value of the attribute ContractID to the variable x.

Definition 4.4 *(Predicate Constraint)* A *predicate constraint* is a relation among variables and values using the operators in {" = "," $\neq$ "," $\in$ "," $\ni$ "," < "," > "," $\leq$ "," $\geq$ "}.

Definition 4.5 *(Atomic predicate)* An atomic predicate is either a goal predicate clause or a n-tuple $(P_1, P_2, \cdots, P_n)$, where P_i, $1 \leq i \leq n$, is either an assignment predicate clause or a predicate constraint.

Example 4.1 (Age(Sender, x), Age(Receiver, y), $x > y$) is an atomic predicate which specifies that the Sender is older than the Receiver. The first two components are the second level assignment predicate clauses, and the third component is a predicate constraint. (ContractID(ω, x, *??*-), $x > 50$) specifies that a number consisting of the first two digits of ContractID is greater than 50.

Definition 4.6 *(Predicate)*

1. A truth value (TRUE or FALSE) is a predicate.
2. An atomic predicate is a *predicate.*
3. If P is a predicate, then $\neg$P is a predicate.
4. If P and Q are *predicates*, then $(P \vee Q)$ and $(P \wedge Q)$ are also *predicates.*

Definition 4.7 *(Variable declarant)* Attached to a link (f_1, f_2) in the folder organization, a *variable declarant* is used to define a variable for representing an attribute or an object appearing in the predicate of the folder f_1, which can be used to specify the local predicate of the folder f_2. The format of a declarant is var-name = string, where var-name is the variable and string is an attribute or an object.

The variable declarants are provided for sophisticated users to specify advanced folder organization. An example is given in the next chapter.

4.2 Justification of the Predicate Specification

The purpose of the predicate specification is to provide the user with a language for directing the document filing or retrieving documents. In either case, the user specifies criteria in terms of predicates. The proposed predicate specification is a modified FOPL (First Order Predicate Logic). Five major modifications have been made to create an appropriate language for specifying knowledge about documents in TEXPROS.

Firstly, FOPL can be used for describing facts about any object in the universe of discourse. The evaluation of predicates requires predefined knowledge about the involved objects. As a personal document processing system, TEXPROS employs a flexible and dynamic modeling approach for meeting different users' needs. Since the application domain is not predefined, it is not reasonable to expect a predefined knowledge base. Therefore, the system must have learning capability for acquiring the needed knowledge. The successful acquiring of knowledge depends on the availability of the information. The availability means the existence of the resources, known to the user, from which the needed knowledge can be acquired. Objects that are irrelevant to the documents are unimportant to document filing and retrieval. So there is less chance that the end user can help in acquiring the knowledge about these objects. In other words, the availability of information will be little. In the proposed predicate specification, only two kinds objects are valid: frame instances and the objects that are related to documents (i.e., appear in frame instances as values of attributes). To prevent misinterpretation of predicates and therefore falsedrops, we introduced the concepts of first level predicate clause and second level predicate clause for expressing facts about frame instances and related objects, respectively. Using two different concepts enhance the restriction about the objects.

Secondly, FOPL allows multiple objects to appear in a single atomic sentence for expressing their relationship. Predicates with multiple objects are sometimes ambiguous. For example, the predicate father(x, y) can be interpretated as "x is the father of y" or "x's father is y". The order that the objects x and y appear in the parameters is important for interpretation. It is difficult to formalize the convention such that the user and the system interpretate the predicate in the same way when the predicate has multiple objects. In the proposed predicate specification, each predicate clause has only one object. The relationships among frame instances and objects that are related to the frame instances are expressed using the

assignment predicate clauses and predicate constraints. The relationship between a frame instance and an object that is related to the frame instance is inexplicitly expressed using the object identifier such as Author:A1 which represents the first author of the document.

Thirdly, the proposed predicate specification has precise syntax including restrictions on which symbols can be used in specifying predicates. For instance, attributes of templates can be used as the name of the first level predicate clauses. The precise syntax guarantees that only the facts that are relevant to document filing and retrieval can be expressed, and therefore the predicate can be understood within the application domain.

Fourthly, we introduced the concepts of attribute and value in predicates specification. A predicate clause specifies one property, in terms of attribute-value pair, of an object. And a path-notation can be used to represents a precise value, such as FACULTY.CIS.NJIT which means Ph.D. student in the CIS department at NJIT. As an example, the predicate Position(x, FACULTY.CIS.NJIT) states that x is a faculty member of the CIS department at NJIT. Here, the attribute-value pair (Position, FACULTY.CIS.NJIT) represent a property of the object x. The same predicate can be specified as faculty(x, cis, njit) in FOPL, where x, cis and njit are objects. With the concepts of attribute and value, predicate clauses (atomic sentences in FOPL) can have fixed number of parameters (the goal predicate clauses have two parameters, and the assignment predicate clauses have two or three parameters). This simplifies the predicate evaluation, and therefore reduce the chance of misinterpretation.

Finally, there is no quantifier in the proposed predicate specification. An existential quantifier is needed for interpretating some of the object identifiers such as Author:Ai or Author. Variables in the proposed predicate specification are defined by assignment predicate clauses or variable declarants, not the quantifier. In rare cases, the universal quantifier will be needed, and its interpretation is much more

complicated and time consuming. So it is not supported in the proposed predicate specification. Therefore, there is no need to specify the existential quantifier explicitly. Another reason is that the local predicates in a folder organization are not independent each other. A filing process goes through filing paths. So the local predicates of the folders along each filing path should have the same scope. The implicit existential quantifier, if needed, should sit outside the predicate of the filing path (i.e., the ANDing of all the local predicates on the filing path).

CHAPTER 5

FOLDER ORGANIZATION

The folder organization is used to capture the users' knowledge about document organization. Defined by the user, the folder organization provides a flexible and dynamic document filing model [16]. Folders are heterogeneous repositories of frame instances and are organized to form a folder organization, which is one of the common ways of organizing and storing documents for their retrieval in an office environment. The folder organization is defined by a user corresponding to his/her view of the document organization, which is obtained by repeatedly dividing documents for particular areas of discourse into groups until the user believes the groups are descriptive. Each folder has a user defined criterion for governing the automatic document filing. The folder organization is an underlying structure of an agent-based filing architecture which will be described in Chapter 8. Each folder is associated with a filing agent containing a criterion. The criteria used to govern the document filing are defined in terms of predicates. Agents, depending on how they are connected, communicate and cooperate each other to implement the folder organization. It is shown that this filing system provides a flexible search and retrieval facility that allows browsing through collections of frame instances and retrieval of frame instances according to different criteria, using the information related to document types and the frame instances in close proximity within a folder in the folder organization. Automatic filing of frame instances into proper folders of a folder organization based on folders' criteria becomes a central issue. This is made possible by the predicate evaluation engine which is presented in Chapter 7. In this chapter, we shall describe the folder organization and its properties.

5.1 Definition of Folder Organization

In this section, we extend the notion of a rooted DAG structured folder organization with multiple typed links (sometimes, with attachment of variable declarant). A a new link type between folders, AND link, is introduced. It is shown that the predicates as the criteria for filing frame instances into folders can be simplified and, in turn, can be easily evaluated. Finally, we shall prove that the new folder organization and predicate specification can prevent falsedrops that happen in the previous work.

Definition 5.1 *(Folder Organization)* A *folder organization* is a two tuple, $\mathcal{FO}(G, \Delta) = \{ G(V,E), \Delta\}$, where

1. $G(V,E)$ (also denoted as $G(\mathcal{FO})$) is a rooted directed acyclic graph, and
   - each vertex in $V(G)$ corresponds to a folder;
   - each edge (f_i, f_j) defines a link from f_i to f_j, which can be an OR-link or an AND-link; and
   - some of the edges are labeled with variable declarants.
2. $\Delta = \{\delta_f \mid f \in V(G)\}$ is a set of predicates, where δ_f is the local predicate of a folder f.

For the sake of clarity, we shall refer to the local predicate of a folder f as the predicate of the folder f, which is distinguished from the notion of a global predicate defined later.

Let ω denote any frame instance, Ω be the entire frame instance base, and $\delta_f(\omega)$ denote that the frame instance ω satisfies the predicate δ_f of the folder f based on the knowledge and facts in the system.

Definition 5.2 *(OR-link)* A link (f_1, f_2) is called an *OR-link* if $(\forall\omega \in \Omega)\delta_{f_2}(\omega) \wedge (\omega \in f_1) \Rightarrow \omega \in f_2$.

Definition 5.3 *(AND-link)* A link (f_i, f) $(1 \leq i \leq n)$ is called *AND-link* if it has the following properties:

1. $(\forall\omega \in \Omega)\delta_f(\omega) \wedge (\omega \in f_1) \wedge (\omega \in f_2) \wedge \cdots \wedge (\omega \in f_n) \Rightarrow \omega \in f$.

2. $(f_1, f), (f_2, f), \cdots, (f_n, f)$ $(n \geq 2)$ are the only incoming AND-links of f.

Intuitively, if a parent folder f', which is OR-linked to the folder f, has a copy of a frame instance ω, and the frame instance ω satisfies the local predicate δ_f of the folder f, then the frame instance ω can be filed into the folder f. If all the parent folders f_i $(1 \leq i \leq n)$, which are AND-linked to the folder f, have a copy of a frame instance ω, and if the frame instance ω satisfies the local predicate δ_f of the folder f, then the frame instance ω can be filed into the folder f. The OR-links and AND-links in a folder organization define the semantics of the folder organization, and determine the document filing process. More specificly, the filing process is carried on from the root folder down to the leaf folders. The root folder contains all the frame instances that satisfy its local predicate. In order for a frame instance ω to be deposited into a non-root folder f, it must satisfies two conditions. Firstly, the frame instance ω must appear in one of the OR-lined parent folders of the folder f, or it must appear in **all** of the AND-linked parent folders of folder f. Secondly, the frame instance ω must satisfies the local predicate of the folder f.

From users point of view, there should be a criterion for each folder that governs the content of the folder. This criterion is called the global predicate of the folder. Theoretically, any frame instance ω is deposited in the folder f if and only if it satisfies the global predicate of the folder f. However, as mentioned before, for efficiency, the filing process is done by examing the local predicates along the filing paths rather than the global predicates of the folders. According to the definition 5.2 and 5.3, the global predicates of the folders in the folder organization can be generated recursively.

Definition 5.4 *(Global Predicate)* The global predicate of a folder is the predicate that governs the content of the folder. Let P_f denote the global predicate of any folder f. Using the definition 5.2 and 5.3, global predicates are given as follows:

1. The global predicate of the root folder is its local predicate.
2. Let $f_1, f_2, \cdots, f_n, f'_1, f'_2, \cdots, f'_m$ be all the parent folders of a folder f where (f_i, f) $(1 \leq i \leq n)$ is OR-link and (f'_i, f) $(1 \leq i \leq m)$ is AND-link. Then

$$P_f = \delta_f \wedge (\sum_{i=1}^{n} P_{f_i} \vee \prod_{j=1}^{m} P_{f'_j})$$

Theorem 5.1 *A frame instance ω is filed into folder a f if and only if ω satisfies the global predicate of the folder f.*

Proof: By induction.

1. Clearly, all of the frame instances in the root folder must satisfy its local predicate. Since the local predicate of the root folder is also its global predicate, the theorem holds for the root folder.
2. Suppose that the theorem holds for all parent folders of a folder f. Let ω be any frame instance. Let $f_1, f_2, \cdots, f_n, f'_1, f'_2, \cdots, f'_m$ be all the parent folders of f where (f_i, f) $(1 \leq i \leq n)$ is an OR-link and (f'_i, f) $(1 \leq i \leq m)$ is an AND-link. By the definition 5.2 and 5.3, ω is filed into f if and only if

$$(\delta_f(\omega) \wedge (\omega \in f_1)) \vee (\delta_f(\omega) \wedge (\omega \in f_2)) \vee \cdots \vee (\delta_f(\omega) \wedge (\omega \in f_n))$$

$$\vee(\delta_f(\omega) \wedge (\omega \in f'_i \mid 1 \leq i \leq m))$$

$$\Leftrightarrow (\delta_f(\omega) \wedge (P_{f_1}(\omega) \vee P_{f_2}(\omega) \vee \cdots \vee P_{f_n}(\omega) \vee (P_{f'_i}(\omega) \mid (1 \leq i \leq m))))$$

$\Leftrightarrow \omega$ satisfies $\delta_f \wedge (\sum_{i=1}^{n} P_{f_i} \vee \prod_{j=1}^{m} P_{f'_j})$. So the theorem holds for folder f.

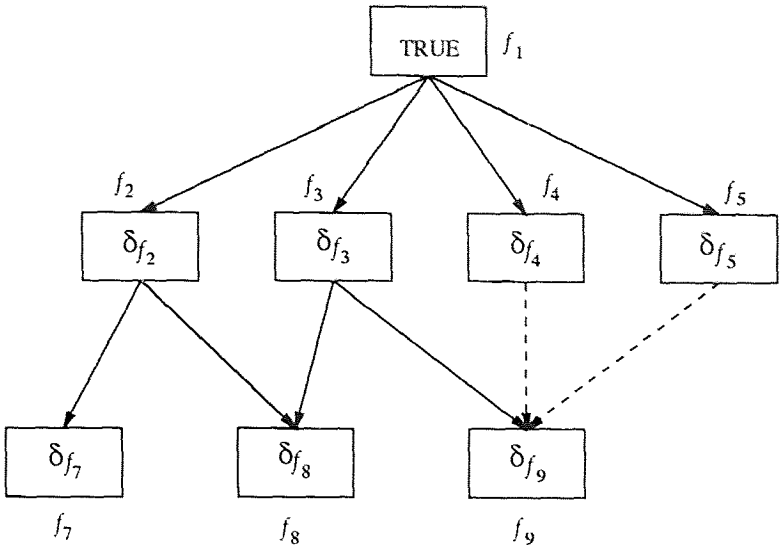


Figure 5.1 A folder organization

3. Since a folder organization is a rooted, directed acyclic graph, by induction, the theorem holds for any folder in the folder organization.

Example 5.1 Figure 5.1 depicts a folder organization, in which each folder f_i, $1 \leq i \leq 9$, has an associated local predicate δ_{f_i} and the local predicate δ_{f_1} for the root folder f_1 is TRUE. In the organization, the links (f_4, f_9) and (f_5, f_9) (in dashed lines) are of AND-link type, and the rest of the links (in solid lines) are of OR-link type. The global predicate of each folder is as follows: $P_{f_1} = TRUE$; $P_{f_j} = \delta_{f_j} \wedge P_{f_1} = \delta_{f_j} \wedge TRUE = \delta_{f_j}, 2 \leq j \leq 5$; $P_{f_7} = \delta_{f_7} \wedge P_{f_2} = \delta_{f_7} \wedge \delta_{f_2}$; $P_{f_8} = \delta_{f_8} \wedge (P_{f_2} \vee P_{f_3}) = \delta_{f_8} \wedge (\delta_{f_2} \vee \delta_{f_3})$; $P_{f_9} = \delta_{f_9} \wedge (P_{f_3} \vee (P_{f_4} \wedge P_{f_5})) = \delta_{f_9} \wedge (\delta_{f_3} \vee (\delta_{f_4} \wedge \delta_{f_5}))$.

