Copyright Warning & Restrictions

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

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

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

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

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

ABSTRACT

KNOWLEDGE MANAGEMENT FOR TEXPROS

by
Jianshun Hu

Most of the document processing systems today have applied AI technologies to support their system intelligent behaviors. For the application of AI technologies in such systems, the core problem is how to represent and manage different kinds of knowledge to support their inference engine components' functionalities. In other words, knowledge management has become a critical issue in the document processing systems. In this dissertation, within the scope of the TEXt PROcessing System (TEXPROS), we identify knowledge of various kinds that are applicable in the system. We investigate several problems of managing this knowledge and then develop a knowledge base for TEXPROS. In developing this knowledge base, we present approaches to representing and managing different kinds of knowledge to support its inference engine components' functionalities.

In TEXPROS, a dual-model paradigm is used, which contains the folder organization and the document type hierarchy, to represent and manage documents. We introduce a new System Catalog structure to represent and manage the knowledge for TEXPROS. This knowledge includes the system-level information of the folder organization and the document type hierarchy, and the operational level information of the document base itself. A unified storage approach is employed to store both the operational level information and system level information. Such storage is to house the frame template base and frame instance base.

An enhanced two-level thesaurus model is presented in this dissertation. When dealing with special kinds of data in processing documents, a new structure "DataDomain" is presented, which supports the extended thesaurus functionalities, pattern recognition and data type operations. Based on the dual-model paradigm of TEXPROS, a concept of "Semantic Range" is presented to solve the sense ambiguity problems. In this dissertation, we also present the approaches to implement the general KeyTerm transformation and approximate term matching of TEXPROS.

Finally, a new component "Registration Center" at the knowledge management level of TEXPROS is presented. The registration center aims to help users handle knowledge packages for specific working domain and to solve the knowledge porting problem for TEXPROS. This dissertation is concluded with the future research work.

KNOWLEDGE MANAGEMENT FOR TEXPROS

by
Jianshun Hu

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 1999

APPROVAL PAGE

KNOWLEDGE MANAGEMENT FOR TEXPROS

Jianshun Hu

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

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

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

Dr. Franz Kurfess, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT, Newark, NJ

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

BIOGRAPHICAL SKETCH

Author: Jianshun Hu

Degree: Doctor of Philosophy

Date: May, 1999

Undergraduate and Graduate Education:

Doctor of Philosophy in Computer Science,
New Jersey Institute of Technology, Newark, New Jersey, 1999

Master of Science in Computer Science,
New Jersey Institute of Technology, Newark, New Jersey, 1998

Bachelor of Automatic Control,
Shanghai JiaoTong University, Shanghai, P.R. China, 1995

Major: Computer Science

Publications:

Xuhong Li, Jianshun Hu, Xien Fan, Chih-Ying Wang and Peter A. Ng, "Automated Document Filing and Retrieval System: An Overview," in *Proceedings of the 3rd Biennial World Conference on Integrated Design and Process Technology*, Berlin, Germany, pp. 231-241, July 1998.

Jianshun Hu, Xuhong Li, D.C. Hung, Simon Doong and Peter A. Ng, "Managing Knowledge for an Intelligent Document Processing System," To appear in *Proceedings of the 1st International Conference on Enterprise Information Systems*, March 1999.

Xuhong Li, Jianshun Hu, Zhenfu Cheng, D.C. Hung and Peter A. Ng, "Automatic Document Analysis and Understanding System," To appear in *Proceedings of the 1st International Conference on Enterprise Information Systems*, March 1999.

Jianshun Hu, Xuhong Li, Simon Doong, D.C. Hung and Peter A. Ng, "A Thesaurus Model for Document Processing System: The TEXPROS Approach," To appear in *Proceedings of the 4th World Conference on Integrated Design and Process Technology*, June 1999.

Xuhong Li, Jianshun Hu, Zhenfu Cheng, Simon Doong, D.C. Hung, C.S. Wei and Peter A. Ng, "An Integrated Document Processing System: Document Classification and Information Extraction," To appear in *Proceedings of the 4th World Conference on Integrated Design and Process Technology*, June 1999.

This dissertation is dedicated to
my parents and my lovely wife

ACKNOWLEDGEMENT

The author would like to take great pleasure in acknowledging his research advisor, Professor Peter A. Ng, for his kindly assistance and remarkable contribution to this dissertation. He spent time and effort to review the drafts of the manuscripts, and provided a lot of helpful comments and crucial feedback that make this final manuscript possible. Special thanks are given to Dr. Gary Thomas, Dr. D.C. Hung, Dr. Franz Kurfess and Dr. Ronald Curtis for their continuous support in reviewing the research progress and providing feedback with valuable comments.

TABLE OF CONTENTS

Chapter **Page**

Chapter **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In the information era of today, large amounts of documents are produced, stored and circulated among offices in the office environment. There is a significant need to use an intelligent document processing system to manage these documents. Many document systems have been developed [2, 5, 12, 52, 53, 56, 57, 68, 74, 79], which are intended to relieve the burdens for processing and managing these documents, either in a hard copy or digital format. The application domains of these systems are quite specific. For example, several systems, such as Diamond [65], MULTOS [20, 66, 67], MINOS [6] were developed to handle the multimedia information documents. There are various text-based and bibliographic document retrieval systems, such as the automatic text structure and retrieval [54, 56, 57], the Kabiria's distributed client/server architecture [3, 4], the retrieval system RUBRIC [43], the Intelligent Interface for Information Retrieval (I^3R) [12], the Grant [11], and the IS RUSSIA [13]. For filtering and sorting electronic message documents, there are a variety of systems, such as AGORA [46], electronic mail systems [48, 70], MAFIA [40], electronic information services [42, 69], INFORMATION LENS [41], electronic publication systems [68,80], Digital Library [19, 27] and teleconferencing systems [1, 58]. These document processing systems focus on finding solutions for the problem areas of the user-friendly and intelligent system behaviors: document representation, document organization, retrieval space narrowing, storage efficiency, friendly user interface and cooperating user-query answering. Artificial intelligence technologies are commonly applied to develop the intelligent document processing

systems. Knowledge-based techniques and methods for processing and managing documents embody some aspects of human knowledge and expertise to perform tasks, which are ordinarily done by human experts [64]. A typical knowledge-based system contains a knowledge base and an inference engine [10, 18, 28, 32, 33, 50, 60, 62, 64, 81]. The inference engine has a set of functional components that operate upon the knowledge in the knowledge base.

A file system can be viewed as a conventional document processing system. Tree structures are used to organize documents as files, which allow users to use a simple approach to retrieve and browse these documents. Such a file system does not support any intelligent behaviors since it does not have any knowledge base. For example, to categorize a collection of documents, a user could only organize the files manually without any help from the file system. Also, to locate documents of interest, a user must browse through the file system hierarchy to search for these documents, one at a time. Even though there are some tools available to retrieve documents in the current file systems, the retrieval methods are simple and naïve.

Compared with the conventional document processing systems, an intelligent document processing system employs knowledge-based techniques and methods, and inference engine components, which operate on the knowledge, to provide the system with a user-friendly interface and intelligent system behaviors. These intelligent system behaviors include automatic document classification and extraction, intelligent document categorizing and filing, cooperative query answering, etc. These techniques and methods and the capabilities of performing the intelligent system behaviors empower the document processing systems to process and manage documents in the office environments. Thus,

knowledge management becomes the central and essential part of these systems. The representation of the knowledge, the organization of the knowledge, and the use of inference engine components for deriving the applicable knowledge are the central issues of investigating and developing an intelligent document processing system.

1.1 Related Work

To apply AI technologies to ease the document management and retrieval, an effective approach uses a knowledge base to support the conceptual model for describing both documents and other conceptual information. For example, MULTOS [20, 66, 67] employs a knowledge base to support multimedia document filing and retrieval according to various search parameters. Besides the logical and layout structures, defined according to the Office Document Architecture (ODA) standard, it introduces a conceptual structure to capture the semantics of document contents. It divides the filing system into three categories to deal with the small amount of documents by a single user, recently accessed documents and the whole document base. Kabiria[3, 4] employs a knowledge base to store static knowledge, procedure knowledge and domain knowledge. The knowledge helps the system to classify the documents, create the instances and define the roles and the relationships of the documents, which allow users to browse documents at the class and instance levels. RUBRIC [43] employs a knowledge base to build a semantic concept tree structure. Production rules are employed to map the semantic concepts into text patterns. Users can enter a concept to initiate a search process on the concept tree. The production rules are repeatedly applied to browse the concept tree until the related document information is found. The limitation of these systems is as follows: Although

these systems provide approaches to classifying documents based on their conceptual structures (even with the working procedures in Kabiria), these systems fail to provide users with ways of representing a document organization (such as, the folder organization in TEXPROS) from the user's viewpoint. By allowing a user to create their own document organization, it will greatly facilitate the users to manage the documents and to expedite the search of documents by narrowing the searching space. In addition, in contrast with the consistent way used by the TEXPROS, these systems handle the meta-data and documents in different ways, which limit the user' behaviors to browse and maintain the meta-data knowledge.

In general, it is reasonable to assume that knowledge about the working domain is incomplete or unavailable during the system design. Many of these systems are designed for a specific working domain. For example, GRANT [11] is developed to handle a set of documents which are submitted in response to a grant request for scientific research, and therefore the number of documents is limited. Based on the assumption that domain knowledge is available during the system design and the working domain is focussed on flight control, CoBase[7, 8, 9] provides a framework for integrating the class-oriented data grouping and subject-oriented knowledge grouping to support cooperative query answering. The framework is characterized by a three-layer organization. The object layer consists of a database, and the subject layer has a knowledge base. The dynamic linkage between them is handled at the object-subject layer. The limitation of these systems is their domain dependency. These systems are designed based on the availability of the domain knowledge, during the system design. This could limit porting these systems from one

domain to another without any difficulty. This also could limit the user's behaviors to extend the knowledge base to tailor the working domain.

Thesaurus is widely used in both commercial and experimental information retrieval (IR) and document processing systems. A group of research work focuses on the application of thesaurus technology in indexing documents. IS RUSSIA system [13] is a thesaurus-based indexing text retrieval system for automatic processing of Russian official texts. Its search engine includes thesaurus-based components, such as subject headings, a list of described topics, main and specific thematic nodes, mentioned descriptors and relations between topics. The thesaurus for Sociopolitical Life is pre-defined. Upon the arrival of a text, the system will go through four steps: identification of terms in texts, disambiguation of terms using thesaurus projection, construction of thematic nodes and determination of status of the thematic nodes. After the four steps, the thematic representation of the text is created, which will be used for indexing, categorizing and summarizing the text. Users are allowed to retrieve these documents through the index and category. Semantic Forests [59] is a topic-labeling software for text retrieval system. The Semantic Forests system processes a given text to derive a list of topics for the indexing. Before the Semantic Forests processes it, the text must be preprocessed. By filtering the document, only the <TEXT> portion is selected and formatted. Then, the numbers that are represented as words are converted into their numeric values. The next step is to transform multiword units into single tokens, which are derived either by hand or by the frequencies of their occurrences in the text. After the preprocessing, the Semantic Forests reads in every word of the preprocessed document. Using the thesaurus dictionary, the system builds a weighted-topic tree based on the definitions and the

frequencies of the occurrences of the words. Finally, the obtained trees are merged together to form a common graph. Based on the weighted-topic graph, the documents are indexed using the topic lists and are ready to be retrieved. The limitation of these systems is that they need a complete, pre-defined thesaurus to build the target domain. In other words, these systems are domain dependent. It limits the user's behaviors to extend the thesaurus to other application domains. Also, the use of the thesaurus for indexing the texts cannot represent all the information that a document contains, such as its layout information or its document type, etc. It limits the user's retrieval behaviors, such as finding the "student-transcript" documents.

In automatically constructing a thesaurus for text information retrieval, a formal term dependence model, based on relevance judgements [53, 55, 82], is proposed. The frequency of the occurrence of terms in relevant and non-relevant documents is used to estimate the probability of terms, which are similar to the terms stored in documents, appeared in a given query. But the drawback of this model is that it works under the assumption that the degree of full relevance is available. Such full relevance information may not be always available in the real world. Other relevant research efforts include automatic term classification [30, 31], the co-occurrence of terms [51], the similarity matrix between terms [47] and the association thesaurus [29]. These systems are term-collection-dependent. The thesaurus used in these research works is constructed based on probability of having the similarity among the terms. The derivation of the similarity among the terms is based upon the probability present a problem. The thesaurus model is not precise. These systems require a large collection of documents available to build the thesaurus, which is domain-dependent and is not available for most of the working

environments, such as the personal document-processing environment. Besides these systems, the applications of AI technologies to address various problems are commonly used by many researchers in their work.