5.2 Justification of the New Folder Organization Specification

In [77], a filing path from folder f_i to folder f_j in a folder organization is just a path from f_i to f_j. Each filing path q of a folder f has an associated predicate equal to $\prod_{v \in V(q)} \delta_v$. The global predicate P_f for each folder f can be represented as:

$$P_f = \sum_{q \in paths(f)} \left(\prod_{v \in V(q)} \delta_v \right)$$

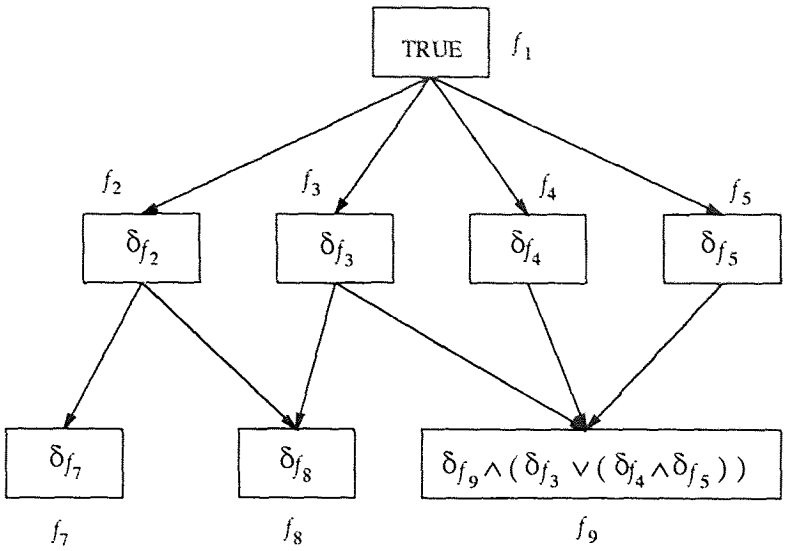


Figure 5.2 An alternative folder organization of Figure 5.1

where $paths(f)$ is a set of filing paths from the root folder to f, and δ_v is the local predicate of the folder v. Without AND-links, the user sometimes has to specify a complicated local predicate for a folder in order to describe the relationships between the folder and its parents. An example is given later.

Definition 5.5 *(Content Equivalent)* Two folder organizations $\mathcal{FO}_1$ and $\mathcal{FO}_2$ are said to be *content equivalent* if and only if there exist a one-to-one mapping g from the set of folders in $\mathcal{FO}_1$ to the set of folders in $\mathcal{FO}_2$ such that for any folder f in $\mathcal{FO}_1$, f and the folder $g(f)$ have the same global predicate.

Example 5.2 Figure 5.2 is a folder organization without AND-link, which is content-equivalent to the folder organization in Figure 5.1. However, the local predicate of the folder f_9 is $\delta_{f_9} \wedge (\delta_{f_3} \vee (\delta_{f_4} \wedge \delta_{f_5}))$.

The variable declarants (see definition 4.7) attached to links are provided for sophisticated users for specifying advanced folder organizations. Figure 5.3 shows two content equivalent folder organizations. The folder ROOT contains all the articles. The folder CIS contains all the articles authored by persons in the CIS department. The folder Fac contains the articles written by faculty members of CIS department.

The folder Student contains all the articles by students. And the folder Coauthored contains the articles co-authored by at least one faculty member and at least one student. Variable declarants are used in Figure 5.3(a). They simplify the local predicate of the folder Coauthored.

We conclude this section by giving a folder organization for NJIT in Figure 5.4. The folder organization is rooted with the folder ROOT which contains all the frame instances. Folder NJIT stores frame instances whose owners (owner is an abstract attribute which represents the attribute author of article type, the attribute sender or receiver of letter and memo types, etc.) are affiliated to NJIT. It is divided into subfolders based on different groups of owners of documents. For example, the folder CISFac holds the frame instances whose owners are faculty members of the CIS department at NJIT. The folder CISStudent contains the frame instances which are owned by CIS students. The folder CISRA holds the frame instances whose owners are CIS RAs (Research Assistant). The folder CISPhD contains the frame instances whose owners are CIS Ph.D. students. The frame instances which are owned by CIS master students are stored in folder the CISMS. The folder PUBLICATION contains all the articles authored by either a faculty member of the CIS department or a CIS Ph.D. student who is also a research assistant in the CIS department.

5.3 Falsedrop

A falsedrop is a filing of a frame instance into a folder to which it should not belong. Since the folder organization is defined by the user, whether a filing is a falsedrop depends on what the user thinks. So falsedrop is a relative concept. In other words, as long as the result of a fling is exactly what the user thinks it should be, it is fine. Otherwise, it is a falsedrop. From the users' point of view, the content of a folder is determined by its global predicate. So a filing of a frame instance into a folder is

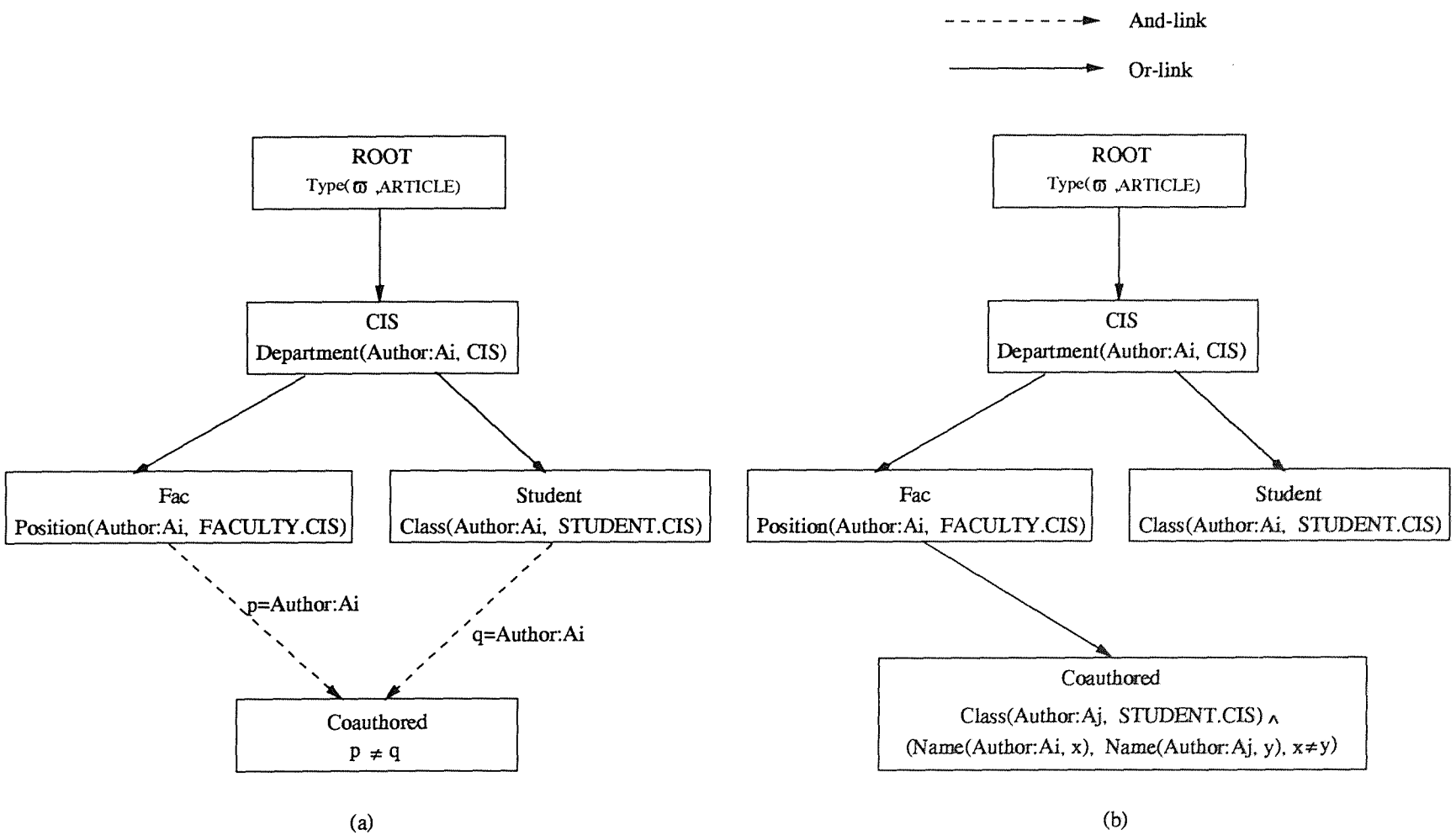


Figure 5.3 Folder organization with labeled links

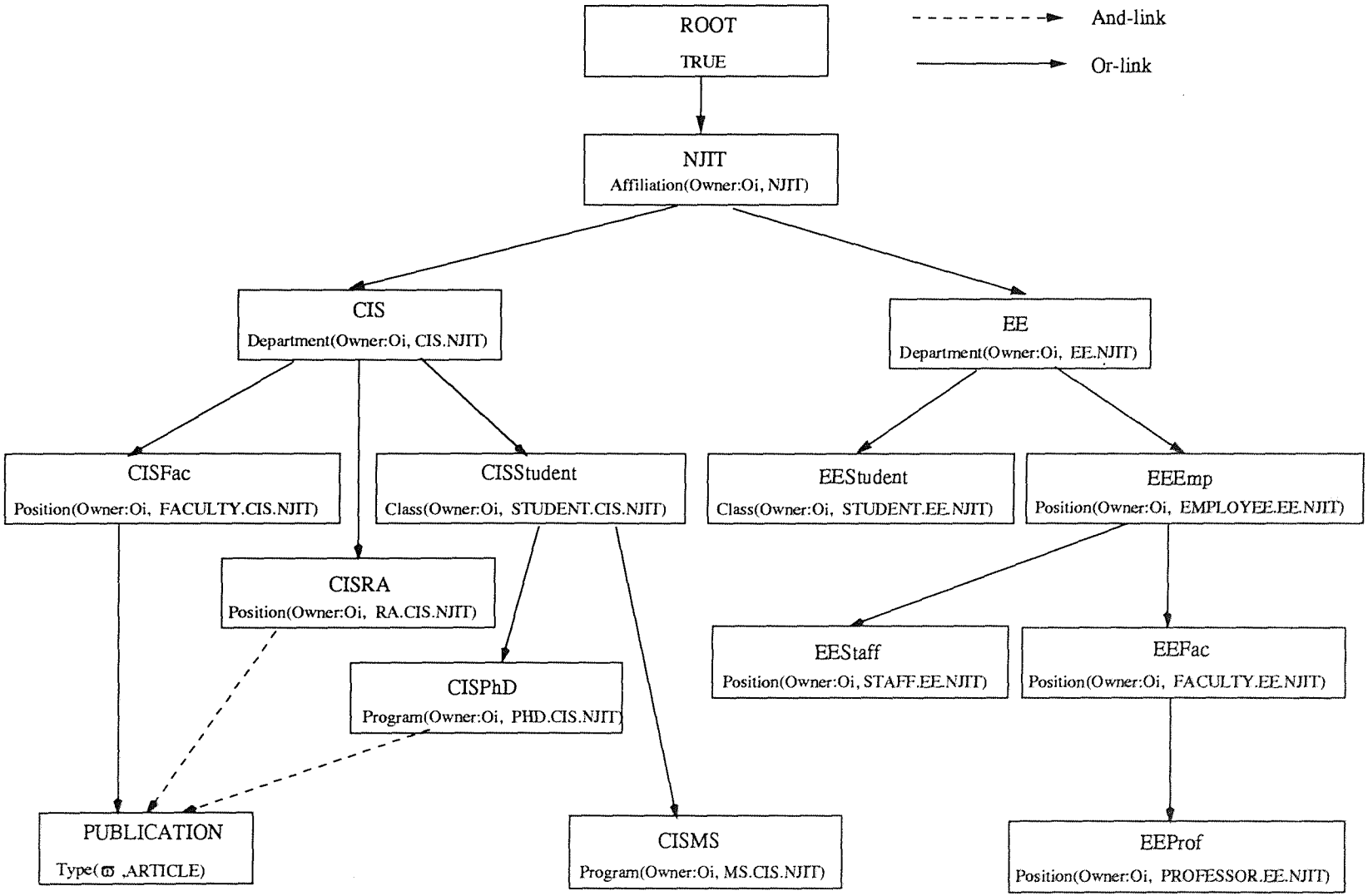


Figure 5.4 A folder organization for NJIT

a falsedrop if and only if the frame instance does not satisfy the global predicate of that folder. There are two types of falsedrops.

5.3.1 Falsedrops Caused by Incorrect Evaluation

Falsedrops will happen if predicates are not correctly evaluated. For example, suppose John Smith is staff member in the EE department at NJIT according to the knowledge base, then the slide in Figure 2.1 will be deposited in to the folder EEStaff in Figure 5.4. However, from the user's point of view, this filing is a falsedrop because the user knows that John Smith is not a staff member in the EE department at NJIT. There are two possible reasons that may cause incorrect predicate evaluation. The first one is that the evaluation engine is not correct. This is solved by showing the inference rules are correct. The second one is due to the incorrect knowledge as in the above example. In order to solve this problem, whenever this kind of falsedrops happen, the system will update the knowledge base accordingly.

5.3.2 Technical Falsedrops

From the users' point of view, the content of a folder is determined by its global predicate. However, for efficiency, the filing system does the filing by examing each local predicate along the filing paths rather than examing the global predicate. So even if the predicate evaluation is correct, it is still possible that a frame instance may not satisfy the global predicate of a folder although it satisfies all local predicates along the filing path of that folder. This kind of falsedrops is called technical falsedrops. Examples can be found in Chapter 3, where the ambiguous predicate specification and folder specification caused the filing system and user to have different understanding on the folder organization.

Corollary 5.1 *The proposed folder organization and predicate specification prevent technical falsedrops.*

Proof: Given any filing of a frame instance ω into a folder f, then the global predicate of f is true from the system's point of view according to Theorem 5.1. Suppose the predicate evaluation engine is correct, then the frame instance ω satisfies the global predicate of f. Therefore, the filing cannot be a technical falsedrop.

CHAPTER 6

DOCUMENT REPOSITORY

Repositories can be characterized as storing and managing both data and metadata (i.e., the information about the structure of the data). Repositories must maintain an evolving set of representations of information, and support evolving structure of information and its metadata cleanly, e.g., without recompiling when new properties of data or new relationships are added. In this chapter, we shall present a multilevel repository architecture and describe the search and retrieval strategies which are applicable to large collections of frame instances of various document types, by taking into account the repository architecture.

6.1 Multilevel Repository Architecture

As shown in Figure 6.1, we employ a three level architecture of a document repository to store documents. At the first level, the storage contains original documents. A physical storage containing frame instances is at the second level. Analogous to the inverted indexing, each frame instance has a pointer to its corresponding original document, besides which, it contains the most relevant information of the document, in a precise and succinct manner, pertinent to the user. The third level is the folder organization. Each folder is a virtual repository for a set of frame instances, which is also called the logical storage for the frame instances. It is called a virtual repository because it only stores pointers to the frame instances at the second level.

We adopt this multilevel access structure as the system repository architecture to support direct access to documents which requires retrieving their corresponding frame instances through the use of specific information, such as attributes and document type, from the document type hierarchy, and the folders containing these frame instances.