Sense ambiguity is considered to be an AI problem that occurs in retrieving the document information. To retrieve a document, it is very common that the retrieved documents do not have the exact terms that are used in the query; however, they have the same concepts (senses) represented by these terms (or called words interchangeably). Terms are sometimes ambiguous. The ambiguity of terms can cause the problem of retrieving irrelevant documents. To solve this problem, there are a number of approaches to dealing with the sense ambiguity problem. For example, the Word Experts System [61] uses a procedural approach to handling the sense disambiguation. In the system, each word has its own meaning. When an ambiguous case occurs, messages are exchanged among the related words in an attempt to get the correct sense. The spreading activation and semantic networks are used to solve the sense disambiguation [26]. But these systems only use the information derived from words to solve the sense ambiguity problem. It is very often that the information extracted from documents is very helpful to solve this problem. This information includes the context of a document, its document type, its document semantic range, etc. There are some research efforts which take into account the special data type of words from the documents. Precise understanding of words will greatly enhance, and thus ease, the management of documents and empower the process of retrieving documents. For example, the Semantic Forests can preprocess the words whose type is only a number or a date. No other type of words is considered. However,

there are special data types, which are specific to certain working domains only. So, we propose to extend the domain knowledge to deal with the problem.

Knowledge porting and augmenting are investigated for processing document information. A method for porting and augmenting the thesaurus knowledge is proposed [49] for processing medical information. Three existing medical information systems, such as MeSH, CMIT and SNOMED, use three different thesauri. They intended to integrate these thesauri as a unit, based on the relational schema. But, they did not take other kinds of knowledge into consideration, and did not give a general model for knowledge porting and augmenting.

1.2 TEXPROS Approach

TEXt PROcessing System (TEXPROS) is an intelligent knowledge-based document processing system [34, 37]. It automates the process of processing, managing and retrieving documents for the general working domain such as in an office environment. TEXPROS has a powerful knowledge base coupled with an inference engine. It has several functionality components, including the SCAN/OCR, classification, extraction, filing, retrieval process, browser and synthesizer. It employs a dual model [44, 45] to capture the conceptual information from documents and manage these documents. The SCAN/OCR [37, 77] component is employed to deal with the hard copy documents. Upon the arrival of any document, this component drives the scanner to scan the hard copy document, which is saved as an image file. The OCR software is used to read text from the image files. This step is omitted if the document is already in an electronic text format, for example email.

The classification component [21, 22, 23, 24, 78] is used to classify documents into different document types based on their layout structures and conceptual structures. (Each type is defined in terms of a frame template.) This component creates a mapping block graph for each of the documents using its block structure, which is the OCR's reading result. Based on the knowledge (or a block of knowledge) from each frame template, the classification component classifies a document by matching its block structure graph against each frame template. By classifying a document, we mean to find its document type, and therefore, the frame template of its type is identified.

The extraction component [21, 25] is employed to extract key information from documents. Given a document, the frame template of its document type is identified. Based on the definition of the frame template, TEXPROS will extract information from the document to create a frame instance. A frame instance of a document consists of key information pertinent to the users. It can be considered as an index of the document.

When receiving the frame instance of a document, TEXPROS will invoke the automatic filing [14, 15, 16, 17, 34, 83, 84, 85] component to file the document into the corresponding folders. The user constructs a folder organization to represent his/her view of his/her own document organization. A folder organization consists of folders, each of which is associated with a criterion specified in terms of a predicate. This predicate is defined to govern the kind of documents that will be deposited into its associated folder. Therefore, this automatic filing procedure is a predicate driven process. In fact, the storage is a three-leveled architecture [15, 16, 17]. At the bottom level, it consists of an original document base. At the top level, a logical representation of the storage is created, which is called the folder organization. The intermediate level is a frame instance base,

which is a depository of all the frame instances. Each frame instance has a pointer pointing to its corresponding original document, which is stored in the document base. The folders in the folder organization contains only pointers pointing to the frame instances, which qualify their associated predicates.

Given a three-level architecture of the storage, documents and information from the documents can be retrieved using the retrieval component [35, 38, 39] and browsing component [34, 35, 36, 38, 71, 72, 73]. The retrieval component provides users with a formal query language. This system is capable of processing incomplete and imprecise queries. Without a familiarity with the meta-data, database itself, or the formal query language, a user can use the browsing component to submit topics of a vague query. When the system generates a null answer, the generalizer sub-component will be employed to give users cooperative answers.

The synthesizer component [35, 74, 75] is used to synthesize information obtained from the related documents. For example, after finding all the related documents about the meeting schedule of a user, the synthesizer component is able to pack together all the information about these meetings.

To support these functional capabilities of TEXPROS, a powerful knowledge base is presented to be an organization of the system catalog, the frame template/instance base, the thesaurus, the semantic range, the DataDomain, the registration center, etc. Therefore, the knowledge base plays a major role in processing and retrieving documents by supporting the functionality of the inference engine components. Representing and managing this knowledge become essential and critical issues in the success of the

development of TEXPROS. This motivates us to investigate the knowledge management for TEXPROS.

In dealing with the knowledge management for TEXPROS, we focus on the following concerns:

- The use of various kinds of knowledge to support the intelligent behaviors of the inference engine components.
- The use of these knowledge to support all the inference engine components. From the integration standpoint, we investigate the representation and organization of the knowledge for improving the system performance and maintaining the system consistency.
- TEXPROS is an intelligent document processing system for the general working domain. We investigate and develop a method of knowledge management in such a way that the system can be applied to different working domains. The extension of the domain knowledge, and the knowledge porting are to be investigated.

1.3 Organization of this Dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we briefly introduce the TEXPROS document model and then describe the system catalog structure, frame template base and frame instance base. In Chapter 3, we investigate various problems of the thesaurus model of TEXPROS, and then propose their solutions. The semantic range and the usage of semantic range evaluation for solving sense ambiguity are presented in Chapter 4. In Chapter 5, the structure of DataDomain and DataDomain Agent are

presented. In Chapter 6, A KeyTerm Transformation Component is presented to cooperate with thesaurus, semantic range and DataDomain, and to solve the general KeyTerm transformation problem. In Chapter 7, we investigate the use of existing techniques to implement the approximate term matching in TEXPROS. The registration center is proposed in Chapter 8 for solving the knowledge porting problem of TEXPROS. Finally, the conclusion and future research work are described in Chapter 9.

CHAPTER 2

SYSTEM CATALOG

In this chapter, we shall introduce an intelligent TEXt PROcessing System. We then present an extension of the System Catalog and the implementation of a frame template and a frame instance bases.

2.1 TEXPROS Document Model

The document model for TEXPROS employs a dual modeling approach for storing, classifying, categorizing, filing, browsing and retrieving, and reproducing documents, as well as extracting, browsing, retrieving and synthesizing information from a variety of documents of a pre-defined application document [34, 37, 44, 45]. This model consists of two hierarchies: a document type hierarchy and a folder organization. The document type hierarchy (DTH) depicts the structural organization of the documents and the folder organization represents the user's document filing system in the real world.

In a user's working environment, by identifying common properties for each kind of document type, documents are partitioned into different classes. Each document class is represented by a frame template, which describes the common properties in terms of attributes of the documents of the class and is referred to as the document type of that class. 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 (DTH). The frame templates, which are the members of the document type hierarchy, 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 [63]. Figure 1 shows a sample document type hierarchy for an office environment.

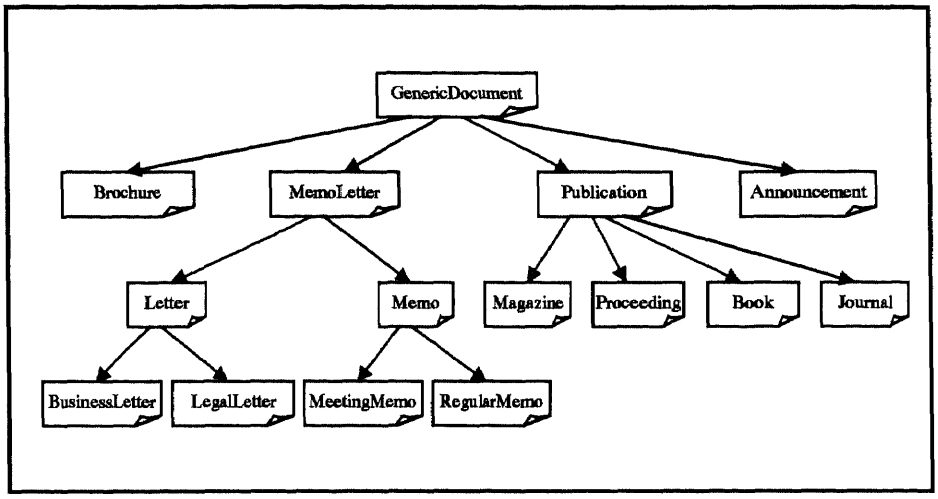
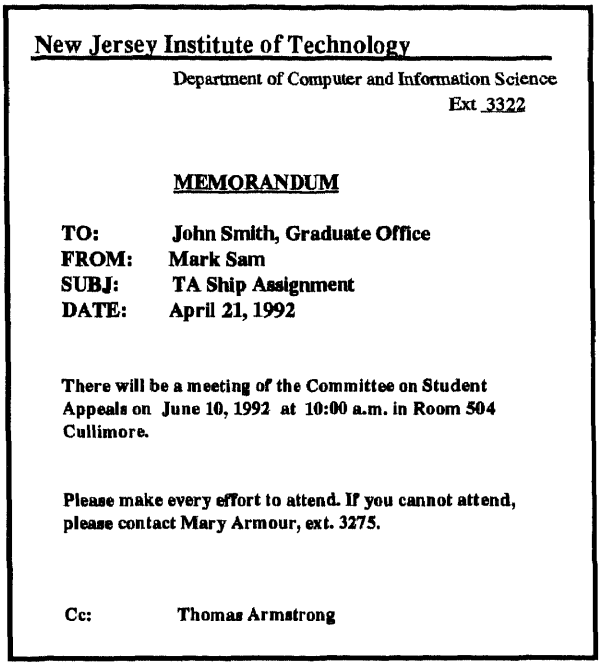


Figure 1 A Sample Document Type Hierarchy

Upon the arrival of a document, the classification component of TEXPROS recognizes its document type (frame template). Based on its frame template, the significant and static information is extracted from this document to yield a synopsis of the document, which we called a frame instance. The frame instance of a document is an instantiation of that particular frame template. Figure 2 shows the frame template and the frame instance of a meeting memo document.

After extraction, the frame instances of documents of different document types will be saved into the instance bases. However, through the automatic filing process, the frame instances are filed into the folder organization [15, 16, 17], by depositing the frame instances into folders if they qualify the criteria of the folders. In fact, the folder

organization contains only pointers, which point to the locations where the frame instances are deposited in the frame instances base.



New Jersey Institute of Technology

Department of Computer and Information Science
Ext 3322

MEMORANDUM

TO: John Smith, Graduate Office
FROM: Mark Sam
SUBJ: TA Ship Assignment
DATE: April 21, 1992

There will be a meeting of the Committee on Student Appeals on June 10, 1992 at 10:00 a.m. in Room 504 Cullimore.

Please make every effort to attend. If you cannot attend, please contact Mary Armour, ext. 3275.

Cc: Thomas Armstrong

(a)

Sender			
Receiver			
Cc			
Subject			
MemoDate			
MtgDescription	MtgDay	MtgDate	
		MtgTime	
	MtgPlace		
	Synopsis		
Remark			

(b)

Sender	John Smith		
Receiver	Mark Sam		
Cc	Thomas Armstrong		
Subject	TA Ship Assignment		
MemoDate	April 21, 1992		
MtgDescription	MtgDay	MtgDate	June 10,1992
		MtgTime	10:00 a.m.
	MtgPlace	Room 504 Cullimore	
	Synopsis		
Remark	If can not attend, contact Mary Armour, ext 3275		

(c)

Figure 2 (a) An Original Meeting Memo Document (b) Its Corresponding Frame Template (c) Its Corresponding Frame Instance

The system provides users with the flexibility for creating their folders, which can be naturally organized as a folder organization. A folder can be considered as a particular set of frame instances. The frame instances can be homogeneous or heterogeneous. That is, the frame instances in a folder may be over different frame templates. Frame instances are grouped into a folder on the basis of its user-defined criteria, specified as predicates, which determines when a frame instance belongs to a folder.

To cope with file organization and to automate document filing (i.e., placing an incoming frame instance into an appropriate folders), we implement the logical folder organization using an agent-base architecture. Each folder is monitored by a filing agent. Each agent has its criteria and private data structures for holding the frame instances, and operations for manipulating the data structures. The criteria are used to govern the placement of a frame instance in appropriate folders. The repository contains frame instances that satisfy the agent's criteria, but do not satisfy its children's criteria. The agents communicate with each other through message passing. Figure 3 shows a sample folder organization.

The original documents and their related frame instances are stored in the original document base and frame instance base, respectively. In TEXPROS, the storage architecture is organized into three levels: at the first (lowest) level is an original document base; the second level is a frame instance base, and the folder organization is at the third (highest) level. Based on the user's viewpoint, the documents are categorized to enhance greatly the retrieval and browsing performance.