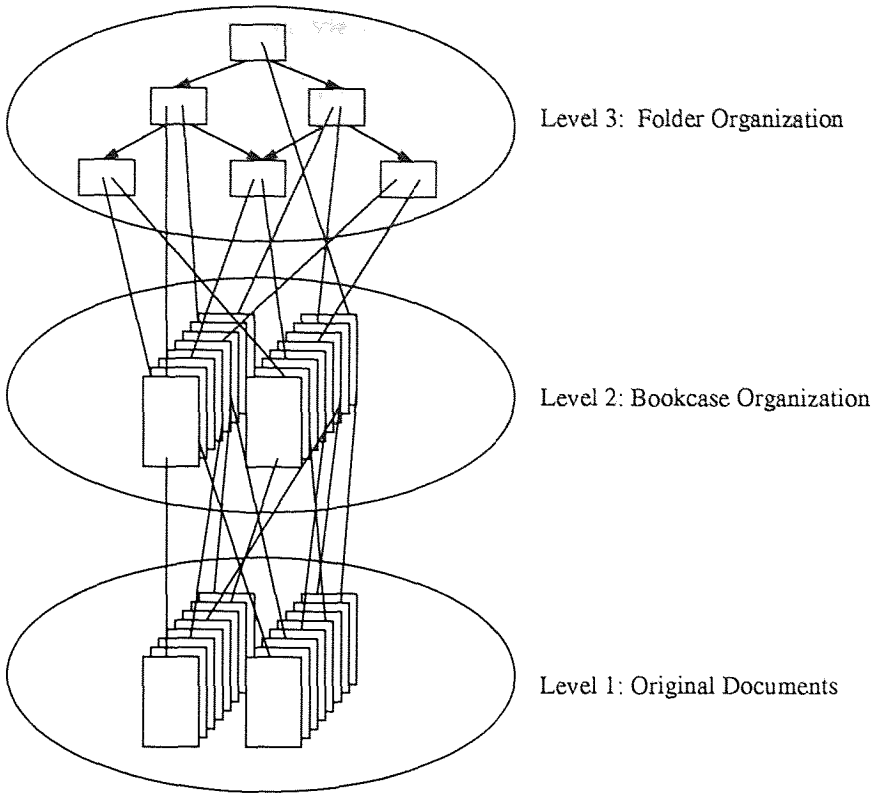


Figure 6.1 The system storage architecture

Folder organization sometimes allows rapid frame instance search and retrieval. Since all the related frame instances, which are collected in common groups (called folders) based on the user-predefined criteria for the folders, appear to be closely together, so that browsing in a collection of frame instances can be conducted effectively. In [65], an effective browsing technique is implemented for moving a given query toward the relevant items, such as frame instance type, frame instances of a document type, folders containing relevant frame instances of a document type, and so forth, using the well-known relevance-feedback process [26, 28]. Then document search can be further narrowed by examining all the frame instances that satisfy the predicates of folders and query formulation. The improvements in retrieval effectiveness are dependent on the effective use of the specific or related information in the system catalog and thesaurus.

From an implementation view point, we adopt the notion of encapsulation (which is most often achieved through information hiding) for designing the system repository: the implementation of the folder organization at the third level is independent from the physical storage of the frame instances at the second level, whose implementation is, in turn, independent from the physical storage of the original documents at the first level. Thus this leads to a clear separation of concerns. The folder organization does not add any requirements to the physical storage for frame instances at the second level. In other words, we benefit from the folder organization without losing any flexibility for organizing the storage at the second level.

6.2 The Bookcase Organization

On the second level, a bookcase organization is employed as the physical storage for frame instances. Frame instances are grouped into boxes. A box is a set of frame instances which are of the same document type. Each frame template (i.e., a document type) can have a bookcase which consists of several boxes. Frame instances of the type are stored in different boxes. For example, as shown in Figure 6.2, letters and memorandums are stored in boxes based on the attribute Date. Journal articles (say, ACM Transactions on Information Systems (TOIS)) can be organized based on a pair of attributes, volume and number of the issue where the articles appear. The pointer of a frame instance ω has the form BoxName:Offset, where BoxName is the name of the box, in which ω is located, and Offset is the offset of ω in the box BoxName. The box and bookcase are formally defined as follows.

Definition 6.1 *(Box)* *Box* is a set of four-tuple, $\mathcal{BX} = (\mathcal{K}, \mathcal{B}, \mathcal{P}, \mathcal{S})$, where:

1. $\mathcal{K}$ is an attribute which has an ordering defined.

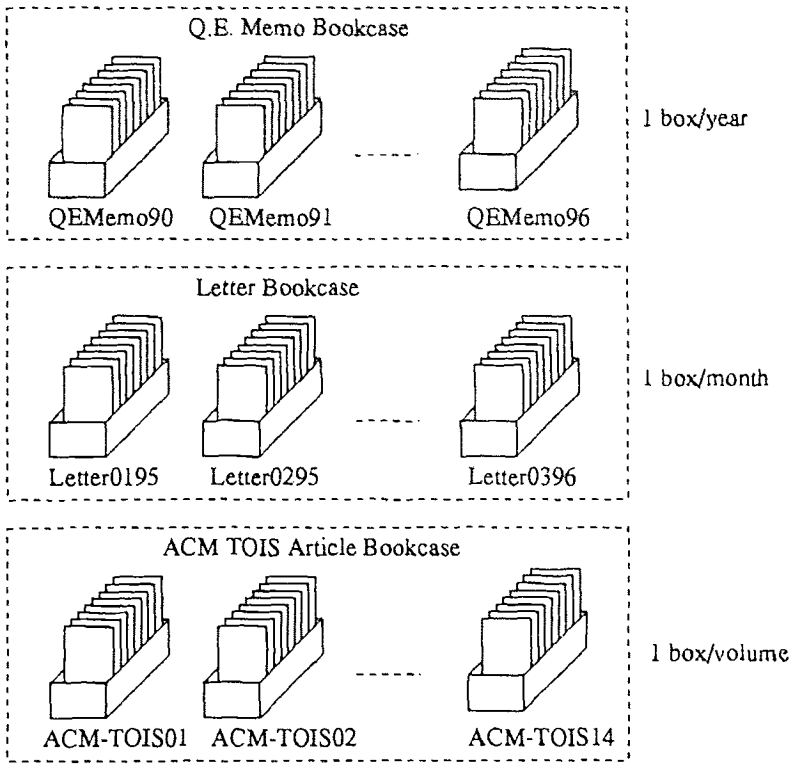


Figure 6.2 A sample bookcase organization

2. $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$, where $\mathcal{B}_1 \leq \mathcal{B}_2$, is the range of the values of the attribute $\mathcal{K}$ in all of the frame instances in the box.

3. $\mathcal{P}$ stores the common features of all of the frame instances in the box, specified as a predicate.

4. $\mathcal{S}$ is a set of frame instances.

Given a box $\mathcal{BX}$, $\mathcal{K}(\mathcal{BX})$ and $\mathcal{B}(\mathcal{BX})$ determines which frame instances should be stored in $\mathcal{BX}$. $\mathcal{P}(\mathcal{BX})$ is used to store the common features of all of the frame instances in $\mathcal{BX}$, and can be initialized as $(\mathcal{K}(\mathcal{BX})(\omega, \mathrm{x}), \mathcal{B}_1(\mathcal{BX}) \leq \mathrm{x} \leq \mathcal{B}_2(\mathcal{BX})))$. For example, let *Lecture97* be a box in bookcase *Lecture* storing all lectures which are produced in 1997. Then it can be defined as *Lecture97* = (Date, (01/01/97, 12/31/97), $\mathcal{P}$, $\mathcal{S}$), where $\mathcal{P}$ is initialized as (Date(ω, x), x$\geq$01/01/96, x$\leq$12/31/97).

Definition 6.2 *(Bookcase) Bookcase* is a set of three-tuple, $\mathcal{BC} = (\mathcal{T}, \mathcal{P}, \mathcal{C})$, where

1. All of the frame instances in the same bookcase have the same document type $\mathcal{T}$.

2. $\mathcal{P}$ is the common features of all frame instances in the bookcase, specified as a predicate.

3. $\mathcal{C}$ is a set of boxes.

Given a bookcase $\mathcal{BC}$, $\mathcal{T}(\mathcal{BC})$ defines the document type assigned to this bookcase. It determines the content of the bookcase. $\mathcal{P}(\mathcal{BC})$ is used to specify the common features of all of the frame instances in $\mathcal{BX}$, and can be initialized as Type(ω, $\mathcal{T}(\mathcal{BC})$). For example, if a bookcase is used to store documents of Lecture type, then it can be defined as *Lecture* = (Lecture, $\mathcal{P}$, $\mathcal{C}$), where $\mathcal{P}$ is initialized as Type(ω, Lecture). It should be noted that our future work will show how both $\mathcal{P}$(*Lecture97*) and $\mathcal{P}$(*Lecture*) can be enriched by the knowledge discovery and data mining. These common features are very userful in narrowing down the search space without accessing the contents of the documents.

6.3 Multilevel Retrieval Strategy

In this section, we shall discuss the document and information search and retrieval strategies using the multi-level repository architecture. Document and information retrieval is predicate-driven (i.e., queries are given in terms of predicates). The multilevel system storage architecture supports a multilevel retrieval strategy. Specifically, a knowledge-based query preprocessing is applied first to reduce the search space to a small subset of frame instances. Then the documents or information are found from the reduced search base. This retrieval process can be conceptual and content-based, or knowledge-based. The knowledge-based query preprocessing

requires the support of the folder organization at the third level, the bookcase organization at the second level, and the knowledge base which will be introduced in next chapter. The bookcase organization at the second level storage supports most of the text-based IR techniques for conducting content-based retrieval on multimedia documents. The first level storage will support various real content-based information retrieval on multimedia documents.

6.3.1 Knowledge-Based Query Preprocessing

The query preprocessing is knowledge-based and predicate driven. The goal of this preprocessing is to reduce the search space using the information contained in the query. Since all of the related frame instances, which are collected in groups (called folders) based on the user-predefined criteria for the folders, appear to be closely together, browsing in a collection of frame instances can be conducted effectively. The frame instance collection search strategy uses information obtained from the folder organization and the document type hierarchy. Using the folder organization and the document type hierarchy, the search space can be reduced to a particular folder f and frame instances of a particular type (i.e., a frame template T), respectively. The former case can be done by identifying folders whose criteria can be derived from the query formulation. For example, in Figure 5.4, to search for a memo which is written to Jennifer Wallace by John Smith regarding the result of her qualifying examination, only Folders JOHN, CISPhD, SpecialDoc and PhDQ.E.Memo will be the targets to be searched first. For the latter case, the document type of the frame instances will be identified using specific information contained in the system catalog [31, 32, 33]. This search can also be accomplished effectively and efficiently using a browsing mechanism [31, 65]. Normally, selective access is desired to particular frame instances of a document type on demand, and frame instance access may be considerably simplified when browsing capabilities allowing a flexible traversal of the document

type hierarchy, the folder organization and the text structure of frame instances are made available. Traditionally, textual database allows any search through their contents (words, phrases, etc.) or their structures (e.g., by navigating through a table of contents), but not both at the same time. By mixing the contents and structures in queries, TEXPROS allows us to pose very powerful queries, being much more expressive than each mechanism by itself. The system storage architecture can incorporate the dual models which means that the search space can be further reduced to $T \cap f$, focusing on frame instances of a particular type within a particular folder.

However, the usefulness of narrowing searching space from f to $f \cap T$ depends on the efficient way for generating $f \cap T$. In other words, once the folder f is identified, we could do the search on f. The complexity would be O(n*m), where n is the number of frame instances in the folder f and m is the average size of the frame instances in the folder f. If we cannot generate the set $f \cap T$ faster than O(n*m), it would not be worthy to reduce the searching space from f to $f \cap T$.

Using this three-level repository architecture, it is possible that the folder organization and the document type hierarchy can cooperate with each other in order to further speed up query processing by narrowing the searching space to $T \cap f$, assuming that a document type T and a folder f have been located. At the level of the logical storage for frame instances, each folder in a folder organization contains pointers to the frame instances. The pointer tells in which bookcase and box the frame instance can be found at the second level storage. So there is no need to examine each frame instance in the folder f and check whether it is of the document type T. Instead, the information about which frame instance in the folder f is of a particular document type T can be obtained simply by looking at its pointer. Furthermore, the pointer can also lead to the fact whether the frame instance has some properties using the common property of the box (bookcase) $\mathcal{P}(\mathcal{BX})$ $(\mathcal{P}(\mathcal{BC}))$.

So a box B, which contains the relevant documents, can be identified by examining if $\mathcal{P}(\mathcal{B})$ can be derived from the query formulation. Then the searching space would be reduced to $f \cap B$ instead of $f \cap T$. For this repository architecture, the $\cap$ operation, such as $T \cap f$, can be done in $O(n)$, where n is the number of frame instances in f. Since we are examining the pointers of frame instances, not their contents, and the size of a pointer of a frame instance is much less than its content, the complexity of $\cap$ operation (such as $T \cap f$) is much smaller than the one of searching frame instances in f which is O(n*m).

Let e be a query formula. The following query preprocessing algorithm, preProcess(e), generates a subset of frame instances. The content-based search will be done based on this subset.

1. Identify the document type T and then the bookcase $\mathcal{BC} = \{\mathcal{T}, \mathcal{P}, \mathcal{C}\}$ for the document type T.
2. Transform the query e, which is a predicate, into disjunctive normal form.
3. For each conjunctive element e_i, do the following:
   (a) Call findFolder(e_i) to find the smallest folder f whose global predicate can be derived from the conjunctive formula e_i.
   (b) Generate $\mathcal{B}_i = \{\mathcal{BX} \in \mathcal{C} : e_i \Rightarrow \mathcal{P}(\mathcal{BX})\}$.
   (c) Generate a set of frame instance
   $$\mathcal{S}_i = f \bigcap (\bigcup_{\mathcal{X} \in \mathcal{B}_i} \mathcal{X}).$$
4. Generate a set of frame instances $S = \bigcup S_i$.

The algorithm findFolder(e_i) is given as follows:

1. Place the root of the folder organization into a stack called OPEN.
2. While OPEN is not empty, do the following:

(a) Pop out a folder f from OPEN and visit it.

(b) Push all the child folders of f whose local predicates can be derived from e_i into OPEN.

(c) Record the smallest folder (the one contains the smallest number of frame instances) ever visited as F.

3. Return F.

6.3.1.1 Performance Analysis: The complexity of findFolder() depends on the number of folders that are visited by the program. In most cases, a leaf folder and the folders along one of the filing path will be visited. In the worst case, all folders may have to be visited. Let k be the number of folders in the folder organization. To determine if the local predicate of a folder can be derived from e_i in step 2(b), the predicate evaluation engine will be invoked. Let d be the average time needed by the predicate evaluation engine for evaluating a predicate. The complexity of findFolder() is $O(d \times \log k)$ on average and $O(d \times k)$ in the worst case. The complexity of step 3(b) is $O(d \times t)$ where t is the number of boxes in the bookcase $\mathcal{BC}$. As we have discussed early, the step 3(c) needs $O(n)$ time, where n is the number of frame instances in the folder f. The step 3 has s iterations, where s is the number of conjunction elements in the query formula e. So the complexity of preProcess() is $O(s \times (d \times \log k + d \times t + n))$ on average and $O(s \times (d \times k + d \times t + n))$ in the worst case. As we will see in the next chapter, d can be considered as a constant number. Assume that s and t are insignificant, then the complexity of preProcess() will be $O(\log k + n)$ on average and $O(k + n)$ in the worst case.

6.3.2 Frame Instance Search

The document and information retrieval is applied on the frame instances in the search base which is generated by the query preprocessor preProcess(). The

retrieval can be content-based or knowledge-based. Since the frame instances contain conceptual information of their original documents, the content-based frame instance retrieval can also be considered as conceptual-based retrieval of the original documents.