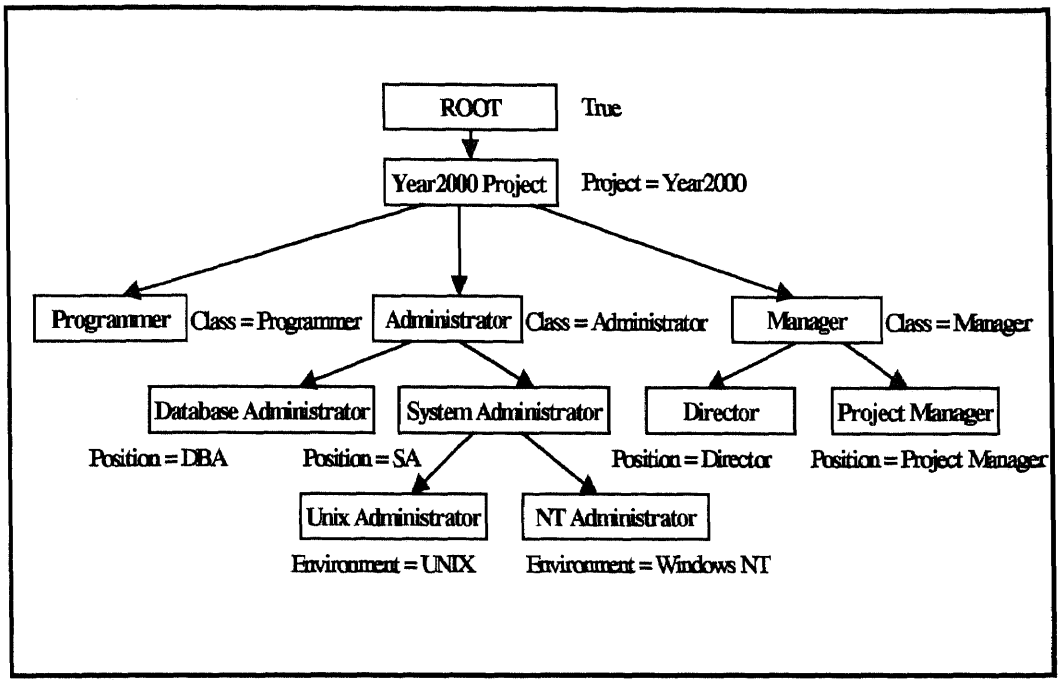


Figure 3 A Sample Folder Organization

2.2 System Catalog Structure

System Catalog is an important part of the knowledge base of TEXPROS. It is employed to describe the meta-data of the actual folder organization and document type hierarchy, which are combined to be the dual models of TEXPROS. In TEXPROS, we employ the concept of frame templates and frame instances at the operational level (both are considered as inference engine components) and the system level (both are used to describe the information in system catalog). At the operational level, the concept of frame templates is used to form the document type hierarchy for classifying the given documents; and the concept of frame instances is used to describe the key information of a particular document based on its document type which is specified in terms of a frame template for its class of documents. Similarly, at the system level, the concept of frame templates is

used to classify the information stored in the system catalog. And the concept of frame instances is used to contain the information about the folder organization and the document type hierarchy. This uniform approach to describe both the operational knowledge of the structures and content of documents, and the system knowledge about the classification and the repositories of documents. This consistent approach of representing the documents and the system organization allows the system catalog to be directly stored into the storage base in the same way as the frame instances are stored into the frame instance base. It also gives users a great flexibility to classify, file and retrieve the meta-data information in the same way as the documents are classified, filed and retrieved.

Figure 4 depicts the structure of System Catalog. System Catalog can be considered as a collection of sets of system frame instances, whose types are of the system frame templates. These system frame instances are employed to store the information of the folder organization and the document type hierarchy. The frame instances of the type SYSFOLDERS contain the information of each folder in terms of the folder name, and its filing predicate and threshold. It also describes the types of frame instances in the folder in terms of frame templates. They are specified in terms of the attributes "FTNames" and "FrameInstanceIDs", which are repeatable attributes.

A set of frame instances of the type SYSFOLDERHIERARCHY is used to describe the structure of a folder organization. The repeatable and composite attribute "Depends_On" is used to specify the parent folders of a folder in terms of the parent folder name, and their linkage information in terms of the link type with a label. The repeatable and composite attribute "Parents_Of" is used to specify the children folders of a

folder and their linkage information. Specifically, the ROOT of the folder organization is a root folder, which has no parent folder. A leaf folder is a folder which has no child folder.

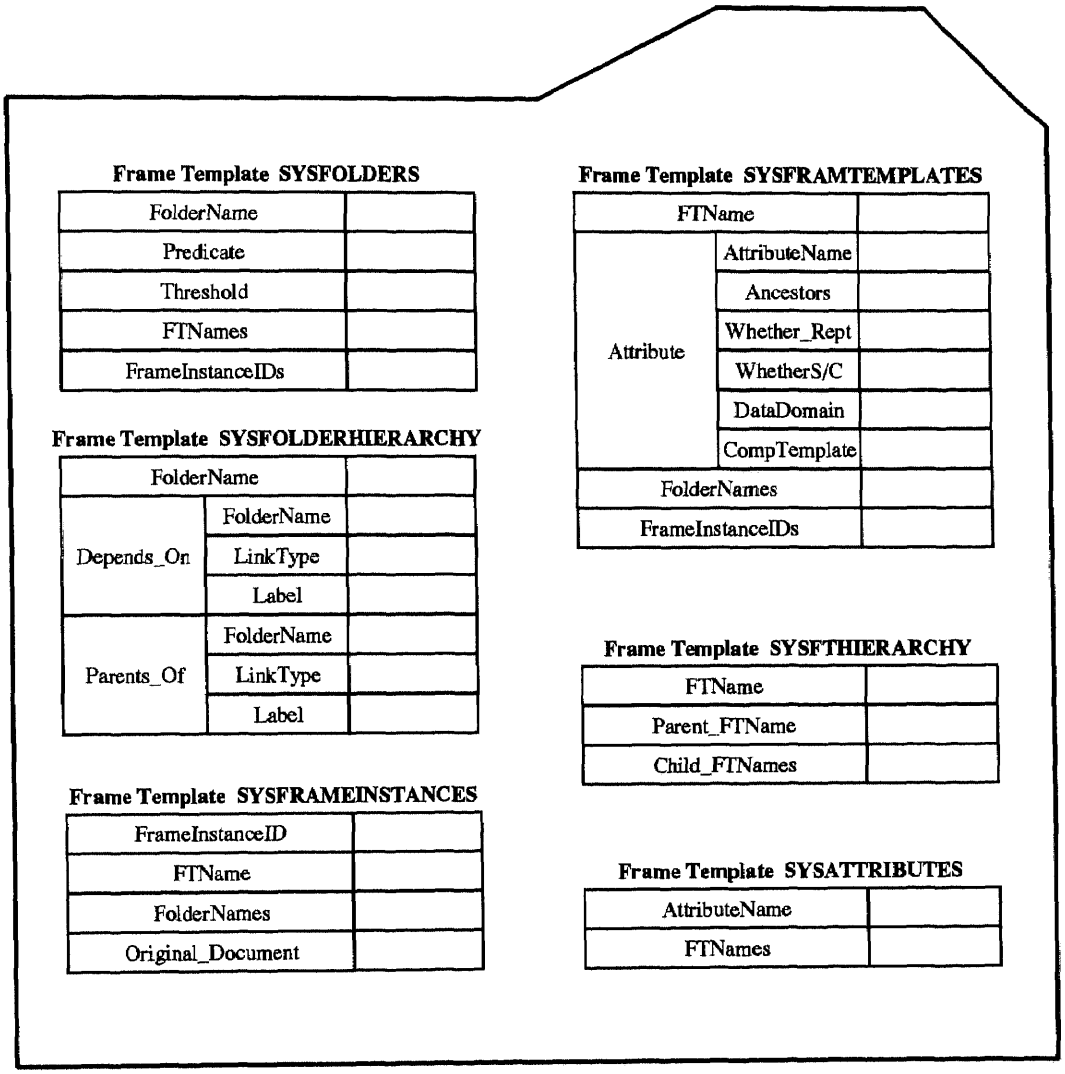


Figure 4 System Catalog Structure

A frame instance of the type SYSFRAMETEMPLATES describes a frame template, which is defined by a set of attributes. So, in this type, a repeatable and composite attribute "Attribute" is used to describe the detailed information of each attribute

contained in the frame template, "FTName". The detailed information includes the name of the attribute, "AttributeName", the ancestor attribute names, "Ancestors" of this attribute, whether this attribute is repeatable, "Whether_Rept", whether this attribute is a composite attribute, WhetherS/C", the DataDomain for specifying the domains of the values of these attributes, and the user-defined composition template. This frame template also has an attribute "FolderNames" to describe the information about the folders over this frame template and an attribute "FrameInstanceIDs" to specify all the frame instances whose type is of this frame template. These two attributes are repeatable.

A set of frame instances of the type SYSFTHIERARCHY describes the structure of a document type hierarchy. For each frame template, "FTName", it stores its only parent frame template, "Parent_FTName", and a number of its children frame templates, "Child_FTNames". Here we should note that there is only one frame template that has no parent frame template. It is the ROOT of the document type hierarchy. A set of frame instances of the type SYSATTRIBUTES gives all the frame templates, "FTNames", which have the attribute, "AttributeName". A set of frame instances of the type SYSFRAMEINSTANCES describes the folders "FolderNames" which have the frame instance, specified in terms of "FrameInstanceID" of the type "FTName. It also specifies the original document from which this frame instance is extracted. This is not for the frame instance base. It only stores the information of the relationship between this frame instance, its type, the folders where it is deposited, and the original document it represented.

2.3 Frame Template Base

TEXPROS uses the uniform approach to handling the operational knowledge about the documents and the system information about the folder organization and the document type hierarchy. In other words, the concepts of frame templates and frame instances are used at the operational and system levels. For implementing the TEXPROS prototype, at the lowest level, the relational database is used as storage base. In the remaining chapter, we will discuss the construction of a frame template base and a frame instance base, based upon this storage base, so that we can store and manage all the meta-data knowledge information and document information to support all the inference components.

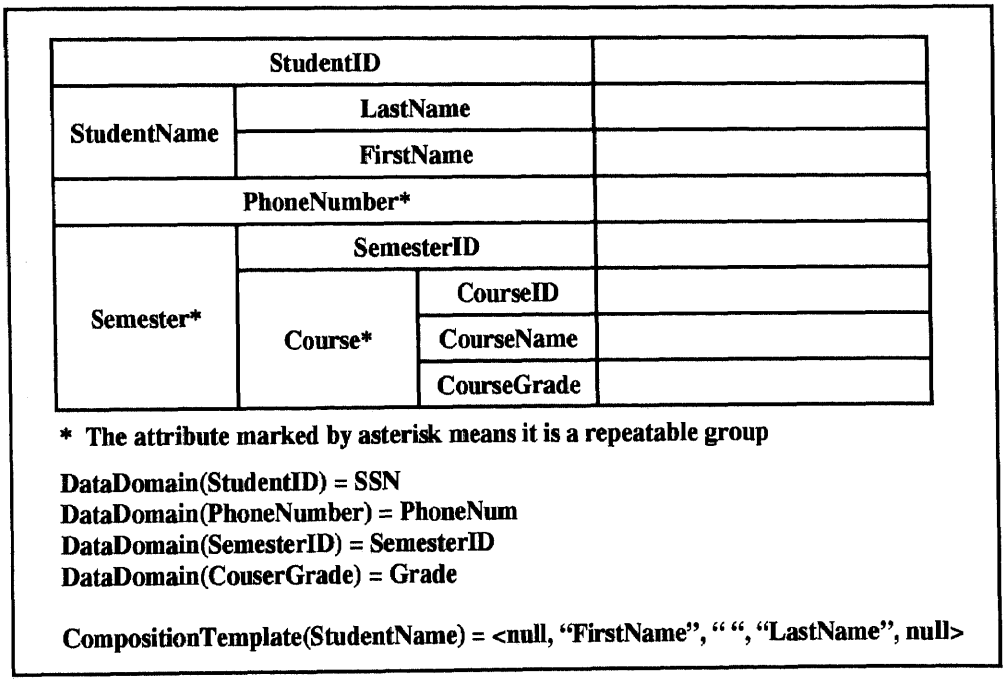


Figure 5 Frame Template "Transcript"

Conceptually, a frame template consists of a set of attributes describing the characteristic properties of a kind of documents. Based on this conceptual definition, it

seems that a frame template can possibly be mapped directly into a relational database table. In implementing the frame templates, we found that, in additional to the attribute names, they must include detailed information that is also very critical to users.

Consider an example in Figure 5. It is a frame template for "Transcript" documents. As shown in the figure, the definition of a frame template is far more complex than a relational database table. The first issue is that the composite attribute is widely used in the frame template definition in TEXPROS. For example, in frame template "Transcript", there is a composite attribute "StudentName", which consists of two child attributes, "LastName" and "FirstName". By means of the concept of composite attribute, users are allowed to specify an aggregate information, which is a block of information closely related to each other. Users are also allowed to define multi-level composite attributes to structure the composite attributes and attributes in a hierarchical structure. For example, The composite attribute "Course" is a child of the composite attribute "Semester", which consists of attributes (called atomic or simple attributes) "CourseID", "CourseName" and "CourseGrade". But the relational database doesn't support the concept of composite attributes. The second issue is that the concept of repeatable attributes is used to specify more than one value for an attribute. For example, in the definition of the "Transcript" frame template, the attribute "PhoneNumber" is a repeatable attribute. This anticipates that some transcript documents may contain more than one student phone number. For this case, we may call it a multi-valued attribute. It violates the first normal form of the relational database. Moreover, a composite attribute can be a repeatable attribute. For example, the repeatable composite attribute "Semester" means the student transcript contains one or more semesters of grade information. The students may take many courses

per semester. Therefore the "Course" must be a repeatable composite. Thirdly, various important features, such as the DataDomain and the CompositionTemplate, are not supported by relational database. We will present this knowledge in the later chapters. The fourth issue is that the sequences of attributes must be kept in order to fully reproduce the frame templates. It is used to indicate the position of the attributes in the frame template or in the corresponding composite attribute block.

Based on these considerations, it is impractical to map each frame template directly to a relational database table. Instead, we define a relational database schema for the frame template base to include the complete definition of each frame template (that is, the full knowledge about the frame template). This knowledge is managed at the knowledge management level, which stores and reproduces the frame templates when the inference engine components need them. So, the complete details of frame template definitions and the storage details of the frame templates are hidden in the knowledge base, and are transparent to the inference engine components. That is, the components simply see the well-designed frame templates.

The relational schema for the frame template base is as follows:

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

This relational schema is based on the conceptual definition of frame template. It allows the system to store the details of frame template knowledge. According to this schema, each record is for an attribute of a frame template. "Ancestors" is used to reflect the composite attribute structure. "WhetherRep" and "WhetherS/C" are used to indicate whether this attribute is repeatable and whether this attribute is a simple attribute or a

composite attribute, respectively. "Seq" is used to indicate the position of an attribute in the frame template or the composite attribute block to which an attribute belongs. "DataDomain" and "CompTemplate" are used to store the knowledge of DataDomain and composition template. Based on this schema, we can store the complete knowledge of a frame template in the frame template base. Figure 6 shows the representation of the frame template "Transcript" in the frame template base.

FT_Name	Att_Name	Ancestors	WhetherReq	WhetherS/C	Seq	DataDomain	CompTemplate
Transcript	StudentID		N	Y	1	SSN	
Transcript	StudentName		N	N	2		<null,"FirstName","","","LastName", null>
Transcript	LastName	StudentName	N	Y	1		
Transcript	FirstName	StudentName	N	Y	2		
Transcript	PhoneNumber		Y	Y	3	PhoneNum	
Transcript	Semester		Y	N	4		
Transcript	SemesterID	Semester	N	Y	1	SemesterID	
Transcript	Course	Semester	Y	N	2		
Transcript	CourseID	Semester##Course	N	Y	1		
Transcript	CourseName	Semester##Course	N	Y	2		
Transcript	CourseGrade	Semester##Course	N	Y	3	Grade	

Figure 6 Frame Template "Transcript" in Frame Template Base

From this example, we can see that the complete information about the frame template "Transcript" is kept in the frame template base. When the inference engine components need this frame template, we can access the complete information of the template "Transcript" from the frame template base. Using the information of the "Ancestors" and "Seq", we can fully reproduce the frame template structure. Other information, such as the DataDomain and composition template, etc. can also be restored.

2.4 Frame Instance Base

Every frame instance is kept in the frame instance base. Conceptually, a frame instance consists of a set of two-tuples <attribute name, value>. It will be used to store both the system catalog knowledge and the information extracted from documents.

Consider an example of a sample "Transcript" frame instance as shown in Figure 7. The complete information of this frame instance is much more complex than the conceptual two-tuples definition. Firstly, the system has to deal with the representation of composite and repeatable attributes. Analogous to the way of handling the attributes of a frame template, in the case of a frame instance, all the attribute information, including the set of two-tuples <attribute name, value>, will be stored into the frame instance base. Secondly, sequence information is necessary to be stored to fully reproduce the frame instance. Thirdly, the system stores a frame instance as an internal frame instance, which consists of a set of three-tuples <attribute name, original value, key value>, instead of the two-tuple definition of a frame instance. The detailed discussion will be given in the following chapters. Fourthly, the aggregate of values, as a block of values is very common and critical to a frame instance. In TEXPROS, users are allowed to define a composite attribute for specifying an aggregate of the related information as a block. Also, the composite attribute is allowed to be repeatable. So, the same block of attributes may appear in the frame instance a finite number of times. At that time, the attribute name "AttributeName" and ancestors "Ancestors" are insufficient to be the unique key for the values. Additional information is needed to indicate which values belong to the same block (we call it a value block).

ATTRIBUTE			VALUE
StudentID			777987777
StudentName	LastName		Hu
	FirstName		Jason
PhoneNumber			973-5961111
PhoneNumber			9735961112
PhoneNumber			9735961113
Semester	SemesterID		1996 Fall
	Course	CourseID	CIS610
		CourseName	Data Structure
		CourseGrade	A
	Course	CourseID	CIS630
		CourseName	Operating System
		CourseGrade	B+
	Course	CourseID	CIS650
		CourseName	Computer Architecture
		CourseGrade	A
Semester	SemesterID		1997 Spring
	Course	CourseID	CIS601
		CourseName	C++ Programming
		CourseGrade	A

Figure 7 A Sample Frame Instance of Type "Transcript"

For example, in the example of the "Transcript" frame instance, the "Semester##Course##Grade" of the CourseIDs "CIS610", "CIS630", "CIS650" and "CIS601" are "A","B+","A" and "A". The annotation of these courses is "Semester##Course##CourseID". These grades and courses have the same attribute names and ancestors, "Semester##Course##Grade" and "Semester##Course##CourseID"

respectively. Without the value block information, we have no way to retrieve in which course the student received a grade of "B+". So, the value block information is kept to indicate which values are in the same block. For example, the values "CIS630", "Operating System" and "B+" are in the same value block.

Based on these considerations, we define a relational database schema for the frame instance base to represent all the frame instances. The precise information which is represented by the frame instance bases are managed at the knowledge management level, and the frame instances could be reproduced when they are needed by the inference engine components. Analogous to the frame templates, all the detailed information is hidden from the inference engine components. The components access only the actual frame instances, each of which corresponds to a document in the original document base.

The relational schema for the frame instance base is:

(FI_ID, Att_Name, Ancestors, Orig_Value, Key_Value, Seq, End_Seq).

This schema contains the conceptual definition of frame instance. Each record corresponds to an attribute-value pair of a frame instance. FI_ID is the unique identity for each frame instance. Att_Name and Ancestors are for the attribute name and their corresponding ancestors. Key_Value is the internal representation of the Orig_Value, which will be discussed in the following chapter. Seq and End_Seq are used to specify the sequence information and value block information. There are some default rules for the "Seq" and "End_Seq". If an attribute is not a composite attribute, then the "End_Seq" is marked as -1. If an attribute is a composite attribute, then the "Seq" and "End_Seq" are marked as the sequence of the first value and the last value that are in the range of this composite attribute value block. Using this approach, the information about blocks of

values is completely kept. Also, this approach can be used to keep the multi-level value block information and repeatable value block information. Figure 8 shows the complete information of the sample "Transcript" frame instance, which is shown in Figure 7, in the frame instance base.

Based on this example, most of the information of a frame instance kept in the frame instance base is hidden from the inference engine components. When the inference engine components can access a frame instance of all the relevant attribute-value pairs from the frame instance base by means of the frame instance ID. Then the information of the "Seq" and "End_Seq" is used to fully reproduce the frame instance structure.

Also, in frame instance base, the usage of "Seq" and "End_Seq" gives another strong support to the vague query retrieval. Originally, upon the arrival of a query, the browsing or retrieval component can retrieve only at the document level. But with the help of "Seq" and "End_Seq", the retrieval process can operate precisely at a block level. For example, let consider a query "CIS601 with A". At the frame instance level, without "Seq" and "End_Seq", the system may find a bundle of transcript documents of those students, who took CIS601 and received all possible grades ranging from A to F, but had an "A" in one of the other courses. On the other hands, at the block level, with the "Seq" and "End_Seq", the system will find CIS601 is at sequence 18 and an "A" at sequence 20, which are both in the same block "Course", whose sequence ranges from 18 to 20. It should be noted that the relationship of values in the same block is stronger than those at the frame instance level, and therefore they fit better the user's retrieval request.

FI_ID	Att_Name	Ancestors	Orig_Value	Key_Value	Seq	End_Seq
10001	StudentID		777987777	777-98-7777	1	-1
10001	StudentName		Jason Hu	Jianshun Hu	2	3
10001	LastName	StudentName	Hu	Hu	2	-1
10001	FirstName	StudentName	Jason	Jason	3	-1
10001	PhoneNumber		973-5961111	973-5961111	4	-1
10001	PhoneNumber		9735961112	973-5961112	5	-1
10001	PhoneNumber		9735961113	973-5961113	6	-1
10001	Semester				7	16
10001	SemesterID	Semester	1996 Fall	1996F	7	-1
10001	Course	Semester			8	10
10001	CourseGrade	Semester##Course	CIS610	CIS610	8	-1
10001	CourseGrade	Semester##Course	Data Structure	Data Structure	9	-1
10001	CourseGrade	Semester##Course	A	A	10	-1
10001	Course	Semester			11	13
10001	CourseGrade	Semester##Course	CIS630	CIS630	11	-1
10001	CourseGrade	Semester##Course	Operating System	Operating System	12	-1
10001	CourseGrade	Semester##Course	B+	B+	13	-1
10001	Course	Semester			14	16
10001	CourseGrade	Semester##Course	CIS650	CIS650	14	-1
10001	CourseGrade	Semester##Course	Computer Architecture	Computer Architecture	15	-1
10001	CourseGrade	Semester##Course	A	A	16	-1
10001	Semester				17	20
10001	SemesterID	Semester	1997 Spring	1997S	17	-1
10001	Course	Semester			18	20
10001	CourseGrade	Semester##Course	CIS601	CIS601	18	-1
10001	CourseGrade	Semester##Course	C++ Programming	C++ Programming	19	-1
10001	CourseGrade	Semester##Course	A	A	20	-1

Figure 8 A Sample "Transcript" Frame Instance in Frame Instance Base

CHAPTER 3

THESAURUS MODEL

A thesaurus consists of a collection of sets of items (phrases or words) and relations between these items. It is widely used in the area of document information retrieval [13, 29, 47, 51, 59, 30, 82]. Using a thesaurus, both exact word matching and sense matching can be used to determine the exact internal representation of indexed terms. Therefore, a good thesaurus model will improve the retrieving performance of the system. There is a great deal of research and commercial IR products that use a thesaurus. In [35], a basic thesaurus model for the TEXPROS was proposed and used. But, this thesaurus model has several shortcomings.

- The thesaurus model, which is proposed in [35] does not considered the difference between the general domain and the personal working domain of thesaurus, which is essential for the system to handle the thesaurus knowledge extension.
- This thesaurus model has difficulty keeping the information in the thesaurus consistent. This model does not support, and therefore can not provide such operations as merge, delete, and modify the terms and the synonymous groups in the thesaurus, which are important to users.
- This thesaurus model considers only the simple attribute cases without taking composite attribute cases into consideration. And in fact, this thesaurus model will fail when handling the composite attributes.
- This thesaurus model ignores the sense ambiguity.

Although it is inadequate to be used in the real world, this basic thesaurus model gives us an insight and a clue for defining a reasonable and powerful thesaurus component that is feasible in our application. In this chapter, we present a way to create an enhanced two level thesaurus model for TEXPROS. In contrast to the basic model, the enhanced model separates the thesaurus of a specific working (application) domain from the general working (application) domain. This allows users to easily extend the thesaurus knowledge for specific working domains. The enhanced model supports the operations, such as, modify, delete and merge operations, etc. We will also discuss how to solve the information consistency problem during the application of these operations. Some other problems existing in the basic model, such as the composite attribute problems and sense ambiguity problems etc., will also be solved in this enhanced thesaurus model.