Due to the large volume of multimedia documents and lack of semantic content based access for some media type such as image and video, the usefulness and efficiency of a multimedia document processing system can often be improved greatly by transforming the original document into a formal structure of the document type, which can be stored and manipulated easily. For our case, frame instances are the formal structures of their corresponding documents. They reduce the size considerably, and therefore the storage space and processing time are saved in using short texts. Text retrieval operations on the frame instances, which are stored at the storage at the second level, depend directly on the content representations, such as content identifiers, which are used to describe the stored document contents. A substantial effort [20, 21, 32] was devoted to analyzing the content of the stored documents, dealing with the generation of the content identifiers, and comparing query formulations and the content representations, including document descriptors. The retrieval of frame instances depends partially on an exact match between the values of the content identifiers (i.e., the structured part of frame instances) and the attribute values used in the query formulations. Often, information retrieval decisions may depend on the contents of the unstructured part of frame instances. However, the text size of the contents of the unstructured part of frame instances are considerably smaller in comparison with the contents of the corresponding original documents. For this case, various classical text retrieval methods, such as inverted indexing, KMP string matching [1, 29], clustering searching, etc. are applicable. And the document search and retrieval can be done in an incremental way by examining a small set of frame instances of a particular type within a folder. For

example, suppose Professor John Smith in CIS department wants to retrieve his memo to Jennifer Wallace regarding the result of her qualifying examination taken recently. And suppose there is a folder called PhDQ.E.Memo which contains all the memos regarding CIS Ph.D. student qualifying examination. He may first search the PhDQ.E.Memo folder because it is the most likely folder where the memo can be found. And since Professor John Smith knows that the memo was sent by him, he might look in CISFac next. And since Jennifer Wallace is a Ph.D. student in the CIS department, then CISPhd folder might be the next one to be searched. If we fail to identify the frame instance of the corresponding memo after searching through the folders PhDQ.E.Memo, CISFac and CISPhd, then we can conclude that possibly the memo is not in the system.

Frame instances are synopsis of original documents and are plain text. The second level storage not only provides a platform for various text-based information retrieval techniques, but also makes it possible to use text-based IR techniques for providing content based multimedia information retrieval. This is important because today's office documents are multimedia documents, and the automatic semantic interpration of some media types such as image and video is far from applicable given the state-of-the-art of computer intelligence.

The knowledge base, used for evaluating whether a frame instance satisfies a predicate, supports knowledge-based document retrieval. The user is allowed to specify a query containing second level predicate clauses. With the support of the knowledge base which will be introduced in the next chapter, the evaluation engine can determine which frame instances in the search base satisfy the query.

6.3.3 Content-Based Retrieval on Original Documents

It is possible that the document retrieval may have to go through the contents of original documents although this is what we are trying to avoid. This could happen

because a frame instance contains only the synopsis of its corresponding original document. Although the real content based retrieval on multimedia documents is still far from applicable, much research has been focusing on this area. The system has the capability to support various potential content-based multimedia information retrieval techniques.

CHAPTER 7

KNOWLEDGE-BASED PREDICATE EVALUATION

In chapter 4, we formalized predicates for specifying folder criteria and queries. In this chapter, we shall introduce an evaluation engine which is used to determine whether a frame instance satisfies a predicate. The predicate evaluation is knowledge-based [17]. This chapter will define the structure for representing knowledge. For supporting dynamic and flexible modeling, we shall present a learning agent for acquiring the needed knowledge based on the user-defined folder organization.

7.1 The Predicate Evaluation Engine

As shown in Figure 7.1, an evaluation engine consists of a control module, two evaluation modules, an object base, a domain knowledge base and an inference engine. The control module parses and divides the inputed predicate into predicate clauses and constraints. It controls the other modules of the evaluation engine and makes the final conclusion based on the outputs of them. Since only variables, values and a set of operators are used to specify predicate constraints, the process of evaluating predicate constraints is straightforward. The process of evaluating the predicate clauses has three phases. In the first phase, it involves the evaluation module I to evaluate the first level predicate clauses, which specify some characteristics of frame instances. In the second phase, the evaluation module II evaluates the second level predicate clauses. The system catalog, which contains the document type hierarchy and the thesaurus, is needed by both of the evaluation modules for interpreting the frame instance. Any second level predicate clauses which cannot be successfully evaluated by the evaluation module II goes to the third phase, in which the inference engine makes further evaluation using the knowledge base. In

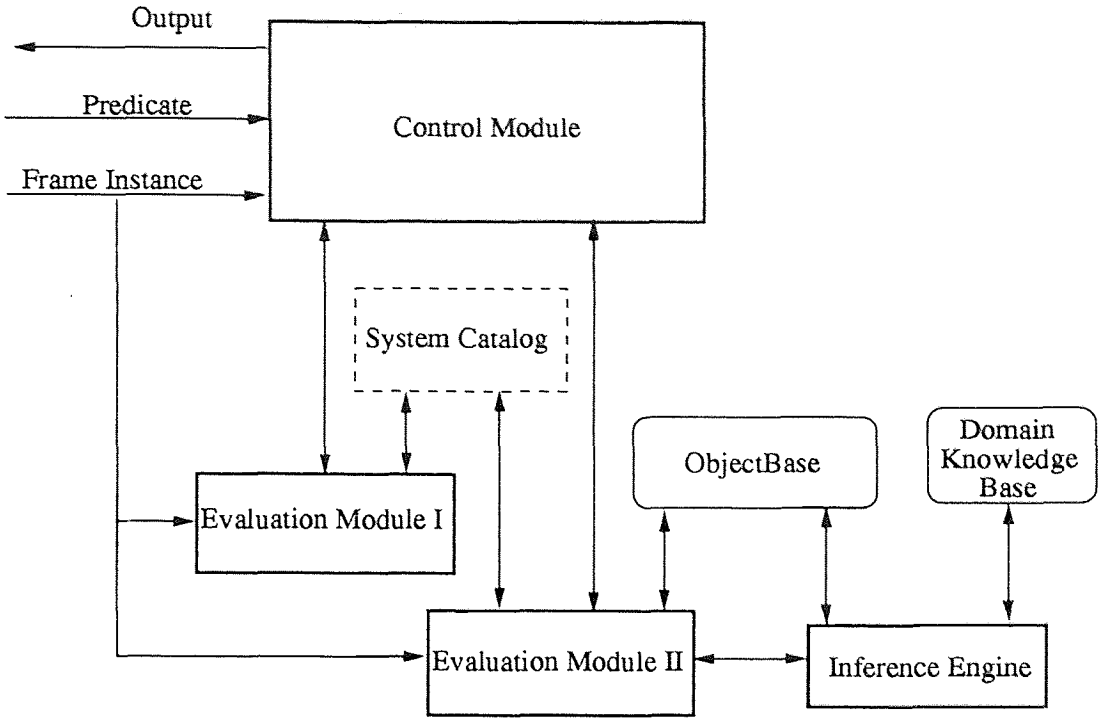


Figure 7.1 The architecture of the evaluation engine

the remainder of this chapter, we shall discuss the components of the evaluation engine.

7.1.1 Evaluation Module I

The evaluation module I evaluates the first level predicate clauses $g(\omega, b[, r])$ by determining whether the attribute g has a value b in the frame instance ω, or by assigning the value of the attribute g appearing in the frame instance ω to the variable b, based on the pattern r. For example, Date(ω, 4/25/96) can be evaluated to be true by matching this predicate clause against the attribute-value pair (Date, 4/25/96) appearing in the frame instance ω. For the other case, ContractID(ω, x, *??*-), the module I will assign the first two characters of the value of attribute ContractID in the frame instance ω to the variable x.

7.1.2 Evaluation Module II

Evaluation engine II evaluates second level predicate clauses which specify high level conceptual information of documents. Basically, there are two kinds of predicate clauses sent to this module for evaluation. Both goal predicate clauses and assignment predicate clauses specify the properties of the objects, which are the first parameters of the predicate clauses. For example, Program(Sender, PhDStudent.CIS) is a goal predicate stating that the sender is a PhD student of the CIS department. An assignment predicate Age(Sender, x) will assign the age of the Sender to the variable x. The process of evaluating these predicate clauses requires additional knowledge about the involved objects. This knowledge is not contained in the documents. An object base for the involved objects is maintained to store the knowledge about these objects.

7.1.2.1 Object Base: The object base is used to maintain the knowledge or facts about the objects which are involved in specifying second level predicates of the folder organization. It consists of a set of object pages. Each object page is associated with one object. The object base is domain dependent, which means that the object base contains different knowledge in different domains. It would be impossible to encapsulate all the knowledge about one object without giving a specific domain. Given an application domain, the object base only maintains the needed knowledge of objects within the domain. Knowledge or facts about one particular object is encapsulated into one object page in the terms of (attribute, values) pairs. Attributes in an object page are called *property names*. They can be multiple valued. Each property name, together with one of its value, defines a *property* of the object. Figure 7.2 shows an object page for John Smith, which is self-explanatory.

Name	"John Smith"
Affiliation	NJIT
Position	FACULTY.CIS.NJIT
Program	PhD.EE.NJIT
Department	
Class	STUDENT.EE.NJIT

Figure 7.2 Am sample object page

7.1.2.2 Evaluation of Second Level Predicates: Given a goal predicate clause $g(a, b)$, if (g, b) has an exact match with a property in the object page for the object a, then we can conclude that the goal predicate clause is true. For example, the predicate clause Position(John Smith, FACULTY.CIS.NJIT) is true because it has an exact match with the property (Position, FACULTY.CIS.NJIT) in the object page of John Smith. For an assignment predicate clause, $g(a, b[, r])$, the module will assign the value of attribute g, which is a property name of the object page for the object a, to the variable b. For example, the assignment predicate clause Affiliation(John Smith, x) yields x = NJIT, which is dependent upon the availability of the (attribute:Affiliation, value:NJIT) pair in the object page of John Smith and the exact match between the name of the predicate clause and an attribute from the object page.

If no property is matched, the predicate clause will be sent to the inference engine for further evaluation assisted by other mechanisms. For example, the module fails to evaluate the predicate clause Position(John Smith, EMPLOYEE.CIS),

because there is no EMPLOYEE.CIS associated with the property Position in the object page for John Smith.

7.2 Knowledge Base

The knowledge base consists of the object base and the domain knowledge base. The domain knowledge base contains the knowledge of the application domain. We also identified some rules which tell how the knowledge in domain knowledge base and object base can be used for evaluating predicates. A domain (say, NJIT Affiliation) may consist of subdomains (such as, College and School), and each of the subdomains may have, in turn, subdomains (such as, College has subdomains Department and Office, and School has a subdomain Division). Each of the subdomains has various properties of interest for describing objects. For example, the subdomain Department of NJIT may have properties (Class, STUDENT), (Position, EMPLOYEE), (Program, PHD), (Program, MSStudent), (Position, FACULTY), and (Position, STAFF). A property is called *unique* if it appears only in one subdomain. The knowledge about the subdomains that an application domain has and their relationships is stored in domain organization. The property relation is used to store the knowledge about what properties of interest each subdomain has and what are their relationships.

Definition 7.1 *(Domain Organization)* Given an application domain D_0, the *domain organization* of D_0, $\mathcal{DO}(D_0) = D_0(V, E)$, is a directed tree where:

1. Each vertex in $V(D_0)$ (also denoted as $V(\mathcal{DO})$) corresponds to a domain.

2. Each edge (D_i, D_j) in $E(D_0)$ denotes that D_j is a subdomain of D_i.

Definition 7.2 *(Property Relation)* Given a domain D_i in a domain organization $\mathcal{DO}(D_0)$, the *property relation* of D_i, $\mathcal{PR}_{\mathcal{DO}(D_0)}(D_i) = D_i(N, L)$, is a directed tree where:

1. D_i is the root of the tree.

2. Each non-root vertex in N corresponds to a property.

3. Each directed solid edge (P_m, P_n) in L, connecting two properties, denotes that the property P_n implies the property P_m.

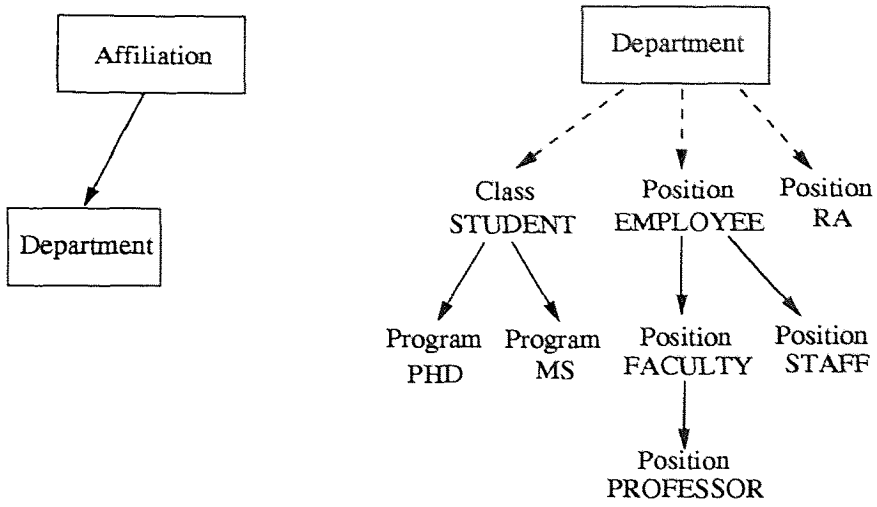
4. Each directed, dashed edge (D_i, P_j) in L connects the root D_i and a property P_j whose in-degree of the solid edges is zero.

Some of the subdomains, say D_k, of the domain organization of D_0 may not have the property relation; then $\mathcal{PR}_{\mathcal{DO}(D_0)}(D_k)$ is empty.

Definition 7.3 *(Domain Knowledge)* Given an application domain D_0, the *Domain knowledge* of D_0 is a two-tuple, $\mathcal{DK}(D_0) = \{\mathcal{DO}(D_0), \mathcal{PR}\}$, where $\mathcal{DO}(D_0)$ is the domain organization of D_0, and $\mathcal{PR} = \{\mathcal{PR}_{\mathcal{DO}(D_0)}(D) \mid (D \in V(D_0))\}$ is a set of property relations defined for the domains in the domain organization $\mathcal{DO}(D_0)$.

Thus, a domain knowledge base is composed of a collection of domain knowledge.

Example 7.1 For the sake of simplicity, Figure 7.3 depicts a simple domain knowledge of the domain Affiliation. Figure 7.3(a) shows the domain organization of the Affiliation (of which NJIT is an example). The domain Affiliation has a subdomain Department. Figure 7.3(b) shows the property relations of the domain Department. The domain Department has (Class, STUDENT), (Program, PHD), (Program, MS), (Position, EMPLOYEE), (Position, FACULTY), (Position, STAFF), (Position, PROFESSOR), and (Position, RA) (RA stands for Research Assistant) as its properties of interest. The property relation tells that all PHD and MS students are STUDENTs; PROFESSORs are FACULTY members; and FACULTY and STAFF are EMPLOYEEs of the Department (says, Department of CIS) at NJIT.



(a) Organization of Affiliation (NJIT) domain (b) Property relationships of domain Department

Figure 7.3 The domain knowledge of NJIT affiliation

Let $D_i \in V(\mathcal{DO})$ $(0 \leq i \leq n)$, be a domain in the domain organization $\mathcal{DO}(D_0)$. $D_n \cdot D_{n-1} \cdot \ldots \cdot D_0$ is called the *full name* of the domain D_n if and only if (D_0, D_1), (D_1, D_2), $\cdots$, (D_{n-1}, D_n) are the directed edges in $\mathcal{DO}(D_0)$ which means that D_i is the subdomain of D_{i+1}, for $0 \leq i \leq n-1$. Values of a domain are called *domain instances*. For example, CIS is a domain instance of the domain Department. Let $D_n \cdot D_{n-1} \cdot \ldots \cdot D_0$ be the full name of the domain D_n,and d_n, d_{n-1}, $\cdots$, d_0 be the domain instances of D_n, D_{n-1}, $\cdots$, D_0, respectively. Then $d_n \cdot d_{n-1} \cdot \ldots \cdot d_0$ is called the full name of domain instance d_n. The full name of a domain (or domain instance) is the precise description of the domain (or domain instance).