Synonym is the most important part in the thesaurus of TEXPROS. It refers to a set of words that have the same meaning. They are exact and reliable. During processing documents, synonyms from the thesaurus base are often used for classifying, filing and retrieving documents, extracting information from the documents, and browsing through information. In this dissertation, we build our thesaurus model on the synonym relation. In the future research, we will extend the thesaurus model to other relations, such as related terms (RT), narrow term (NT), etc.

3.1 Two Level Thesaurus Model

Synonyms are sets of words or phrases (which are referred to as terms here) that have the same meaning between them. {USA, United States of America, US} is an example of a set of synonyms. There are two kinds of synonyms. One is for the general application

domain and the other is for the specific application domain. The synonyms of the USA are examples of a general application domain, because they are widely applicable everywhere. Most of the synonyms in the thesaurus dictionary are of the general domain. But there is another kind of synonyms, which are of specific domain. For example, in the Computer and Information Science (CIS) Department of the New Jersey Institute of Technology (NJIT) as the specific domain, the name of Professor Peter Ng can be written in various forms, such as {PNg, PANg, Peter Ng, Peter A. Ng, P. Ng}, which is a set of synonyms within the specific domain. Outside the New Jersey Institute of Technology, PNg may not refer to Peter Ng, but Philip Ng. Therefore, this set of synonyms may not be synonyms when they are not within this specific domain.

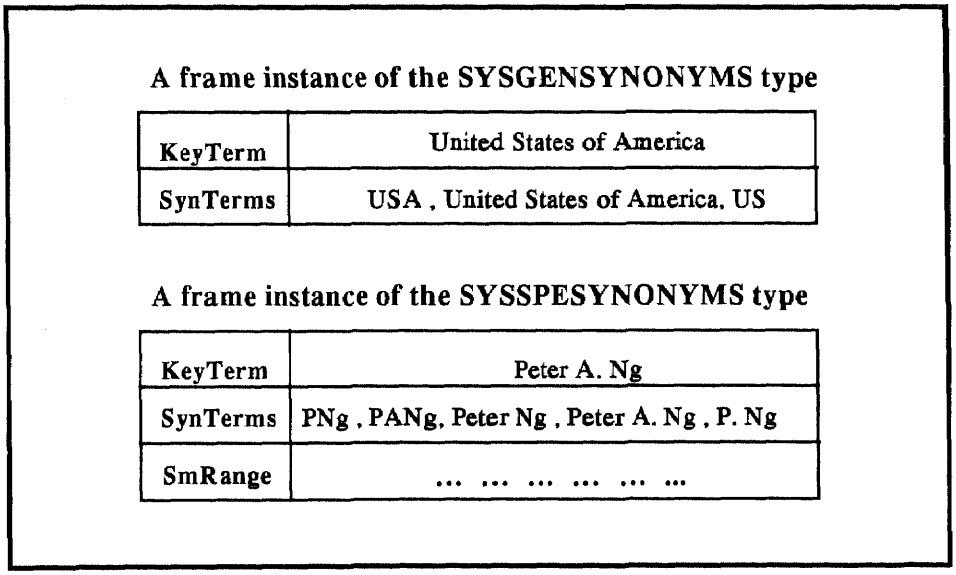


Figure 9 Examples of Frame Instances for Thesaurus Model.

Based on the discussion above, we divide the thesaurus model into two levels of thesaurus. The first level of a thesaurus consists of those synonyms for the general

domain. The second level consists of those synonyms for the specific domain. In TEXPROS, two frame templates are created to represent and store this two-leveled thesaurus (synonyms), as shown in Figure 9. The two system frame templates are SYSGENSYNONYMS(KeyTerms, SynTerms(Repeatable)) and SYSSPESYNONYMS(KeyTerm, SynTerms(Repeatable), SmRange(Repeatable)). These two frame templates have the same structure, except the SmRange. SynTerms is a repeatable attribute, which has a set of words or phrases that have the same meaning. KeyTerm is one of the SynTerms, and is selected to be the unique and internal representation of this synonym group. All the thesauruses (synonyms) are stored as the frame instances, whose type is either of SYSGENSYNONYMS or SYSSPESYNONYMS. The critical difference between these two frame templates is the frame instances. The frame instances of the SYSGENSYNONYMS type are used to store the synonyms of the general domain. The frame instances of the SYSSPESYNONYMS are used to store the synonyms of the specific domain. In frame template SYSSPESYNONYMS, SmRange is used to store the semantic range, which will be discussed later.

The characteristic overview of a two level thesaurus model is depicted in Figure 10. At the level 1, the thesaurus for the general domain can be created before the system is delivered to the users. The synonyms of a general domain can be derived from a thesaurus dictionary. For each group of synonyms, there corresponds a frame instance, whose type is of SYSGENSYNONYMS. A synonym is selected from each group of synonyms to be the default KeyTerm. Such a synonym is a common, unique, the most meaningful or semantically well-expressed term in the group. In fact, the thesaurus for the general

domain is a fixed knowledge base, which supports the entire system. It is predefined and loaded into the system before the system is in use. Users are not allowed to modify, delete or merge the synonyms of this part. The system will automatically handle the control to maintain the consistency of the thesaurus between the two levels.

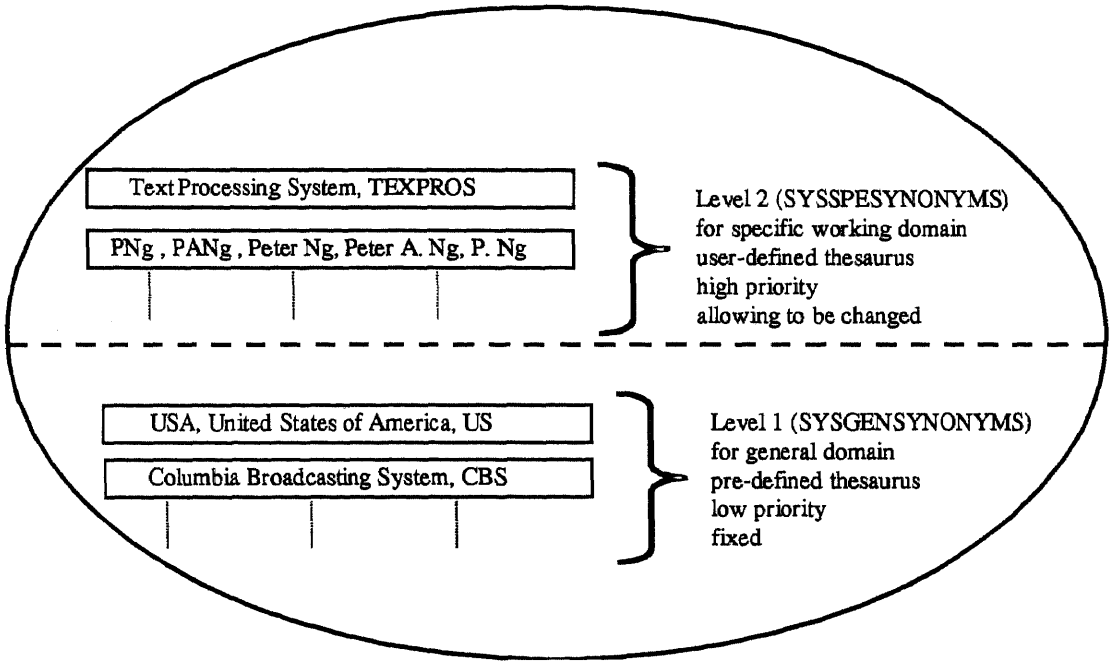


Figure 10 Two Level Thesaurus Model

At the level 2, the complex thesaurus is for the specific domain. This part of thesaurus concerns the different knowledge of different users' application domains. There are two ways for dealing with the default values (knowledge) of this part. Firstly, some of the default knowledge for this part can be created for the user before the system is delivered. This knowledge is obtained from the end users. The synonyms of this type are grouped to form frame instances of SYSSPESYNONYMS type. They will be registered at the registration center (which will be discussed later) of the system. The second way is to

leave this part empty for users to create their thesauruses of specific (application) domain. Users are allowed to extend and update the thesaurus for their specific application domain, when the system is in operation. A friendly interface is provided to deal with all these operations. The internal process of the system will automatically deal with the consistency between these synonyms (which will be discussed in the next section). By means of these operations, users can incrementally empower the thesaurus knowledge and the TEXPROS system.

Since the system allows users to extend and update the thesaurus for the specific application domain, it is possible for the user to create something that conflicts with the thesaurus for the general application domain. Conflicts arise if the same word or phrase, which is occurred in both general and specific domains, has a different meaning. To solve the problem, the system applies the rule that the thesaurus for the specific application domain (at the level 2) always has a higher priority than the thesaurus for the general application domain (at the level 1). In other words, when the system looks in the thesaurus, it will always check the thesaurus of the specific domain first until the KeyTerm is found. Otherwise, the system looks in the general domain thesaurus. So, if there is a conflict, the one for the specific domain, which has the higher priority, will be used. Otherwise, the user has to state explicitly that the thesaurus of the general domain should be used.

This two-level thesaurus allows the system searching for documents using either exact word matching or sense matching. For example, when a user wants to find every document related to a word, he/she is generally not interested in retrieving documents that exactly match the same word, but with the concept (sense) that the word represents.

Without the thesaurus, the system can only apply the exact word matching to give users the documents related to exact word. Obviously, this will not generate a satisfactory result. A thesaurus describes the relationships between the senses of the words, rather than words. With the thesaurus' help, the system can find all the documents related to professor Peter Ng, regardless whether the user gives the query specified in the exact word "P.Ng". And in fact, the sense matching takes the user's specific application domain into account. That is, "P.Ng" is with the CIS department at NJIT. Obviously, the sense matching gives users a lot more satisfactory retrieval results than exact word matching does. The sense matching is also very useful in the document classification component, filing component, etc.

In TEXPROS, the thesaurus is organized as a separated component in the system. So, all the synonyms stored in the component can be used to support the entire system. Also, this component is reusable, which means the synonym groups can be stored once and used by various components of the system.

The thesaurus empowers the TEXPROS in classifying documents into various types, extracting information from documents, automated filing frame instances, retrieving information and documents from the document base. But it also has its performance problem. For example, there are two synonym groups, {CIS, C_IS, CS, Computer and Information Science, Computer Science} and {PNg, PANg, Peter Ng, Peter A. Ng, P. Ng}. Assume that the user wants to find all of the documents, which are related to "PANg <and> C_IS. Because of the "and" operator, there are 5*5=25 possible combinations here. For the worst case, this requires the system going over all documents ten times, even if the

query processing is optimized. The performance will be extremely slow and is unacceptable. To improve the system performance, the notion of KeyTerm is employed.

Definition 1 *(KeyTerm)* A *KeyTerm* is the internal representation of a synonym group. In the TEXPROS' thesaurus model, a synonym group consists of a group of words or phrases in the synonym group that have the same meaning; and different synonym groups cannot share the same meaning. Therefore, the term for a *KeyTerm* must be unique. It is also the unique representation of the meaning for the synonym group.

We should note that, in the thesaurus of the level 1, all the KeyTerms are pre-defined and do not allow users to change them. In the thesaurus of the level 2, users can create their preferred word or phrase as the KeyTerm of a synonym group. Since it is the unique representation of the meaning for its synonym group, in general, KeyTerm should be the most exact and meaningful word or phrase in the group. In addition to its representation, KeyTerm is also used to improve the system performance.

Here, we shall give the algorithm for finding the KeyTerm for a given Term from the two leveled thesaurus model.

Algorithm transformToKeyTerm(term_source)

BEGIN

/* first, check the 2nd level thesaurus and try to find the corresponding KeyTerm*/

$f = \sigma_{term\_source \in SynTerms}(SYSCATALOG(SYSSPESYNONYMS))$

IF f ≠ empty **THEN**

RETURN { sfi [KeyTerm] | sfi = f } ;

/* second, check the 1st level thesaurus and try to find the corresponding KeyTerm*/

$f = \sigma_{term\_source \in SynTerms}(SYSCATALOG(SYSGENSYNONYMS))$

IF f ≠ empty **THEN**

RETURN { sfi [KeyTerm] | sfi = f } ;

/* The failure of finding the KeyTerm in the 1st and 2nd level means that this term has no synonym in the thesaurus. So, return itself. */

RETURN term_source;

END

In the remaining section, we shall define the notion of internal frame instances, and then give an example to show how the KeyTerm is used as the internal representation of value for improving the system performance.

Definition 2 *(Internal Frame Instance)* An *internal frame instance* is the internal representation of a frame instance in the frame instance base. Each internal frame instance is a set of 3-tuples < Attribute, Original Value, Key Value>, where Attribute is an element (attribute name) of a frame template (i.e., the document type); Original Value contains the value extracted from the originial document; and Key Value is either the KeyTerm for the Original Value or the original value itself.

In this example, we have two sets of frame instances as shown in Figure 11 and Figure 12. On the left-hand side are the original frame instances, which are extracted

directly from two given original documents, respectively. On the right-hand side are the corresponding internal frame instances after incorporating the Key Values of the KeyTerms from the thesaurus component, which will finally be stored into the frame instance base. The column of Key Value in the frame instance is transparent to the user. Given a frame instance of a set of 2-tuple (Attribute, Orig Value), there corresponds a frame instance, which consists of a set of 3-tuple (Attribute, Orig Value, Key Value). Each tuple contains an attribute name, its original value and the key value of the original value. For example, "James Bond" is the KeyTerm of the synonym group containing the term "J. Bond". Also, "Computer and Information Science" is the KeyTerm of the synonymous group containing the terms "CS" and "C_IS". So, the system can recognize that both the "CS" and "C_IS" from the original frame instances share the same meaning, based on their corresponding Key Values contained in the internal frame instances. This mechanism provides an effective and useful approach for the TEXPROS' retrieval process.

Frame Instance

Attribute	Value
Sender	P. Ng
Sender's Dept	CS
Receiver	PhD students
Receiver's Dept	CS
Date	May 1st 1998
Content	I will send you an email to let you know your comprehensive exam

Internal Frame Instance

Attribute	Orig Value	Key Value
Sender	P. Ng	Peter A. Ng
Sender's Dept	CS	Computer and Information Science
Receiver	PhD students	PhD students
Receiver's Dept	CS	Computer and Information Science
Date	May 1st 1998	May 1st 1998
Content	I will send you an email to let you know your comprehensive exam	I will send you an email to let you know your comprehensive exam

Figure 11 An Example of Internal Frame Instance

Frame Instance

Attribute	Value
Sender	J. Bond
Sender's Dept	CIS
Receiver	Peter Ng
Receiver's Dept	C_IS
Date	April 15th 1997
Content	Please inform me the date of meeting regarding the department-joint program

Internal Frame Instance

Attribute	Orig Value	Key Value
Sender	J. Bond	James Bond
Sender's Dept	CIS	Computer and Information Science
Receiver	Peter Ng	Peter A. Ng
Receiver's Dept	C_IS	Computer and Information Science
Date	April 15th 1997	April 15th 1997
Content	Please inform me the date of meeting regarding the department-joint program	Please inform me the date of meeting regarding the department-joint program

Figure 12 Another Example of Internal Frame Instance

Consider the previous query example, "PANg <and> C_IS". Before the process of matching takes place, "PANg <and> C_IS" is transformed into internal representation form "Peter A. Ng <and> Computer and Information Science", which is used to match into the Key Values contained in the relevant frame instances. The system can find all the possible combinations by going over all the frame instances once to find these two frame instances, in comparison with twenty-five times required to traverse all the frame instances without using the internal representation. The system performance can be improve greatly at the cost of additional storage for storing Key Values and the look up of the KeyTerms for the terms used in the query.

There is a major difference between the previous basic thesaurus model [35, 37] and this proposed new model. In the previous model, after the transformation, the frame instances remain as sets of 2-tuple value pairs, in which the original values are deleted. In the proposed new model, after the transformation, the frame instances are sets of 3-tuple value pairs. The original values are used to support the thesaurus operations in order to

keep the thesaurus consistent In Section 4.2, we shall discuss the operations and consistency of the thesaurus.

Except for the frame instances, we have similar operations for the folder name, frame template name and attribute name. A frame template, called SYSTERMASSOC, is employed to handle them. The frame template SYSTERMASSOC has three attributes: Term, IndexTm, IndexTmType and IndexTmType can be a "folder" type or a "frame template" type, or an "attribute" type. IndexTm is used to store the original folder name, the frame template name or the attribute name. Term is the KeyTerm corresponding to the IndexTm. The details can be found in the previous work for TEXPROS [35, 37].

3.2 Thesaurus Operations and Consistency

To support intelligent document processing, users require a powerful and complete thesaurus. For a general working (application) domain system, it is almost impossible to pre-store a complete thesaurus before the system is in operation.

In the previous section, a two-leveled architecture for a thesaurus model is presented. At the first level, the thesaurus is for the general domain, which can be constructed based on the thesaurus dictionary. The thesaurus at this level is standard and complete. We can pre-define and pre-store it in TEXPROS as a fixed knowledge base. But at the second level, the thesaurus is for the specific working (application) domain, and is not standard. Users of different application domains may require different thesaurus knowledge at this level. Also, the knowledge at this level is continuously growing and dynamically changing. So, it is almost impossible to pre-define a thesaurus at this level. It is impossible to define it completely even if the users provide substantial information.

Therefore, the system must provide users with a set of operations that allows them to construct and modify their own thesaurus.

The previous thesaurus model [35] does not provide any operations, because it is a one-level thesaurus model, without concerning the impossibility of pre-defining and updating the specific domain thesaurus. Furthermore, the previous thesaurus has some drawbacks on the consistency structure. The operations operated upon the previous model can yield inconsistent results or suffer from information lost, which will be discussed in the following "Computer and Information Science" example.

In contrast to the previous model, the consistency problem does not occur in the newly proposed thesaurus model. Below, we shall define several operations for the thesaurus. And we shall give algorithms with examples to show how to keep the system consistent during the application of these thesaurus operations.

In this thesaurus mode, we define seven operations. They are the Set, Add, Merge, Delete, Split, Destroy and Modify operations. All these operations are only allowed to operate on the thesaurus of the specific domain (i.e., the thesaurus at level 2).

The Set operation allows user to replace the old KeyTerm with a preferred KeyTerm, which is selected from the synonym group. Below is the algorithm for the operation Set.

Algorithm Set(new_term, f)

/* new_term is a preferred term selected by a user, and f is the synonym frame instance of the SYSSPESYNONYMS type, upon which this operation will be operated. */

BEGIN

IF new_term = { sfi [KeyTerm] | sfi = f} **THEN**

RETURN ;

ELSE

KeyChange(new_term, { sfi [KeyTerm] | sfi = f }) ;

set f[KeyTerm] as new_term ;

END

Algorithm KeyChange(new_key, old_key)

BEGIN

frame instance group fog $=\sigma_{\text{Term = old\_key and IndexTmType = "folder"}}$ (*SYSCATALOG* (SYSTERMASSOC)) ;

FOR each frame instance fo in fog **DO**

set fo[Term] as new_key;

frame instance group fmg $=\sigma_{\text{Term = old\_key and IndexTmType = "frame template"}}$ (*SYSCATALOG* (SYSTERMASSOC)) ;

FOR each frame instance fm in fmg **DO**

set fm[Term] as new_key;

frame instance group fag $=\sigma_{\text{Term = old\_key and IndexTmType = "attribute"}}$ (*SYSCATALOG* (SYSTERMASSOC)) ;

FOR each frame instance fa in fag **DO**

set fa[Term] as new_key;

find those 3-tupe value pairs whose keyValue=old_key and put them into value pairs group vpg

FOR each 3-tuple value pair vp in vpg **DO**

set vp[KeyValue] as new_key;

END

The Add operation allows a user to add a new term into a synonym group. Below is the algorithm for the operation Add.

Algorithm Add(new_term, f)

/* new_term is the new term to be added into a synonym frame instance f, and is not defined in the thesaurus at the level 2. f is the synonym frame instance of SYSSPESYNONYMS type. */

BEGIN

$fg=\sigma_{new\_term \in SynTerms}(SYSCATALOG(SYSGENSYNONYMS))$

IF fg ≠ empty **THEN**

put new_term into a term group tg ;

ConsistencyKeep({ sfi [KeyTerm] | sfi = f } , tg }

ELSE

KeyChange({ sfi [KeyTerm] | sfi = f } , new_term) ;

add new_term into SynTerms of frame instance f ;

END

Algorithm ConsistencyKeep(new_key, source_term_group)

BEGIN

FOR each term t in source_term_group **DO**

frame instance group fog $=\sigma_{\text{IndexTm = t and IndexTmType = "folder"}}$ (*SYSCATALOG* (SYSTERMASSOC)) ;

FOR each frame instance fo in fog **DO**

set fo[Term] as new_key;

frame instance group fmg $=\sigma_{\text{IndexTm = t and IndexTmType = "frame template"}}$ (*SYSCATALOG* (SYSTERMASSOC)) ;

FOR each frame instance fm in fmg **DO**

set fm[Term] as new_key;

frame instance group fag $=\sigma_{\text{IndexTm = t and IndexTmType = "attribute"}}$ (*SYSCATALOG* (SYSTERMASSOC)) ;

FOR each frame instance fa in fog **DO**

set fa[Term] as new_key;

find those 3-tupe value pairs whose OrigValue=t and put them into value pairs group vpg;

FOR each 3-tuple value pair vp in vpg **DO**

set vp[KeyValue] as new_key;

END

The Merge operation allows a user to merge two groups of synonyms into one. These two groups of the synonyms must be at the level 2 the thesaurus. Automatically, the KeyTerm of the target group will be the KeyTerm of the new group. Below is the algorithm for the operation Merge.

Algorithm Merge(ft, fs)

/* ft is the target frame instance, and fs is the source frame instance */

BEGIN

term group tg= { sfi [SynTerms] | sfi = fs } ;

FOR each term t in tg **DO**

add t into SynTerms of frame instance ft ;

KeyChange({ sfi [SynTerms] | sfi = ft },{ sfi [SynTerms] | sfi = fs })

delete frame instance fs ;

END

The Delete operation allows a user to delete a term from a synonym group at the level 2 of the thesaurus. This term cannot be the KeyTerm. Below is the algorithm for the operation Delete.

Algorithm Delete(term_source, f)

/* term_source is the term that needs to be deleted from the synonym group. f is the synonym frame instance, whose type is SYSSPESYNONYMS, that the term_source will be deleted */

BEGIN

$fg=\sigma_{term\_source \in SynTerms}(SYSCATALOG(SYSGENSYNONYMS))$

IF fg ≠ empty **THEN**

put term_source into a term group tg ;

ConsistencyKeep({ sfi [KeyTerm] | sfi = fg } , tg }

ELSE

put term_source into a term group tg ;

ConsistencyKeep(term_source , tg) ;

delete term_source from SynTerms of frame instance f ;

END

The Split operation allows a user to split a synonym group into two groups. Below is the algorithm for the operation Split.

Algorithm Split(f , source_term_group, new_key)

/* f is the synonym frame instance of the SYSSPESYNONYMS type to be split. A source_term_group is the group of terms that will be split from the original synonym group to form a new group. A new_key is a term in the source_term_group that is selected as the KeyTerm for the newly formed synonym group */

BEGIN

ConsistencyKeep(new_key , source_term_group) ;

add a new frame instance ff whose type is SYSSPESYNONYMS ;

set ff[KeyTerm] as new_key;

FOR each term t in source_term_group **DO**

add t into SynTerms of frame instance ff ;

delete t from SynTerms of frame instance f ;

END

The Destroy operation allows a user to destroy a synonym group. Below is the algorithm for the operation Destroy.

Algorithm Destroy(f)

/* f is the synonym frame instance, whose type is SYSSPESYNONYMS , that will be destroyed. */

BEGIN

term group tmg = { sfi [SynTerm] | sfi = f } ;

FOR each term t in tmg **DO**

fg=$\sigma_{t \in SynTerms}$(*SYSCATALOG*(SYSGENSYNONYMS)) ;

IF fg ≠ empty **THEN**

put t into a term group tg ;

ConsistencyKeep({ sfi [KeyTerm] | sfi = fg } , tg }

ELSE

put t into a term group tg ;

ConsistencyKeep(t , tg) ;

delete frame instance f ;

END

The Modify operation allows a user to modify a term in the synonym group. Below is the algorithm for the operation Modify.

Algorithm Modify(term_source, term_target, f)

/* term_source is an old term to be modified. A term_target is the term to replace the old term, which is to be modified. The term should be unique (that is, it is never used before) at the level 2 of the theasurus. f is the synonym frame instance of SYSSPESYNONYMS type, upon which, the Modify operation is operated. */

BEGIN

delete term_source from SynTerms of frame instance f ;

add term_target into SynTerms of frame instance f ;

IF term_souce $\neq$ { sfi [KeyTerm] | sfi = f } **THEN**

put term_target into a term group ttg ;

ConsistencyKeep({ sfi [KeyTerm] | sfi = f}, ttg) ;

ELSE

put term_target into a term group ttg ;

ConsistencyKeep({ term_target, ttg) ;

KeyChange(term_target, term_source) ;

set f[KeyTerm] as term_target ;

$fs=\sigma_{term\_source \in SynTerms}(SYSCATALOG(SYSGENSYNONYMS))$;

IF fs $\neq$ empty **THEN**

put term_source into a term group tsg ;

ConsistencyKeep({ sfi [KeyTerm] | sfi = fs } , tsg }// defined before in part 2)

ELSE

put term_source into a term group tsg ;

ConsistencyKeep(term_source , tsg) ; // defined before in part 2)

END

Here, a simple example is given to show how the operation Modify works. This will also show the cooperation of two level thesaurus and consistency control. Consider a frame instance CBS of SYSGENSYNONYMS type, as shown in Figure 13. This is a synonym group in the thesaurus of level 1. This is a fixed thesaurus delivered to the user by the system.