7.2.1 Rules for using the Knowledge Base

Let $\mathcal{DK}(\mathcal{DO}(D_0), \mathcal{PR})$ be the domain knowledge. Let D_0, D_1, $\cdots$, D_n be the domains, and d_0, d_1, $\cdots$, d_n be domain instances of D_0, D_1, $\cdots$, D_n, respectively. There are four rules for guilding the inference engine to evaluate predicates using the knowledge base:

- Rule 1: If $(P_t, t) \in \mathcal{PR}(D_n)$ is a unique property, then $P_t(x, t \cdot d_n \cdot \ldots \cdot d_i \cdot \ldots \cdot d_0) \Rightarrow P_t(x, t \cdot d_i \cdot \ldots \cdot d_0)$
- Rule 2: $D_1(x, d_n \cdot \ldots \cdot d_i \cdot \ldots \cdot d_0) \Rightarrow D_i(x, d_i \cdot \ldots \cdot d_0)$
- Rule 3: $P_t(x, t \cdot d_n \cdot \ldots \cdot d_i \cdot \ldots \cdot d_0) \Rightarrow D_i(x, d_i \cdot \ldots \cdot d_0)$
- Rule 4: For any properties (P_a, a) and (P_b, b) in property relation $\mathcal{PR}(D_n)$, if there is a path from (P_a, a) to (P_b, b), then $P_b(x, b \cdot d_n \cdot d_{n-1} \cdot \ldots \cdot d_0) \Rightarrow P_a(x, a \cdot d_n \cdot d_{n-1} \cdot \ldots \cdot d_0)$

The first rule claims that if an object x has a property (P_t, t) in the domain instances $d_n \cdot \ldots \cdot d_i \cdot \ldots \cdot d_0$, it also has the property (P_t, t) in a super domain instance $d_i \cdot \ldots \cdot d_0$ provided that (P_t, t) is a unique property. For example, if John Smith is a FACULTY member of the CIS department at NJIT, Position(John Smith, FACULTY.CIS.NJIT), then John Smith is also a FACULTY member of NJIT, Position(John Smith, FACULTY.NJIT).

The second rule states that if an object belongs to the domain instances $d_n \cdot \ldots \cdot d_i \cdot \ldots \cdot d_0$, it also belongs to a super domain instances $d_i \cdot \ldots \cdot d_0$. For example, if John Smith works for the CIS department, Department(John Smith, CIS.NJIT), then he is considered to be a member of NJIT. Affiliation(John Smith, NJIT).

The third rule states that if an object x has a property t in the domain instance $d_n \cdot \ldots \cdot d_i \cdot \ldots \cdot d_0$, then it belongs to a super domain instances $d_i \cdot \ldots \cdot d_0$. For example, if John Smith is a FACULTY member of the CIS department at NJIT, Position(John Smith, FACULTY.CIS.NJIT), then he is considered to be a member of NJIT Affiliation, Affiliation(John Smith, NJIT).

The fourth rule states that if an object x has a property (P_b, b) in the domain instance $d_n \cdot d_{n-1} \cdot \ldots \cdot d_0$ and if there is a path from (P_a, a) (non-root node) to (P_b, b) in $\mathcal{PR}(D_n)$, then the object x also has the property (P_a, a) in the domain instances $d_n \cdot d_{n-1} \cdot \ldots \cdot d_0$. For example, if John Smith is a FACULTY member of the

CIS department, Position(John Smith, FACULTY.CIS), and since there is a path from (Position, EMPLOYEE) to (Position, FACULTY) in $\mathcal{PR}$(Department), then John Smith is also an EMPLOYEE of the CIS Department, Position(John Smith, EMPLOYEE.CIS).

Theorem 7.1 *(Correctness of The Rule Base)* Rules 1-4 are valid.

Proof: Since the domain of D_{i+1} is a subdomain of D_i, $0 \leq i \leq n-1$, the domain $D_n \cdot D_{n-1} \cdot \ldots \cdot D_i \cdot \ldots \cdot D_0$ is a subdomain of $D_i \cdot \ldots \cdot D_0$. Therefore, Rule 2 is valid. Suppose an object x has property (P_t, t) in the domain instance $d_n \cdot d_{n-1} \cdot \ldots \cdot d_0$. If the property (P_t, t) is unique, then it is not a property of interest in the property relation $\mathcal{PR}(D_i)$, for $0 \leq i \leq n-1$. So it will not cause misunderstanding to say that the object x has the property (P_t, t) in the super domain instance $d_i \cdot \ldots \cdot d_0$.

For Rule 3, the left side of the rule claims that object x has property $(P_t,\ t)$ in the domain instance $d_n \cdot d_{n-1} \cdot \ldots \cdot d_i \cdot \ldots \cdot d_0$. So x belongs to domain instance $d_n \cdot d_{n-1} \cdot \ldots \cdot d_i \cdot \ldots \cdot d_0$. By Rule 2, $D_i(x, d_i \cdot \ldots \cdot d_0)$ is true.

For Rule 4, the left side of the rule claims that object x has property $(P_b,\ b)$ in the domain instance $d_n \cdot d_{n-1} \cdot \ldots \cdot d_0$. Assume that the path from $(P_a,\ a)$ to $(P_b,\ b)$ in $\mathcal{PR}(D_n)$ is $(P_a, a), (P_1, k_1), (P_2, k_2), \cdots, (P_m, k_m), (P_b, b)$. By Definition 7.2, object x also has property $(P_m,\ k_m)$ in the domain instance $d_n \cdot d_{n-1} \cdot \ldots \cdot d_0$. Therefore, $P_m(x,\ k_m \cdot d_n \cdot d_{n-1} \cdot \ldots \cdot d_0)$ is true. Similarly, $P_{m-1}(x,\ k_{m-1} \cdot d_n \cdot d_{n-1} \cdot \ldots \cdot d_0)$ is true. By induction, $P_a(x,\ a \cdot d_n \cdot d_{n-1} \cdot \ldots \cdot d_0)$ is true.

7.2.2 Justification of the Knowledge Base

The proposed knowledge base is used for evaluating whether a frame instance satisfies a predicate. The predicate specification has been modified from FOPL for describing facts about either frame instances or objects that are related to frame instances. The knowledge representation structure is designed based on the needs of the predicate specification. The object base is used for storing facts about objects. Each object

is described by one object page. With the object base, the evaluation of the second level predicate clauses can be done by matching properties in the object page. In order for the system to understand the object base, the domain in which the object base is defined is described with the domain knowledge base. It consists of a domain organization and a set of property relations. The domain organization specifies the hierarchy of the domain. Each subdomain has a set of properties of interest for describing objects. The domain knowledge base actually determines the structure of object pages. Traditional knowledge base system has a rule base to support inference [59]. However, the problem of efficiently picking up a rule to fire from a large rule base has not been solved yet. In this dissertation, the rule base has been simplified and embedded in the property relations which are tree structured. We identified four general rules for guiding the use of the knowledge base. The simplification is made based on the following considerations:

1. The negation operator is rarely involved in reasoning in document filing and retrieval. The knowledge base in this dissertation is used for support document filing and retrieval based on the properties of objects. A negative fact that an object does not have a particular property can be used as criterion for document filing and retrieval. But it is not appropriate to be used for reasoning because not knowing that an object has a particular property does not mean the corresponding predicate is false. It is better not to use it for deriving new facts. Therefore, negative fact is not used for reasoning in this dissertation. In other words, rules like $\neg A \rightarrow B$ are not supported.

2. Rules like $A \rightarrow B \vee C$ and $A \wedge B \rightarrow C$ are not supported. The removal of these kinds of rules simplifies the inference process significantly. The cost of doing so is to embed them in the knowledge acquisition process. For instance,

without the rule $A \wedge B \rightarrow C$, the learning agent has to learn whether C is true even though it knows that A and B are true.

After the above simplification, the rule base can be expressed by tree structures. Each node represents a property (or fact). Each edge represents a single rule. With the rule base represented by such tree structures, the inference process becomes finding a path from the goal to a known fact, which can be done very efficiently.

7.3 Knowledge Acquisition

The knowledge base is used for evaluating the second level predicate clauses. This section discusses how to acquire the needed knowledge if there are folders in the folder organization, whose criteria are specified in terms of the second level predicate clauses. A learning agent is developed, which is responsible for building the object base and acquiring the domain knowledge. This agent is a learning process over the lifetime of the system. And knowledge acquired in one application cycle can be reused in another cycle as long as the two cycles have the similar application domains.

7.3.1 Obtaining Domain Knowledge

As we discussed early, frame instances are organized as a user-defined folder organization. Different folder organization contains different domain knowledge. The domain knowledge base can be extracted from the folder organization since the it reflects users' knowledge of an application domain. The knowledge base contains only the knowledge about the involved objects, so the folders with first level predicate clauses as criteria will be filtered out from the folder organization during the knowledge acquisition.

Let $\mathcal{DO}$ be the domain organization being created. Let $D \in \mathcal{DO}$ denote that D is a domain, $d \lhd D$ denote that d is an instance of the domain D, $D_1 \supset D_2$ denote

that the domain D_2 is a subdomain of D_1. Let $(a, v) \in \mathcal{PR}(D)$ denote that (a, v) is a property in the domain D, and $(a, v) \Rightarrow (p, u)$ denote that the property (a, v) implies the property (p, u). Let $p(o, u) \rightarrow a(o, v)$ denote that $p(o, u)$ and $a(o, v)$, which are second level predicate clauses, are local predicates of folder f and g, respectively, where f is the only parent of g. The following learning rules are used by the learning agent to acquire the domain knowledge.

1. If $D(o, d) \rightarrow p(o, u.d)$, then $D \in \mathcal{PR}$ and $d \lhd D$.
2. If $D_1(o, d_1) \rightarrow D_2(o, d_2 \cdot d_1))$ and $D_2 \in \mathcal{DO}$, then $d_2 \cdot d_1 \lhd D_2$ and $D1 \supset D2$.
3. If $p(o, u \cdot d) \rightarrow a(o, v \cdot d))$ and $d \lhd D$, then $(a, v), (p, u) \in \mathcal{PR}(D)$ and $(a, v) \Rightarrow (p, u)$.
4. If $p(o, u.d)$ is the local predicate of a leaf folder and $d \lhd D$, then $(p, u) \in \mathcal{PR}(D)$.
5. If $(a, v), (p, u) \in \mathcal{PR}(D)$ and u is narrow term of v, then $(p, u) \Rightarrow (a, v)$.

Example 7.2 The snapshot of the domain knowledge base in Figure 7.3 is extracted from the folder organization in Figure 5.4. The first rule is applied on the NJIT folder and the CIS folder, which implies that Affiliation is a domain and NJIT is domain instance. When the same rule is applied on the CIS folder and the CISFac folder, the learning agent knows that Department is domain and CIS is a domain instance. Then the second rule can be applied which establishs the fact that the domain Department is a sub-domain of the domain Affiliation. The folder CISStudent and the folder CISMS fires the third rule which suggests that (Class, STUDENT) and (Program, MS) are properties of interest in the domain Department and the property (Program, MS) implies the property (Class, STUDENT). Similarly, (Program, PHD) is also a property of interest in the domain Department which implies the property (Class, STUDENT). The repeated firing of the third rule and the forth rule will finally generate the property relation of the domain Department.

7.3.2 Building Object Base

As we have discussed in the previous section, the domain knowledge can be extracted from the folder organization. But the knowledge about the objects that are involved in specifying predicates cannot be acquired directly from the folder organization. For collecting facts about these objects, the agent must know what, how and where to learn. We define topic as the basic concept of learning. It should be noted that the concept object used in this section is referred to the real object that appears in frame instances. It is not the object identifier used in the filing criteria for identifying the real objects in the frame instances.

Definition 7.4 *(Topic)* A *topic* is a four-tuple (o, a, v, d), where o is an object, (a, v) is a property, d is a domain instance.

A topic (o, a, v, d) is a *fact* if the predicate $a(o, v.d)$ is true.

7.3.2.1 Generating Learning Topic: A *learning topic* is a question which inquires any part of a topic. For example, $(?o, a, v, d)$ states who has property (a, v) in the domain instance d. $?(o, a, v, d)$ states if (o, a, v, d) is a fact. There are three rules for generating learning topics.

1. If $a(o, v.d)$ is a predicate, then $?(o, a, v, d)$ is a learning topic (also called YN learning topic).
2. If $d \lhd D$ and $(a, v) \in \mathcal{PR}(D)$, then $(?o, a, v, d)$ is a learning topic (also called WH learning topic) for each domain instance d of the domain D.
3. A learning topic $(?o, a, v, d)$ has higher priority than $(?o, p, u, d)$ if $d \lhd D$, and $((p, u), (a, v))$ is a link from (p, u) to (a, v) in $\mathcal{PR}(D)$.
4. A learning topic $(?o, a, v, d)$ has higher priority than $?(o, a, v, d)$.

7.3.2.2 Learning Rules: The learning agent uses the learning rules to collect facts, which are properties of objects in the object pages. The "yes" answer of a YN learning topic $?(o, a, v, d)$ generates the fact (o, a, v, d). There are three ways that the learning agent can get the answer of a YN learning topic $?(o, a, v, d)$. The first one is to translate the learning topic $?(o, a, v, d)$ to a predicate $a(o, v \cdot d)$ and launch the inference engine for examining whether the predicate is true. If the existing knowledge implies the fact (o, a, v, d), the learning agent will store it in the object base. Storing duplicate knowledge in the object base increases the efficiency of the predicate evaluation engine, because most of the second level predicate clauses can be evaluated by the evaluation module II without launching the inference engine. The second way is catching the events which imply that the predicate $a(o, v \cdot d)$ is true. For example, a filing of a frame instance into a folder is forced by the user, which implies that the local predicates of the folder along the filing path of that folder are all true. The third way is to translate the learning topic into natural language and ask the user directly. For example, the learning topic $?(o, a, v, d)$ can be translated into "Does the object o have the property (a, v) in the domain instance d?".

The WH learning topics are translated into natural language and raised to the users in descending order of their priorities. For example, a WH learning topic $(?o, a, v, d)$ can be translated into "Who has the property (a, v) in the domain instance d?". The user can answer a WH learning topic by either provide a list of objects or specifying where the answer can be found. When a folder organization has the second level predicate clauses as filing criteria, the document can be organized based on the knowledge of the objects, which are, to a certain extent, related to the documents. This implies that the user has some knowledge about these objects. Otherwise, the user would not be able to specify such a folder organization. So, it is very likely that this knowledge is maintained somewhere. We have identified two sources that the user can provide in answering a WH learning topic. The first source is documents,

from which the information is derived. Some documents contain facts about the evolving objects. For instance, if a Q.E. Memo (qualifying examination memo) is sent to a person from a department, then it implies that the receiver is a Ph.D. student of that department. The second source for acquiring information is the database. It is possible that the information about objects is stored in database. For example, a human resource database contains information about people of a specific organization. The user is allowed to write a query in response to a learning topic. Depending on different sources of the learning process, we defined three kinds of learning rules, which are as follows:

1. If $a(o, v.d)$ is true, then (o, a, v, d) is a fact.
2. If $Type(\omega, T) \wedge A(\omega, o) \wedge P$, then (o, a, v, d) is a fact.
3. If o is in the result of query Q, then (o, a, v, d) is a fact.

The user can always answer the learning topics directly. But the learning process would then be tedious. The above learning rules can simplify the learning process by allowing the user specifying instructions for acquiring the needed knowledge instead of answering the learning topics directly. The first rule instructs the learning agent to add into the object base the fact that is implied by the a predicate's being true. Many events can imply that a predicate clause, say $a(o, v.d)$, is true. For example, a filing of a frame instance into a folder is forced by the user which implies that the local predicates along the filing path are true, or a predicate clause is evaluated to be true by the inference engine. When the system is idle, the learning agent will invoke the inference engine to derive new knowledge about the objects which are related to the YN learning topics.

The second rule directs the learning agent to ask the user how to acquire the knowledge about objects from documents. More specificly, given a WH learning topic $(?o, a, v, d)$, the second rule directs the learning agent to ask the user three questions.

The first one is "Which type of documents contains the needed information?". The second question is "Where in the documents the objects appear?". The third question is "Any other restrictions?". The answers of these three questions generate a rule which can be used by the learning agent to automaticly extract knowledge from the documents of the given type. For example, given a learning topic (?o, Program, PHD, CIS), the interaction between the user and the learning agent generates a rule Type(ω, "Q.E.MEMO")$\wedge$ Receiver(ω, x)$\wedge$ Department(Sender, CIS) which tells the learning agent that whoever receives a Q.E.MEMO from the CIS department is a Ph.D. student of the CIS department and a fact (x, Program, PHD, CIS) can be stored into the object page of x.

The third rule instructs the agent to ask the user to provide a query which can be used to collect information about objects from database. For example, if there is a table called CISFACULTY that contains all the faculty members of the CIS department, then the user is allowed to specify a query like "SELECT: FNAME, LNAME; FROM: CISFACULTY" in response to the learning topic (?o, Position, FACULTY, CIS) which tells the learning agent that the result of the query is a list of CIS faculty members.

7.4 Performance Analysis of the Predicate Evaluation Engine

Let p be the number of predicate clauses in the predicate being evaluated, m be the average size of a frame instance. Let d_1 be the complexity of the evaluation module I for evaluating a single first level predicate clause, d_2 be the complexity of the evaluation module II for evaluating a single second level predicate clause, and d_3 be the complexity of the inference engine for evaluating a second level predicate clause. Then d_1 will be $O(m)$ if sequential search is used. Note that m can be considered as a constant number. For frame instance with a large m, binary search or hash table can be used, and d_1 can be reduced to $O(\log m)$ or constant respectively.

Therefore, $d_1 = O(1)$. For the same reason, $d_2 = O(1)$. For the inference engine, by indexing the properties in the property relations, the inference engine can determine whether a property implies another in constant time. However, the inference engine has to examine each of the properties in the object page to determine whether it can be used to fire a rule. There are only four rules. So d_3 is $O(k)$, where k is the number of properties in the object page. Assume that, among the p atomic predicate clauses in the input predicate, there are p_1 first level predicate clauses, p_2 second level predicate clauses, and p_3 second level predicate clauses that are evaluated by the inference engine. So $p = p_1 + p_2$. Let d be the time needed for evaluating a predicate. Then $d = p_1 \times O(1) + p_2 \times O(1) + p_3 \times O(k)$. As we have discussed in section 7.3, the learning agent will keep enriching the object base by invoking the inference engine. So it is reasonable to assume that p_3 is insignificant because the inference process is done when the system is idle. Therefore, $d = p_1 \times O(1) + p_2 \times O(1) = (p_1 + p_2) \times O(1) = O(p)$.

CHAPTER 8

AGENT-BASED IMPLEMENTATION OF FOLDER ORGANIZATION

An agent-based filing architecture is employeed for implementing folder organization, which can automate the document filing (i.e., deposit an incoming frame instance into appropriate folders) and cope with the subtleties of the folder reorganization. Each folder is associated with a filing agent containing a criterion. The criteria used to categorize documents are defined in terms of predicates. Agents, depend on how they are connected, communicate and cooperate with each other to implement the folder organization.

8.1 Approaches to Filing Frame Instances

Generally speaking, there are two approaches to filing frame instances into a folder organization.

1. Deposit a frame instance into a folder if the frame instance belongs to the folder.
2. Deposit a frame instance into a folder if and only if the frame instance belongs to the folder but does not belong to any of the descendants of the folder.

The second approach was used in the previous work. In this dissertation, we adopt the first approach based on the following considerations:

- As we have discussed in Section 6.3, the query preprocessing is conducted by comparing the query first against the predicates of folders at highest level. For each folder at the higher-level that is sufficiently similar to the query, a query-predicate comparison is then carried out with the lower level folders. The comparison is repeated until some folders at the lowest level are identified.

The frame instances are eventually identified from these folders at the lowest level. Identifying the actual frame instances can be done by comparing the query against the contents and structures of all possible candidates contained in the selected folders at the lowest level. These candidates are a considerably small number of frame instances of a particular document type. That is, the document (represented by a frame instance) search and retrieval can be narrowed to a search space focusing on the frame instances of a particular type within a particular folder. This can also be accomplished using frame instance search and browsing mechanisms [32, 33, 66]. For using the first filing approach, during the frame instance search and browsing, all the frame instances that belong to a folder could be accessed without collecting them from its descendant folders, on the contrary, the second filing approach requires collecting them.

- During folder reorganization, for the second filing approach, removing a folder from the folder organization requires the return of every frame instance of the folder to its parent folders. At the expense of storing pointers in each folder which point to its belongings (i.e., frame instances), the first filing approach does not require the return of any frame instance of a folder, which is to be removed from the folder organization, to its parent folders.

- During folder reorganization, adding a new folder to the folder organization requires the distribution of frame instances from its parent folders to the new folder. In contrast to the second filing approach, the first filing approach does not require these parent folders to collect all the frame instances from their existing descendant folders.

In comparing these two filing approaches, the first filing approach enhances the performance of frame instance retrieval and folder reorganization at the expense of

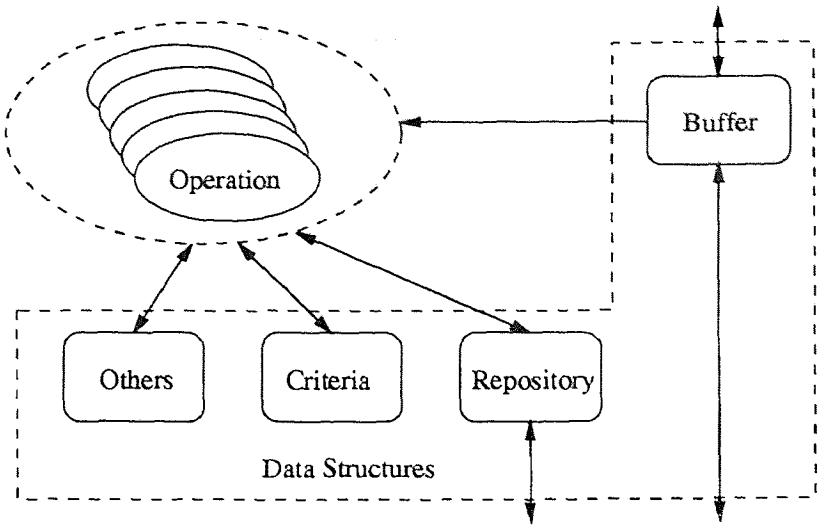


Figure 8.1 The architecture of the filing agent

more storage space for storing frame instance in each folder. But the folder organization is a logical storage of frame instances; each folder contains the pointers to its belongings, which are the frame instances of corresponding documents. Compared with the space required for storing the actual frame instances, the storage space for the folder organization is much smaller. Assuming that each non-leaf folder has two or more descendant folders on average, it is not difficult to prove that the storage space required for the first filing approach is slightly less than twice as much as needed by the second filing approach. It would be far less than twice as much as needed by the second approach, if folders have a larger number of fan-outs.

8.2 Filing Agent

A filing agent is an intelligent object which monitors its associated folder in a folder organization. Each agent has its data structures(called attributes) and operations (or methods; strictly speaking, a method is the implementation of an operation for a class) for manipulating the data structures. The agents communicate with each other through message passing. Figure 8.1 depicts an architecture of a filing agent.

An agent A is said to be a child of agent B (or B is a parent of A) if A's folder is a child of B's. A is a descendant of B (or B is an ancestor of A) if A's folder is a descendant of B's. The terms child(parent) and descendant(ancestor) are used interchangeably.

Each folder is associated with a criterion. All the frame instances in the folder, along with its descendants, must satisfy the criterion of that folder. An agent receives frame instances from its parents and sends the frame instances to its children. There are two ways for sending the frame instances:

1. For any incoming frame instance ω, the parent agent A broadcasts one copy ω to all of A's descendants. A child agent B discards its copy if ω does not satisfy B's criteria.

2. By examining the criteria of the child agents by the parent agent A, A only sends the incoming frame instance to the qualified child agent(s).

TEXPROS employs the first strategy to enhance information hiding and preventing an agent from destroying other agents' criteria. In general, document filing starts from the root agent. For each agent A, when a new frame instance ω arrives at its buffer, A's filing method is invoked to examine if the frame instance belongs to the current folder. If it does, the frame instance will be deposited in A's repository, and it will be sent to all of A's descendants. The frame instance will be discarded from the buffer if it does not belong to A's folder. This will end the filing process on the current branch of the folder organization. The filing process continues until leaf agents or those agents that the frame instance does not belong to their folders are reached.

When a new folder is first added to the folder organization, its filing agent is created automatically by the system. The user needs to define its criterion. A filing agent can be registered as a child of one or more existing agents designated by the

user. After a filing agent is registered, its parent agent will distributes their frame instances to it. The frame instances, which were not in the folder f before it was registered, are placed in it if they belong to it.

When a subfolder x is removed from its parent y, x's agent is withdrawn and it is no longer a child of y's agent. x's descendants will also be unattached from the folder organization. Users can either discard them or attach them to other parts of the folder organization.

8.3 Implementation of Filing Agents

We implement an agent as an object based on the object-oriented paradigm [56]. A filing agent class, which represents a group of filing agents with common attributes, operations and semantics, is defined as follows:

```
class Filing_agent
    attributes
        SetOf frame_instances buffer;
        SetOf frame_instances repository;
        SetOf Filing_agents child_agents;
        SetOf Filing_agents parent_agents;
        Link_type AND_link or OR_link;
        INTEGER threshold;
        StringOf Characters FolderName;
        CRITERIA criteria;
    operations
      Public Operations
        construct(INTEGER);
        destruct( );
        link(Filing_agent, Link_type, Link_mode);
```

unlink(Filing_agent);

rename(StringOf Characters);

disconnect(Filing_agent);

modify_criteria( );

filing(frame_instance);

delete(frame_instance);

distribute(Filing_agent);

Private Operations

discard(frame_instance);

clear();

The buffer of an agent of folder f contains frame instances being processed. The repository contains frame instances that satisfy the agent's criteria. The criteria is a set of conditions for governing the filing and distribution of the frame instances. The threshold specifies the capacity of a filing agent. When the number of frame instances in a filing agent exceeds its threshold, the system will send a warning to the user.

The operations specified in the class definition can be used to manipulate the private data structures. The semantics of the operations are given below:

- A.construct(INTEGER i). This operation initializes the data structures (i.e., attributes) specified in the class definition. The values of i will be assigned to the threshold attribute.

- A.destruct( ). This operation enables the system to reclaim A's storage.

- A.link(Filing_agent B, Link_type L, Link_mode M). This operation creates an L type (i.e., AND or OR) link between B and A. Link_mode M can be IN or OUT. if M is IN, A will be linked as a descendant of B. If M is OUT, A will

be linked as a parent of B. The operation does not cause any frame instance flow between the two agents.

- A.unlink(Filing_agent B). This operation disconnects the link between the agents A and B.
- A.rename(StringOf Characters NewName). This operation changes the name of the folder of the agent A to NewName.
- A.disconnect(). This operation disconnects all incoming links of A.
- A.modify_criteria( ). This operation modifies A's criteria.
- A.filing(frame_instance ω). This operation examines the incoming frame instance ω contained in the buffer of the agent A. If the frame instance belongs to the current folder A, it will be deposited in A's repository and broadcasted to A's descendants.
- A.delete(frame_instance ω). Remove the frame instance ω from A's repository if it is there.
- A.distribute(Filing_agent F). This operation sends all the frame instances in A's repository to the buffer of the agent F, where F must be the descendant of A.
- A.discard(frame_instance ω). This operation discards the frame_instance ω from the buffer of the agent A.
- A.clear(). Empty the repository.

8.4 Folder Reorganization

We have identified seven basic operations that can change the logical folder organization. They include Create_folder, Destroy_folder, Link_folder, Unlink_folder,

Register_folder, Disconnect_folder, and Rename_folder. All these operations are implemented by invoking the operations defined for the folder agent. For notational simplicity, we use the same identifier to represent both the folder and its agent.

- Create_folder(f)–This operation creates a folder f.
- Link_folder(f_1, f_2, L)–This operation makes a folder f_2 to be the descendant of f_1 by creating a L type link (i.e., either an AND-link or OR-link) between them. It is implemented by invoking f_1.link(f_2, L, "OUT") and f_2.link(f_1, L, "IN").
- Unlink_folder(f_1, f_2)–This operation removes the link between f_2 and its ancestor f_1 by invoking f_1.unlink(f_2) and f_2.unlink(f_1).
- Register_folder(f)–This operation registers folder f into the folder organization. When a new folder f is created and connected with links to the folder organization, it is inactive because its repository is still empty. It becomes active when all the frame instances which should belong to it are deposited into its repository. Registering means to make the folder f active. This is done by distributing frame instances from all of f's parents to f.
- Rename_folder(f, NewName)–This operation changes the name of the folder f to another name NewName, by invoking f.rename(NewName).
- Disconnect_folder(f)–This operation disconnects folder f and all its descendants from the folder organization.
- Destroy_folder(f)–This operation destroys folder f.

8.5 Cooperation Between Filing Agents

In previous section, we gave the data structure and operations of the filing agents. However, in order to implement the folder organization, the filing agents must

cooperate with each other. Here we shall discuss the cooperation based on different events.

8.5.1 Filing Process

During the filing process, whenever an agent gets a frame instance (i.e. an incoming frame instance is determined to belong to the current folder), it should broadcast the frame instance to all its children (i.e., immediate descendant agents) so the filing process can go further. To be deposited into a folder B, the incoming frame instance must satisfy the local predicate of the folder and the semantics of the folder organization. It is much easier or simpler to check whether a frame instance satisfies the semantics of the folder organization, in comparison with the process of evaluating whether a frame instance satisfies the local predicate. So the filing operation of the agent will do semantics check before examine whether the frame instance satisfies the local predicate. The predicate evaluation is done by the evaluation engine, which was described in Chapter 7. According to the definition of OR-link and AND-link, an incoming frame instance passes the semantics check if:

1. The frame instance comes through an OR-link (i.e. sent by a parent that is connected with an OR-link), or

2. The frame instance comes through all the AND-links (i.e. each of the parents that are connected by AND-links sent a copy of it), or

3. The frame instance comes through some of the AND-links, and it also belongs to the parents that are connected with the other AND-links.

8.5.2 Manipulating Links

If an agent A is told to add an link to (or from) an agent B (i.e. the link() operation is invoked), A should inform B to add a link from (or to) A. Similarly, if an agent A is told to remove an link to (or from) an agent B (i.e. the unlink() operation is

invoked), A should inform B to remove the link from (or to) A. A folder should be removed from the folder organization, which is a rooted DAG, if the removal of an incoming link causes the folder to be disconnected(i.e., no incoming link exists). All its outgoing links will also be removed. The process will be propagated down to the leaves.

Adding a link between two folders A and B (say, A is the parent of B) will change the semantics of the folder organization. It requires to update the contents of some folders (the child folder B and its descendants): remove the frame instances that no longer belong to these folders, and deposit the ones that should belong to these folders. For efficiency purpose, the link() operation of a filing agent does not cause any frame instance flow between the two folders. Because a folder is allowed to have multiple incoming links, it would be more efficient to update the content of the folder after the completion of re-organizing the folder organization. The update is done according to the follow rules:

1. If links to a newly created folder A have been added, the corresponding agent of the folder A will ask its parent agents to distribute their frame instances to it (invoking the distribute() operation). According to the definition of OR-link and AND-link, any frame instance that belongs to a child folder must belong to at least one of the parent folders. So there is no need to examine all the frame instances in the frame instance base.