A Frame instance of SYSGENSYNONYMS type

KeyTerm	Columbia Broadcasting System
SynTerms	Columbia Broadcasting System, CBS

Figure 13 "Columbia Broadcasting System" Synonym Group

On the other hand, this user is working closely with a computer and information science at a university. A Computer and Information Science synonym group for his specific domain is created as shown in Figure 14.

Unfortunately, the user makes a small typing mistake, as shown in Figure 14. He typed CBS instead of CIS. Then, the user begins to store documents into the system. The

thesaurus of the level 2 has the higher priority, in comparison with the thesaurus of the level 1. The value "CBS", contained in the frame instances of the documents will be converted to the KeyValue "Computer and Information Science". In addition, in the thesaurus of the level 2, there is no system frame instance which defines the KeyValue for the value "CIS", and therefore, by default, the KeyValue of all the "CIS" values are itself. (In other words, its KeyValue is still "CIS").

A frame instance of SYSSPESYNONYMS type

KeyTerm	Computer and Information Science
SynTerms	Computer and Information Science, CS, C_IS, CBS

Figure 14 "Computer and Information Science" Synonym Group before Modification

Assume that in the frame instance base, there are three frame instances containing the tuple <Corporation Name, Columbia Broadcasting System>; ten frame instances containing the tuple <Department Name, Computer and Information Science>; seven frame instances containing the tuple <Department, CS>; fourteen frame instances containing the tuple <Affiliation, C_IS>; eight frame instances containing the tuple <Company, CBS>; and nine frame instances containing the tuple <Department Name, CIS>.

Applying the KeyChange algorithm for transforming a tuple <Attribute, Value> into <Attribure, Orig Value, Key Value>, it yields their internal frame instances, which have the corresponding tuples <Corporation Name, Columbia Broadcasting System, Columbia

Broadcasting System>, <Department Name, Computer and Information Science, Computer and Information Sciene>, <Department, CS, Computer and Information Science>, <Affiliation, C_IS, Computer and Information Science>, <Company, CBS, Computer and Information Science>, and <Department Name, CIS, CIS>.

Using the given term "C_IS" to search for all the frame instances of documents, which are related to "Computer and Information Science", fails to retrieve the relevant frame instances which have the "CIS" as their KeyValue. Moreover, the frame instances of documents, which have the original value "CBS" will be retrieved. The reason is that the KeyValue for the original value "CBS" is mistakenly defined as "Computer and Information Science", because the term "CBS" should not be one of the SynTerms for the "Computer and Information Science" as the KeyTerm in the frame instance of SYSSPESYNONYMS type. Likewise, the term "CIS" is not one of the SynTerms for the "Computer and Information Science" as the KeyTerm in the frame instance of SYSSPESYNONYMS type. What has happened is simply due to the fact that the term "CIS" is mistakenly entered as "CBS" as one of the SynTerms in the frame instance of SYSSPESYNONYMS type. This type of common mistake can occur easily.

But, in the previous thesaurus model of TEXPROS, after transforming the original values into their corresponding KeyValues, the original values are discarded from the system. (That is, the frame instances contain only the KeyValues after the application of transformation). Therefore, it is almost impossible to correct such a mistake. Based on the previous thesaurus model, in this example, using the given term "C_IS", only thirty nine frame instances are retrieved that are related to "Computer and Information Science". Among this thirty nine frame instances, there are eight frame instances with the original

value "CBS", which are related to "Columbia Broadcasting System", but not related to "Computer and Information Science". Furthermore, nine frame instances with the original value "CIS" are lost, that are related to "Computer and Information Science". Even the thesaurus synonym group is modified correctly, there is no way to know, to identify and then correct the frame instances in the frame instance base, that have "CBS" as the original value.

The newly proposed model allows users to use the operation modify("CIS", "CBS", f) to correct this mistake. The execution of the Modify operation will first find that the current KeyTerm for "CBS" is "Computer and Information Science" from a frame instance of SYSSPESYNONYMS type located in the thesaurus of the level 2. Secondly, it will search through all the frame instances, in the frame instance base, for those tuples with the term "CIS" as their Orig Value, and change their KeyValue to "Computer and Information Science". Thirdly, it will find the KeyTerm for the term "CBS" from a frame instance of SYSGENSYNONYMS type located in the thesaurus of the level 1. The uniqueness property of the KeyTerms (i.e., every term in the SynTerms has a unique KeyTerms) assures that the term "CBS" appears only once in the thesaurus of the level 2. And therefore, there is unnecessary to search the term "CBS" in another frame instance of the SYSSPESYNONYMS type. Assume that the operation finds the "Columbia Broadcasting System" as the KeyTerm for the term "CBS" in a frame instance of SYSGENSYNONTMS type. Fourthly, it will search through all the frame instances, in the frame instance base, for those tuples with the term "CBS" as the Orig Value, and change their KeyValue to "Columbia Broadcasting System". Finally, the operation will change the term "CBS" to the term "CIS", that are obtained initially as a term of the SysTerms of a

frame instance of SYSSPESYNONYMS type in the thesaurus of the level two. After the application of the merge operation, the frame instance of SYSSPESYNONYMS type is modified as shown in Figure 15.

A frame instance of SYSSPESYNONYMS type

KeyTerm	Computer and Information Science
SynTerms	Computer and Information Science, CS, C_IS, CIS

Figure 15 "Computer and Information Science" Synonym Group after Modification

For our example, the frame instance base, after the modify operation, contains the internal frame instances, which have the corresponding tuples <Corporation Name, Columbia Broadcasting System, Columbia, Broadcasting System>, <Department Name, Computer and Information Science, Computer and Information Science>, <Department, CS, Computer and Information Science>, <Affiliation, C_IS, Computer and Information Science>, <Company, CBS, Columbia Broadcasting System>, and <Department Name, CIS, Computer and Information Science>.

Clearly, the newly proposed thesaurus model provides users with a set of powerful operations that operate on the thesauruses of two levels, and thus gives a strong support to the data consistency by allowing to trace the erroneous data with correction capability.

3.3 Thesaurus Support to Composite Attribute

The previous discussion has shown that the new thesaurus model supports the document processing components, such as filing, classification, extraction and browsing etc. But all the discussions so far are based on the simple attributes. The given examples show that this thesaurus model can function well for all simple attribute cases. In this section, we intend to demonstrate that the new thesaurus model can be extended to the application of composite attributes. The frame template uses both the simple attributes and composite attributes to characterize the properties of a document. A composite attribute is a special kind of attribute, which is an aggregate of simple attributes and/or other composite attributes, each of which is, in turn, an aggregate of attributes until all the attributes are of simple attribute type. The use of composite attributes in the thesaurus is not as easy and direct as simple attributes. In this section, we will discuss the simple and composite attributes and their values. Also, we will discuss how the composite attributes can be used in our new thesaurus model.

In TEXPROS, a frame template consists of a set of attributes. For example, a frame template "Memo" consists of seven attributes {Sender, Sender's Dept, Receiver, Receiver's Dept, Date, Content}. These attributes are simple attributes, since none of them is an aggregate of other attributes, which are called its child-attributes. Consider the attribute NameOfApplicant{ Last Name, First Name } in Figure 16. Here we have three attributes, NameOfApplicant, Last Name, and First Name. The attribute NameOfApplicant is the parent-attribute of the other two attributes Last Name and First Name, which are simple attributes since they do not have any child-attribute. The attribute, NameOfApplicant, is a composite attribute, which is an aggregate of the two child-

attributes, Last Name and First Name. By introducing the concept of composite attributes, for this case, we may retrieve the First Name and Last Name of applicants. And the retrieved values have clear semantic meanings. For instance, Peter is the first name and Ng is the last name of an applicant. Hence, the composite attributes are widely used in documents, for example, Name{First Name, Last Name}, BirthDate{Day, Month, Year}. We may view the composite attribute is a virtual container of its child-attributes.

However, there are many problems with using composite attributes in the thesaurus. Consider the frame instance given in Figure 16. In this frame instance, we know that it is related to "Peter Ng", whose KeyTerm should be "Peter A. Ng". But, because the composite attribute NameOfApplicant is an aggregate of its child-attributes "Last Name" and "First Name", which are simple attributes, these two simple attributes have their Original Value "Ng" and "Peter". The KeyTerms for these Org Value "Ng" and "Peter" are themselves, without taking the composite attribute NameOfApplicant into consideration. That means, this frame instance has two Key Values "Ng" and "Peter". Then a problem will arise if a user wants to find some documents related to "P. Ng"; this document would not be retrieved because the Key Value of "P. Ng" is "Peter A. Ng", which is neither "Ng" nor "Peter". So, the system will not retrieve this document. The only way to find this document from the document base is that the user has to give the query "Peter <and> Ng". Then, the system cannot find the documents, which are related to "P. Ng" or "Peter A. Ng". In this case, the thesaurus fails to assist the retrieval component by giving the unified Key Value for any terms used in query. Thus, the system is not flexible as before.

Attribute		Value
NameOfApplicant	Last Name	Ng
	First Name	Peter
....		....

Attribute		Orig Value	Key Value
NameOfApplicant	Last Name	Ng	Ng
	First Name	Peter	Peter
....		....	....

Figure 16 An Example of Composite Attribute

Before resolving this problem, we should investigate the purpose of having the concept of composite attributes. Here is an example of a frame template "Transcript". In TEXPROS, a composite attribute is a virtual container of its child-attributes. It allows users to define the block structure in a frame template. For example, consider the frame template "Transcript" as shown in Figure 17. The composite attribute "Semester" is an aggregate of the child-attributes "SemesterID" and "Course", where "Course" is a composite attribute of its child attributes "CourseID", "CourseName" and "CourseGrade". Here the composite "Semester" is used to specify a group of information together as a block, which is related to a specific semester. Attribute "Semester" itself, it has no value. This is the most common use of the composite attribute. It is a virtual container structure, and has no value itself. It is a mechanism to help users group some related information together. However, there is another way of using the concept of composite attributes, which is referred to as value composition. For example, the composite attribute "StudentName" is not only a virtual structure, but it aligns the values of its child-attributes

"LastName" and "FirstName", and gives this new composed value to "StudentName", which will later be used during the process of filing, browsing and retrieving.

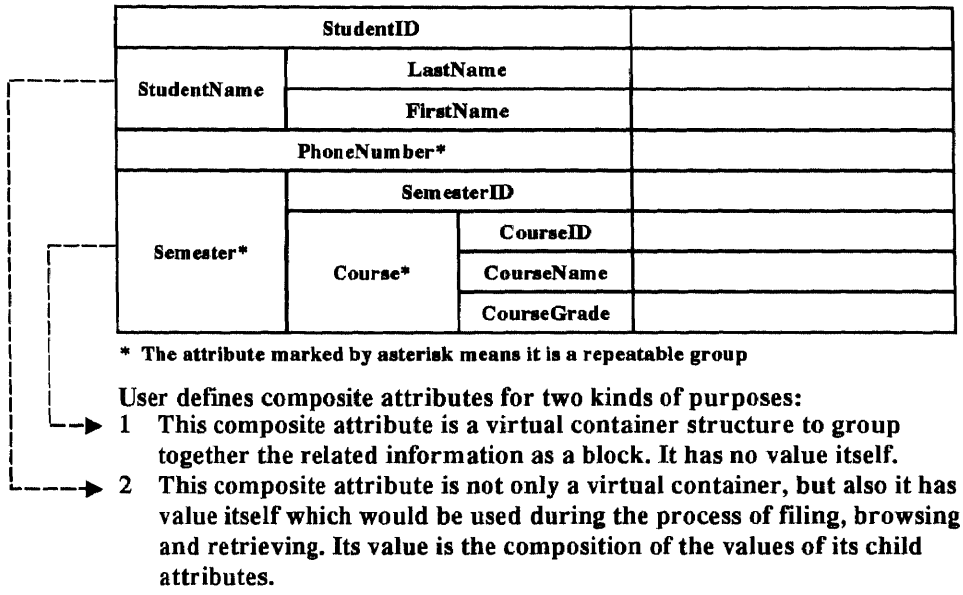


Figure 17 Uses of Composite Attributes

In summary, there are two kinds of uses for the composite attribute in TEXPROS. Firstly, the composite attribute is a virtual container, which could be used to group together all the related information as a block of information. For example, a student may take a number of courses. Each course has it own course id, course name and grade. This block of information can be well represented by Course*, where the asterisk symbol is used to denote this block of attributes {CourseID, CourseName, CourseGrade} is repeatable. Likewise, a student has a complete transcript, which consists of his/her coursework for a number of semesters. However, the composite attributes such as, Semester and Course do not have values themselves. They are only the virtual blocks. Secondly, the composite attribute helps users to align the values of its child-attributes

together to form a value for itself, and we called it the value composition. From this view, if our system can support the value composition and gives the composite attribute "NameOfApplicant" the value "Peter Ng" for the previous example, then the problem will be solved.