2. If new OR-links to a existing folder A have been added, the agent A will ask all of its parent folders that are connected with these new OR-links to distribute their frame instances to it.

3. If new AND-links to a existing folder A have been added and no other incoming AND-link exists, the agent A will ask its all parent folder that are connected with these new AND-links to distribute their frame instances to it.

4. If new AND-links to a existing folder A have been added and there are existing incoming AND-links, the agent A will remove any frame instance in the folder A if the frame instance does not belong to:

   (a) all the parent folders which are connected with AND-links (including the newly added AND-links), and

   (b) any parent folder which is connected with an OR-link.

   The addition of the new AND-links to folder B strengthens the condition of the existing incoming AND-links. So some frame instances will have to be removed from the folder.

When a link from folder A to folder B is removed (invoking unlink() operation), the child agent B and B's descendant agents will update their repositories. The update is done according to the follow rules:

1. If it is an OR-link, remove any frame instance in the B's repository, if the frame instance:

   (a) belongs to the parent folder that is connected with the removed OR-link; and

   (b) does not belong to any remaining parent folder that is connected with an OR-link; and

   (c) does not belong to all the parent folders that are connected with AND-links.

2. If it is an AND-link, the agent A will ask its parent agents that are connected with the remaining AND-links to distribute all its frame instances. The removing of an AND-link weakens the condition of the incoming AND-links. Some frame instances that were rejected before may now belong to the current folder.

8.5.3 Deleting Frame Instances

When an agent is told to delete a frame instance, it will remove the frame instance from the repository if it satisfies the following conditions:

1. It is in the repository,
2. It does not belong to any parent folder that is connected with an OR-link, and
3. It does not belong to all the parent folders that are connected with AND-links.

If the frame instance is removed, the agent will also ask all of its children to delete it. The above restrictions guarantee that a frame instance cannot be removed directly from a non-root folder because the removal violates the semantics of the folder organization. To remove a frame instance from the folder organization, one has to remove it from the root folder.

8.5.4 Modifying Filing Criteria

When the criterion (i.e., local predicate) of a folder A is modified, the agent A clears its repository first. Then it will ask all of its children to delete the frame instances that were previously in its repository. Simply clearing the repository will violate the semantics of the folder organization. So the clearing process will make the folder and its descendants to be inactive, just like new created folders. After the modification is completed, the agent will ask all of its parents to distribute their frame instances to it.

8.5.5 Sending Frame Instances

When sending a frame instance ω from an agent A to an agent B (this happens when A broadcasts a frame instance to its children during filing process, or A distributes its frame instances to its children), the agent A will tag ω with the name of the folder A. The reason for doing so is that a filing agent may receive the same frame

instance from its multiple parent agents. Tagging technique helps the child agents to keep track of the sources of the frame instances and to process them correctly. The tagging is also needed by the agents for doing semantics check during the filing process.

8.6 Performance Analysis of the Filing Process

This agent-based architecture simplifies the document filing significantly. Firstly, a filing process (i.e. the process that files a frame instance into the folder organization) starts from the root agent (by simply invoking the filing() operation of the root agent) and then propagates to its descendants. There is no need to traverse the folder organization in the program. The propagation is accomplished by the cooperation between filing agents.

Secondly, this agent-based architecture can easily be implemented in concurrent and multithread mode. This is because there is no communication between any two invoked filing operations of different agents. The filing process needs not be in topological numbered order. For example, the filing operation of an agent can be invoked when the agent receives a frame instance from a parent which is connected with an OR-link. So an agent could start its filing process without waiting all its parents to finish. In other words, an agent could finish its own filing process earlier than some of its parents. This increases the concurrency on a multi-processor machine.

Assuming the time needed for each agent to finish its filing process is d. As we have discussed earler, the filing process of each agent involves the predicate evaluation and the semantics check(i.e., if the frame instance comes from an OR-linked parent, or comes from all AND-linked parents). Compared with the predicate evaluation, the time for semantics check is insignificant. According to chapter 7, the complexity of a predicate evaluation process is $O(p)$, where p is the number of predicate clauses

in the predicate. It is reasonable to assume that p is not significant. So d is $O(1)$. Let k be the number of folders in the folder organization, l be the number of processors. When $l \ll k$, the complexity of the a filing process is $d \times (k/l)$, or $O(k)$. When $l \approx k$, the complexity will be $d \times \log k$, or $O(\log k)$.

CHAPTER 9

IMPLEMENTATION AND FUTURE WORK

In this chapter, we shall present the overall architecture of the document filing system and report the progress in implementation. The future work will be discussed in section 9.2.

9.1 System Architecture and Implementation

The proposed filing system is implemented using the Java language. Figure 9.1 shows the overall architecture of the system. The user specifies the folder organization through the GUI. The folder reorganization operations defined in section 8.4 are carried out by the filing agents. The bookcase organization and original document storage are transparent to the user. From the user's point of view, frame instances are stored in the folder organization. But actually the folder organization is only a logical storage. Upon the arriving of a new document, the original document filing module stores the original document into the system storage at the original document level. The bookcase agent then deposits the frame instance into the bookcase organization. The pointer of the frame instance is sent to the filing agents, where it will be filed into the folder organization. The filing of the frame instance involves the process of predicate evaluation which determines if a frame instance satisfies a predicate. This is done by calling the control module of the predicate evaluation engine. The control module parses and divides the inputed predicate into predicate clauses and constraints. It controls the other modules of the evaluation engine and makes the final conclusion based on the outputs of them. The evaluation process has three phases. In the first phase, it involves the evaluation module I to evaluate the first level predicate clauses, which specify some characteristics of frame instances. In the second phase, the evaluation module II evaluates the second level predicate clauses.

Any second level predicate clauses which cannot be successfully evaluated go to the third phase, where the inference engine makes further evaluation using the knowledge base. The knowledge base consists of an object base and a domain knowledge base. It is dynamicly created by the learning agent based on the needs for supporting the current folder organization. So the folder organization as a flexible document filing model is made possible by the learning agent.

9.2 Future Work

The major objective of this dissertation is to provide a flexible document filing model which is the folder organization. In order to get the user involved in specifying the folder organization, the dissertation defined a predicate-based language for specifying filing criteria. The folder organization was extended for simplying the local predicates of the folders. It is shown that the folder organization helps a lot in document retrieval. As future work, we need to investigate what are the requirements of a good folder organization, and how to refine an existing folder organization. To support sophisticated users, we need to investigate how to extend both the predicate specification and the folder organization without increasing the requirements of the average users.

To support such a flexible document filing model, the dissertation presents a learning agent for acquiring the knowledge which is needed in document filing and retrieval. The learning agent gives the system the capability of being used in different application domains. There are two issues regarding the knowledge base that should be covered in our future work. The first one is the aging of the knowledge base. While new knowledge keeps coming, some knowledge becomes infrequently used or useless. To keep the size of the knowledge base small, the old knowledge should be moved out of the knowlesge base. Some requirements are needed for determining which piece of knowledge is old enough for moving out. The second issue is to investigate how

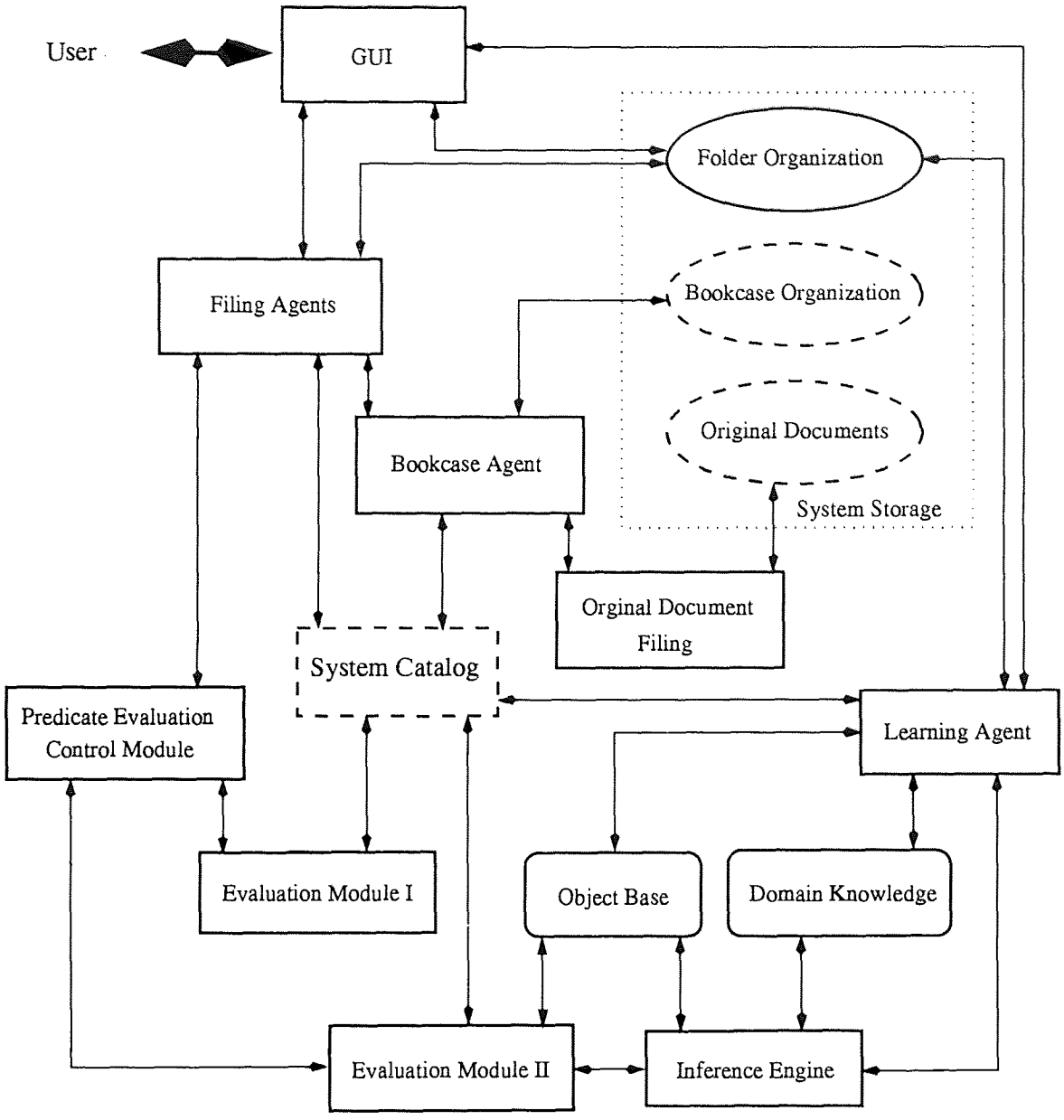


Figure 9.1 The system architecture

the knowledge base can be used for helping a naive user to specify a better folder organization. This is another direction of the learning process. When the system get experienced, it should be able to train the user so that the system can be used more efficiently.

CHAPTER 10
CONCLUDING REMARKS

This dissertation presents a knowledge-based, predicate-driven document filing system using the notion of folder organization, which supports the information representation and manipulation, and conveys meanings from stored information within office documents. Analogous to a cluster file organization, in this organization, folders are used as repositories of collections of frame instances (the synopses of the original documents), regardless of their document types, which satisfy the criteria (specified in terms of predicates) of the folders. It would be possible to determine in which folder a new document would fall, and to find which folder of frame instances a query best fits.

By integrating the document type hierarchy and the folder organization, the dual modeling approach provides a flexible search and retrieval facility that allows browsing through collections of frame instances and retrieval of frame instances according to different criteria, using information related to document types and the frame instances in close proximity within a folder in the folder organization. It provides efficient frame instance access by limiting the searches to those frame instances of a specific document type within those folders which appear to be most relevant to the queries. This search strategy filters out first any frame instance that is of irrelevant document type or in irrelevant folder, and then compares the contents of a small collection of frame instances of an identified document type against the query formulations. This is quite analogous to the global-local text similarity comparison, and should produce a high degree of retrieval precision. Furthermore, the browsing technique[67] can be conducted effectively by moving a given query toward the relevant items, such as frame instance type, frame instances of a document type, folders containing relevant frame instances of a document type, and so forth, using the well-known relevance-feedback process[26, 28].

For supporting the interaction between the user and the system, we formalized the new predicate specification for specifying folder criteria or queries. The user can use predicates to specify characteristics of frame instances and/or the properties of objects which are related to the documents(i.e., those appear as values of attributes of frame templates for describing the document types). For clear understanding and precise interpretation (and therefore, preventing frame instances' falsedrop), we introduced the concepts of the first level predicate clauses and the second level predicate clauses, which are used to specify characteristics of frame instances and the properties of objects which are related to the documents, respectively. We also disambiguated and formalized the values in predicates by specifying the domains in which the values are defined. We extended the notion of a DAG structured folder organization to a DAG structured folder organization with multiple typed links. By introducing a new AND-link between folders, the predicates as the criteria for filing frame instances into folders can be simplified and, in turn, can be easily evaluated.

The system storage employs a three-level architecture for incorporating the dual models. The original documents are stored at the first level. The frame instances are stored at the second level and the third level. The physical storage at the second level is organized based on the document type hierarchy. The logical storage at the third level is organized based on the folder organization. By using this architecture, the folder organization and the document type hierarchy cooperate with each other in order to speed up the search and the retrieval of documents. We apply the notion of encapsulation (which is most often achieved through information hiding) for designing the system repository: the implementation of the folder organization at the third level is independent from the physical storage for the frame instances at the second level, whose implementation is, in turn, independent from the physical storage for the original documents at the first level. Thus this leads to a clear separation of concerns. This multilevel access structure supports direct

access to documents which requires retrieving their corresponding frame instances through the use of specific information, such as attributes and document type, from the document type hierarchy and the folders containing these frame instances. This system storage supports a three-level retrieval strategy. Firstly, a knowledge-based query preprocess is applied for efficiently reducing the search space to a small set of frame instances, using the information in the query formula. Secondly, the frame instance search is applied on the small set of frame instances. The documents are identified by examing which frame instances in the search base satisfy the query formula. This frame instance search can be knowledge-based and/or content-based depending on whether the query formula contains second level predicate clauses and/or first level predicate clauses. Since the frame instances are synopses of their original documents, the content-based frame instance search is also a conceptual-based search of the original documents. The second level storage provides a platform for applying various text-based IR techniques. It makes possible to use text-based IR techniques for content-based multimedia document retrieval. The knowledge base, used for evaluating predicates, supports knowledge-based document retrieval. Finally, the third level storage provides a platform for adopting potential content-based multimedia document retrieval techniques.

We extended an agent-based filing architecture to implement the extended notion of folder organization. Each folder is monitored by a filing agent. A set of filing agents is needed for implementing a folder organization. The filing agents communicate and cooperate with each other for providing functionalities of the folder organization. The dissertation defined the cooperation among the filing agents. A filing process starts from the root agent. To be deposited into a folder, a frame instance must pass the semantics check and satisfy the local predicate of the folder. Upon storing a frame instance ω into a folder, the agent then distributes a copy of the frame instance ω to its immediate descendants. The agent will discard the frame

instance if it does not belong to the folder. The filing process is propagated down to the leaf agents. We adopted the approach that a frame can appear in multiple folders if it belongs to them, instead of only one copy of a frame instance can exist in the folder organization which is employed in our previous work. This new approach frees the agents from collecting the frame instances from their descendants during the folder reorganization and document browsing and retrieval.

To automate document filing and folder reorganization, we established the fact that any frame instance ω deposited in a folder f must satisfy the global predicate of f, which is composed of the local predicates of the folders of the filing paths of f from the rooted folder f_0. We presented a predicate evaluating system for determining automatically whether a frame instance satisfies a predicate. The input predicate is parsed and divided into predicate clauses and constraints. The first level predicate clauses are evaluated based of the information contained in the frame instance. The second level predicate clause specify the properties of objects, such as Position(Sender, FACULTY.CIS.NJIT) and DateOfBirth(John Smith, x). The process of evaluating these predicate clauses is no longer straightforward and simple, and it requires additional knowledge about the objects, which is not in the original documents. To facilitate the evaluation process, we presented a knowledge base, which consists of an object base and a domain domain knowledge base.

An object page for an object contains a number of (attribute, value) pairs which characterize the properties of the object. Depending on the object type, attributes in an object page are domain-dependent. The information contained in the object page of an object can be used for evaluating predicates. Given a goal predicate clause, such as Position(John Smith, FACULTY.CIS.NJIT), if (Position, FACULTY.CIS.NJIT) has an exact match with the property in the object page created for the object John Smith, then the goal predicate clause is true. Likewise, given an assignment predicate clause, such as Affiliation(John Smith, x), x will have the value NJIT, depending

upon the availability of the (attribute:Affiliation, value:NJIT) pair in the object page of John Smith.

We presented a domain knowledge which consists of two components, domain organization and property relations. The domain organization specifies the domains and their relationships using is-a-domain-of or is-a subdomain-of. For example, the domain Affiliation is composed of two subdomains, Department and Office, and the domain Department has a subdomain Lab. The property relations specify the properties for all entities in the domain organization. This provides an expressive power for describing the properties of interest and their relationships within the domain. Each relation between two properties defines a rule that one property implies another, for instance, (Position, PROFESSOR) implies (Position FACULTY) within a department domain (i.e., all professors are faculty members in a academic department of a university. We introduced four rules for using the domain knowledge. With the domain knowledge base, more knowledge about an object can derived based on the object page of the object. For instance, based on the object page in Figure 7.2, we are able to state that John Smith is an employee of the CIS department at NJIT.

For supporting the dynamic and flexible modeling, the dissertation presented a learning agent for acquiring the knowledge which is needed by the predicate evaluation engine. Once a folder organization is specified by the user, the domain knowledge is automaticly extracted from the folder organization. The learning rules were given for guiding the extraction. The attributes in each object page are determined based on the property relations in the domain knowledge base. With the help of the domain knowledge base, the agent knows what knowledge about the objects is needed for supporting the document filing. The acquisition of the knowledge in the object base is done in two ways. The first one is learning from the filing agents or the inference engine. The second one is learning from the user. The knowledge obtained by the learning agent in one application cycle can be used

in another application cycle, provided that the two application domains are similar. So the system can get experienced. The system will get trained when it is used by a sophisticated user. When get experienced, the system can help an naive user to specify a better folder organization.

REFERENCES

1. A. Aho and M. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, June 1975.

2. D. Anastassiou, M. K. Brown, H. C. Jones, J. L. Mitchell, W. Pennebaker, and K. Penningtonlll, "Series 1 Based Videoconferencing System," *IBM Systems Journal*, vol. 22, no. 1/2, pp. 97–110, 1983.

3. N. Bianchi, P. Mussio, M. Padula, and G. R. Rinaldi, "Multimedia Document Management: An Anthropocentric Approach," *Information Processing & Management*, vol. 32, no. 3, pp. 287–303, 1996.

4. A. Celentano, M. Fugini, and S. Pozzi, "Querying Office Systems about Document Roles," in *Proceedings of the 14th Annual Int. ACM/SIGIR Conference on Research and Development in Information Retrieval*, Chicago, Illinois, pp. 183–189, October 1991.

5. A. Celentano, M. Fugini, and S. Pozzi, "Knowledge-Based Document Retrieval in Office Environments: The Kabiria System," *ACM Transactions on Office Information Systems*, vol. 13, no. 3, pp. 237–268, July 1995.

6. S. Chen, *Document Preprocessing and Fuzzy Unsupervised Character Classification*, PhD dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, May 1995.

7. S. Christodoulakis, M. Theodoridou, M. P. F. Ho, 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.

8. S. Cisco and J. Wertzberger, "Indexing Digital Documents," *Inform*, vol. 11, no. 2, pp. 12–20, February 1997.

9. P. Cohen and R. Kjeldsen, "Information Retrieval by Constrained Spreading Activation in Semantic Networks," *Information Processing and Management*, vol. 23, no. 4, pp. 255–268, 1987.

10. W. Croft, "NSF Center for Intelligent Information Retrieval," *Communications of the ACM*, vol. 38, no. 4, pp. 42–43, April 1995.

11. W. Croft and R. Krovetz, "Interactive Retrieval of Office Documents," in *Proceedings ACM-IEEE Conference on Office Information Systems*, New York, pp. 228–235.

12. P. Dadam and V. Linnemann, "Advanced Information Management (AIM): Advanced Database Technology for Integrated Applications," *IBM Systems Journal*, vol. 28, no. 4, pp. 661–681, 1989.

13. P. J. Denning, "Electronic Junk," *Communications of the ACM*, vol. 25, no. 3, pp. 163–165, March 1982.

14. X. Fan, Q. Liu, and P. A. Ng, "A Multimedia Document Filing System," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Ottawa, Ontario, Canada, pp. 492–499, June 1997.

15. X. Fan, Q. Liu, and P. A. Ng, "Knowledge-Based Document Filing: TEXPROS Approach," in *Proceedings of the 13th International Conference on Advanced Science and Technology in conjunction with the 2nd International Conference on Multimedia Information Systems*, Schaumburg, Illinois, USA, pp. 58–67, April 1997.

16. X. Fan and P. A. Ng, "A Dual Model Approach For Modeling Office Documents," To appear in *Proceedings of the Third World Conference on Integrated Design & Process Technology*, 1998.

17. X. Fan and P. A. Ng, "Personal Document Management and Retrieval: A Knowledge-Based Approach," To appear in *Journal of Systems Integration*, 1998.

18. E. A. Fox, R. Akscyn, R. Furuta, and J. Leggett, "Digital Libraries - Introduction," *Communications of the ACM*, vol. 18, no. 4, pp. 22–29, April 1995.

19. 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, October 1983.

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

21. X. Hao, J. Wang, M. Bieber, and P. Ng, "Heuristic Classification of Office Documents," *International Journal of Artificial Intelligence Tools*, vol. 3, no. 2, pp. 233–265, 1994.

22. X. Hao, J. Wang, and P. A. Ng, "Information Extraction from the Structured Part of Office Documents," *Information Sciences*, vol. 91, no. 3/4, pp. 245–274, 1996.

23. X. Hao, J. Wang, and P. Ng, "Nested Segmentation: An Approach for Layout Analysis in Document Classification," in *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, pp. 319–322, October 1993.

24. R. Hunter, P. Kaijser, and F. Nielsen, "ODA: A Document Architecture for Open Systems," *Computer Communication*, vol. 12, no. 2, pp. 69–79, April 1989.

25. C. Huser, K. Reichenberger, L. Rostek, and N. Streitz, "Knowledge-Based Editing and Visualization for Hypermedia Encyclopedias," *Communications of the ACM*, vol. 18, no. 4, pp. 49–51, April 1995.

26. E. Ide and G. Salton, "Interactive Search Strategies and Dynamic File Organization in Information Retrieval," in *The Smart Retrieval System - Experiments in Automatic Document Processing* (G. Salton, ed.), pp. 373–393, 1971.

27. N. Jardine and C. van Rijsbergen, "The Use of Hierarchic Clustering in Information Retrieval," *Information Storage and Retrieval*, vol. 7, no. 5, pp. 217–240, December 1971.

28. J. J.J. Rocchio, "Relevance Feedback in Information Retrieval," in *The Smart Retrieval System - Experiments in Automatic Document Processing* (G. Salton, ed.), pp. 313–323, 1971.

29. D. Knuth, J. Morris, and V. Pratt, "Fast Pattern Matching in Strings," *SIAM Journal of Computing*, vol. 6, no. 2, pp. 323–350, June 1977.

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

31. Q. Liu and P. Ng, "A Browser of Supporting Vageu Query Processing in an Office Document System," *Journal of Systems Integration*, vol. 5, no. 1, pp. 61–82, 1995.

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

33. Q. Liu and P. Ng, "A Query Generalizer for Providing Cooperative Responses in an Office Document System (revised version)," *to appear in Data and Knowledge Engineering*, 1998.

34. Q. Liu, J. Wang, and P. Ng, "An Office Document Retrieval System with the Capability of Processing Incomplete and Vague Queries," in *Proceedings of the Fifth International Conference on Software Engineering and Knowledge*, San Francisco, California, pp. 11–17, June 1993.

35. E. Lutz, H. Kleist-Retzow, and K.Hoernig, "MAFIA - An Active Mial-Filter-Agent for an Intelligent Document Processing Support," *Multi-User Interfaces and Applications*, pp. 16–32, 1990.

36. T. Malone, K. Grant, 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.

37. J. Martin, *The Wired Society: A Challenge for Tomorrow*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

38. B. McCune and et al., "RUBRIC: A System for Rule-Based Information Retrieval," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 9, pp. 939–945, September 1985.

39. C. Meadow, *Text Information Retrieval Systems*, Academic Press, San Diego, California, 1992.

40. C. Meghini, R. Fausto, and C. Thanos, "Conceptual Modeling of Multimedia Document," *Computer*, vol. 24, no. 10, pp. 23–29, 1991.

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

42. F. Mhlanga, Z. Zhu, J. Wang, and P. Ng, "A New Approach to Modeling Personal Office Documents," *Data and Knowledge Engineering*, vol. 17, no. 2, pp. 127–158, November 1995.

43. N. Naffah and A. Karmouch, "AGORA - An Experiment in Multimedia Message Systems," *Computer*, vol. 19, no. 5, pp. 56–66, May 1986.

44. E. Nodtvedt, "Information Retrieval in the Business Environment," Technical Report, Department of Computer Science, Cornell University, TR 80-447, Ithaca, New York, December 1980.

45. B. Nubila, "Concept-Based Indexing and Retrieval of Multimedia Documents," *Information Sciences*, vol. 20, no. 3, pp. 185–196, 1994.

46. E. Ozkarahan, "Multimedia Document Retrieval," *Information Processing & Management*, vol. 31, no. 1, pp. 113–131, 1995.

47. S. Pierre and H. Safa, "Models for Storing and Presenting Multimedia Documents," *Telematics and Informatics*, vol. 13, no. 4, pp. 233–250, 1996.

48. S. Pozzi and A. Celentano, "Knowledge-Based Document Filing," *IEEE Expert*, pp. 34–45, October 1993.

49. J. S. Quarterman and J. C. Hoskins, "Notable Computer Networks," *Communications of the ACM*, vol. 29, no. 10, pp. 932–970, October 1986.

50. R. Rao, J. Pedersen, M. Hearst, J. Mackinlay, S. Card, L. Masinter, P.-K. Halvorsen, and G. Robertson, "Rich Interaction in the Digital Library," *Communications of the ACM*, vol. 18, no. 4, pp. 25–39, April 1995.

51. S. Sakata and T. Ueda, "A Distributed Office Mail System," *Computer*, vol. 18, no. 10, pp. 106–116, October 1985.

52. G. Salton, *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*, Addison Wesley, Massachusetts, 1988.

53. G. Salton, J. Allan, and C. Buckley, "Automatic Structuring and Retrieval of Large Text Files," *Communications of the ACM*, vol. 3, no. 2, pp. 97–108, February 1994.

54. G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*, McGraw Hill, New York, 1983.

55. S. Saxin, "Computer-Based Real-time Conferencing System," *Computer*, vol. 17, no. 10, pp. 33–35, October 1985.

56. M. Snoeck and G. Dedene, "Generalization/Specification and Role in Object Oriented Conceptual Modeling," *Data and Knowledge Engineering*, vol. 19, no. 2, pp. 171–195, June 1996.

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

58. R. Thomas, H. Forsdick, T. Crowley, R. Schaaf, R. Thomlinson, V.M.Travers, and G. Robertson, "Diamond: A Multimedia Message System Build on a Distributed Architecture," *IEEE Computer*, vol. 18, no. 12, pp. 65–78, December 1985.

59. K. Tracy and P. Bouthoorn, *Object-Oriented Artificial Intelligence Using C++*, Computer Science Press, New York, 1996.

60. D. Tsichritzis, "Form Management," *Communications of the ACM*, vol. 25, no. 7, pp. 453–478, July 1982.

61. M. Turoff and S. R. Hiltz, "The Electronic Journal: A Progress Report," *Journal of the ASIS*, vol. 33, no. 4, pp. 195–202, July 1982.

62. J. Tydeman, H. Lipinski, R. Adler, M. Nyhan, , and L. Zwimpfer, *Teletext and Videotex in the United States - Market Potential, Technology, Public Policy Issues*, McGraw-Hill, New York, 1982.

63. W. Ulrich, "Introduction to Electronic Mail and Implementation Considerations in Electronic Mail," in *AFIPS Conference Proceedings*, Arlington, Virginia, pp. 485–492, 1980.

64. E. Voorhees, "The Cluster Hypothesis Revisited," in *Proceedings of the Eighth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, pp. 188–196, June 1985.

65. C. Wang, *An Intelligent Browser for TEXPROS*, PhD dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, May 1998.

66. C. Wang, Q. Liu, and P. Ng, "Browsing in an Information Repository," in *Proceedings of 2nd World Conference on Integrated Design and Process Technology*, Austin, Texas, pp. 48–56, December 1996.

67. C. Wang, Q. Liu, and P. Ng, "Intelligent Browser for TEXPROS," in *Proceeding of International Conference on Intelligent Information Systems Technology*, Grand Bahamas Island, The Bahamas, pp. 389–398, December 1997.

68. J. Wang, F. Mhlanga, Q. Liu, W. Shang, and P. Ng, "An Intelligent Documentation Support Environment," in *Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, California, pp. 429–436, June 1993.

69. J. Wang and P. Ng, "TEXPROS: An Intelligent Document Processing System," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 4, pp. 171–196, April 1992.

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

71. C. Wei, Q. Liu, J. Wang, and P. Ng, "Knowledge Discovering for Document Classification Using Tree Matching in TEXPROS (revised version)," *Information Sciences*, vol. 100, no. 1-4, pp. 255–310, August 1997.

72. C. Wei, J. Wang, X. Hao, and P. Ng, "In Inductive Learning And Knowledge Representation for Document Classification: The TEXPROS Approach," in *Proceedings of 3rd International Conference on Systems Integration*, Sao Paulo, SP, Brazil, pp. 1166–1175, August 1994.

73. M. Williams, "Electronic Databases," *Science*, pp. 445–446, April 1985.

74. C. Winkler, "Desktop Publishing," *Datamation*, vol. 32, no. 23, pp. 92–96, December 1986.

75. R. Wirfs-Brock and R. Johnson, "Surveying Current Research in Object-Oriented Design," *Communications of the ACM*, vol. 33, no. 9, pp. 104–124, September 1990.

76. Z. Zhu, Q. Liu, J. McHugh, and P. Ng, "A Predicate Driven Document Filing System," *Journal of Systems Integration*, vol. 6, no. 3, pp. 373–403, September 1996.

77. Z. Zhu, J. McHugh, J. Wang, and P. Ng, "A Formal Approach to Modeling Office Information Systems," *Journal of Systems Integration*, vol. 4, no. 4, pp. 373–403, December 1994.