Here we shall give several definitions as follows:

Definition 3 *(Simple Attribute)* A *simple attribute* is an atomic attribute, which describes a property of an entity. Assigned to each attribute, there is a value.

Definition 4 *(Composite Attribute)* A *composite attribute* A_0 is an aggregate of a collection of attributes (A_1, A_2, ..., A_n), where A_i (for any i between 1 and n) is either a composite attribute or a simple attribute.

The attributes A_1, A_2 , ..., A_n are called the child-attributes of the composite attribute A_0

Definition 5 *(Composition Template)* For a composite Attribute a1, with n child attributes: ac_1, ac_2, ac_3, ..., and ac_n, the *composition template* ct for a1 is a user-defined sequence $ct = <s_0 , x_1 , s_1 , x_2 , s_2 , \ldots, \ldots, x_j , s_j>$, where s_0, s_1, ... , and s_j are constant strings and $x_{i\,(i=1,2,\ldots j)} \in \{ ac_1, ac_2, \ldots \ldots , ac_n\}$. In other words, the values x_1, x_2, ..., and x_j are concatenated with the constant strings specified as s_0, s_1, ..., and s_j, based on the given template, to form a new string.

Definition 6 *(Value Composition)* A *value composition* for a given composite attribute is a kind of user-defined operation, which composes the values of the child-attributes of a composite attribute. The result of *value composition* operation is a composed-value,

which is a value assigned to that composite attribute as its value. This operation is based on the "composition template" that user defined for that composite attribute.

Definition 7 *(Value of Composite Attribute)* In general, a composite attribute has no value. This composite attribute is only a virtual container of a null value. But, If the composite attribute has a user-defined Composition Template, then this composite attribute has the value, which is obtained by applying the Value Composition operation on the values of its child-attributes.

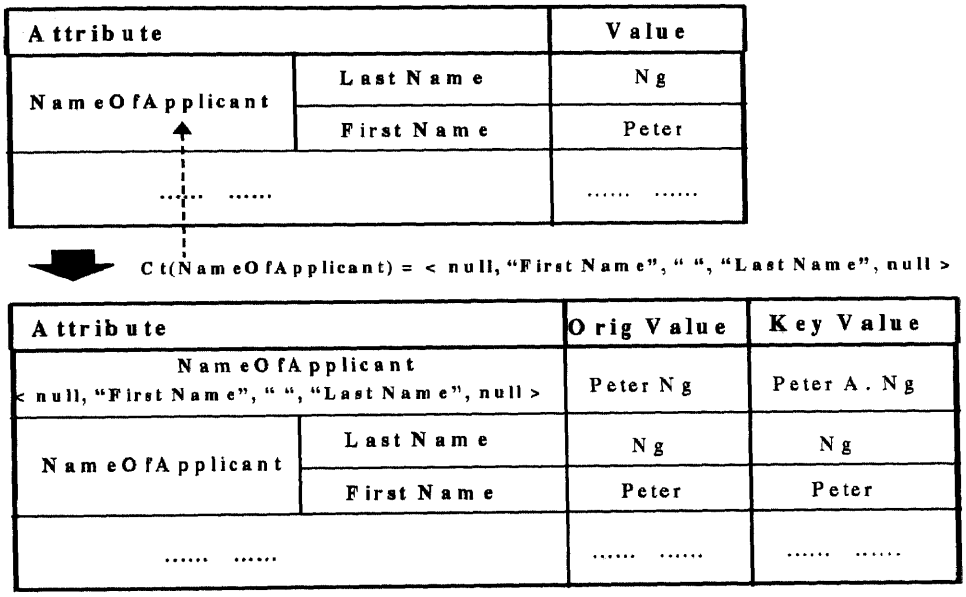


Figure 18 An Example of Value Composition

In Figure 18, given the composite attribute NameOfApplicant, a user can define a composition template for this composite attribute. For example, a user defines this composition template as < null, "First Name", " ", "Last Name", null >. (That means, a string of the first name followed by a blank space and then followed by a string of the last

name.) Then the composite attribute NameOfApplicant has a composite-value, which is the Value Composition of the Original Values of its child-attributes. Also, the composed value will be assigned as the Original Value of the composite attribute NameOfApplicant. Then, using thesaurus, a Key Value, which is related to the composed Original Value, will be found and assigned to "NameOfApplicant".

Having this frame instance, when a user wants to search for all the documents that are related to "P. Ng", then this one won't be missing. We should note that the composite attribute NameOfApplicant can also be used to define the folder predicate for document filing.

When defining the composition template for a composite attribute, a user must decide the following facts:

- Whether the value of a child-attribute should appear in the composite-value.
- The sequence of the values of the child-attributes that appear in the composite-value.
- The constant strings, which are used to connect the values of the child-attributes.

Also, because the composite attribute can be recursively defined, the Value Composition operation supports the recursively defined composite attribute.

Consider the following example. For the composite attribute: Date{Day, Month, Year}, a composition template is defined as < null, "Month", "/", "Day", "/", "Year", null>. It means that the values of its child-attributes will appear in the composite-value in the order as stated in the composite template. The value of "Month" appears first, followed by a special character "/", then followed by the value of "Day" with a special character "/", and ends with the value of "Year". If the frame instance has abstracted

Date{Day=23, Month=2, Year=1998}, after applying the Value Composition operation, which is based on the composition template: < null, "Month", "/", "Day", "/", "Year", null>, then the composite-value for Date is 2/23/1998.

CHAPTER 4

SEMANTIC RANGE

4.1 Ambiguity of Senses

One of the users' requirements for our old thesaurus model is that a group of synonyms is applicable on various different domains if it is applicable in an application domain. There is no ambiguity in our old thesaurus model. This assumption allows us to create and handle easily the thesaurus model. It is a reasonable assumption when the TEXPROS is in the stage of prototyping the system. However, as the system grows in its applications to various functional or application domains, there will be problems, which are caused by the ambiguity of interpreting various terms using the thesaurus, and we call this Ambiguity of Senses.

Consider the frame template Memo, which consists of the attributes "From" and "To". These attributes are referred to as the sender and the receiver(s) of the memo, respectively. However, this template could also use the attribute "Sender" and the attribute "Receiver" instead of "From" and "To", respectively. To allow the attributes "Sender" and "From", and the attributes "Receiver" and "To" to be used interchangeably, the words "Sender" and "From" must appear in the same group of synonyms in the thesaurus. Likewise, the words "Receiver" and "To" must appear together in another group of synonyms. However, there are other frame templates, such as Shipment Invoice, in which the attributes "From" and "To" refer to locations. They could be "Location From" and "Location To", respectively. They could also be "City From" and "City To", respectively. Or, they could be "Country From" and "Country To", respectively. That

means, "From" can be in several different groups of synonyms. Likewise, "To" can also be in the different groups of synonyms. Each has its own semantic meaning. Then, we need to extend the thesaurus model in [35] to handle this sense ambiguity of the attributes "From" and "To.

Here is another example. If "BBC" is an abbreviation of a company "Bernie Business Corporation", then "BBC" and "Bernie Business Corporation" will appear in the same group of synonyms in the thesaurus model. This allows users to use these two terms interchangeably for most of the cases within the application domain of the company. Assume that the British Broadcast Corporation is one of its clients. The term "BBC" is also the abbreviation of the company "British Broadcast Corporation" if the application domain is dealing with news documents. These terms "BBC" and "British Broadcasting Corporation" should also be in the same group of synonyms in the thesaurus. If this is the case, searching for documents related to the company, using the term "BBC", a user will surprisingly find some documents which are related to the news.

Here is the third example. Professor Peter A. Ng is a faculty of the Department of Computer and Information Science (CIS) at NJIT. The term "Peter Ng" (which is Peter A. Ng) is commonly used in most of the documents in the CIS department. Assume that there is also another professor whose name is Peter B. Ng in the Department of Electrical Engineering (EE). Again, the term "Peter Ng" (which is Peter B. Ng) is also commonly used in the documents of the EE department. Consider a user such as the Vice President of Academic Affairs. He knows definitely that "Peter Ng" is either "Peter A. Ng" or "Peter B. Ng" in dealing with the CIS documents or the EE documents. The thesaurus model in [35] fails to deal with this ambiguous term. By grouping all these terms "Peter

Ng", "Peter A. Ng" and "Peter B. Ng" in the same group of synonyms, the system could draw an erroneous conclusion that "Peter A. Ng" is "Peter B. Ng. For this reason, we must separate them into two groups of synonyms. But the thesaurus model in [35] did not allow "Peter Ng" to have two KeyTerms.

In summary, the problem arises from the thesaurus model in [35] because it is based on the assumption that, in the personal environment, a group of synonyms is applicable on various different domains, if it is applicable in an application domain. There is no ambiguity in our thesaurus model. This assumption helps us to create and handle easily our thesaurus model. But in human language, a word can have different meanings. For example, the precise representation of "Peter Ng can be "Peter A. Ng" or "Peter B. Ng", depending on which department is referred to. Since a word has different meanings in different applications, we face another problem: how to recognize which meaning of the word should be used. The answer is the context environment. In reality, a user is usually able to decide the meanings of ambiguous words from reading the document, i.e., from the context environment. Consider the previous examples. For a document of the Memo type, a user will know that the term "From" is the "Sender". In contrast, if the document is of the Shipment Invoice type, then the user will know that the term "From" is "City From". Also, in the example of precise representation of "Peter Ng", the term "Peter Ng" is "Peter A. Ng", provided the document refers to the CIS department. If this document refers to the EE department, then "Peter Ng" is "Peter B. Ng". For the "BBC" example, in most cases, a user uses "Bernie Business Corporation" by default, except that the user will use "British Broadcasting Corporation" when processing the news document.

Unfortunately, there needs a great deal of research to be done on the problem how to capture the precise meanings of words within a given context environment, which is related to the natural language processing in AI. Today, some IR research works [13, 26, 61] are trying to deal with the sense ambiguity problem. They are attempting to use a statistical approach at the word level to solve this problem. They need the pre-defined working domain knowledge or a giant set of testing data to run the statistical approach on it. But this approach doesn't fit for general domain personal document processing systems, like TEXPROS. Also, some useful information at document level will also be helpful to capture the context environment.

In TEXPROS, a thesaurus model is created to determine the precise meanings of the words within the context environment. In TEXPROS, the concept of adopting dual models gives an advantage for simulating the context environment over the other IR systems. These dual models are the Document Type Hierarchy (DTH) and Folder Organization (FO), which are created by users. Users create these dual models to represent their views of the context environment of their personal world. We may be able to find a way to solve the problem of sense ambiguity by taking using these dual models. With this motivation, we introduce semantic range to TEXPROS, based on the concepts of the dual models. The concept of semantic range is employed to solve the ambiguity problem in the thesaurus of the TEXPROS.

4.2 Semantic Range

In the thesaurus model of TEXPROS, a user-defined range of semantic environment for documents is called a semantic range, which is constructed based upon the dual models of

TEXPROS. It is used to specify the valid range of a group of synonyms, and is used for solving the ambiguity problem of the thesaurus. The formal definition of the semantic range is given as follows.

Definition 8 *(Semantic Range)* Given the TEXPROS, a *semantic range* (SR) of two given terms, say T1 and T2, is represented by a 5-tuple SR(T1 -> T2) = <*Folder Range, Frame Template Range, Attribute Range, Value Range, Time Range*>, where the *Folder Range*, specified by a folder name in the folder organization of TEXPROS, is the contents and the documents contained in this folder and its descendent folders. The Folder Range is valid for all the folders if it is empty (null). The *Frame Template Range*, specified by a frame template name in the document type hierarchy of TEXPROS, is the contents and the documents whose type is of the frame template and its descendent frame templates. The Frame Template Range is valid for all the frame templates, if it is empty (null). The *Attribute Range(Ra)*, represented as a Boolean value, determines whether this semantic range is valid for attribute names depending on its truth or falseness. The *Value Range(Rv)*, represented as a Boolean, determines whether this semantic range is valid for values, depending on its truth or falseness. It should be noted that both the Attribute Range and Value Range can be true; and they are meaningless if both are false (Restriction: Ra || Rv == true). The *Time Range* is represented by a starting date and an end date; and it is valid for all the time, if it is a null.

In reality, given a group of synonyms {T1, T2, ...,Tn, K}, in which K is the KeyTerm, a user can define a set of Semantic Ranges SR1, SR2, ..., and SRn, which are

the instances of <Folder Range, Frame Template Range, Attribute Range, Value range, Time Range>, attached to this synonym group. Then we have SR(T1 -> K) = SR1, SR(T2 -> K) = SR2, ... , SR(Tn -> K) = SRn if and only if in the semantic range SRj, Tj and K share the same meaning for any j (1 <= j <= n).

There are two special Semantic Ranges. The first one is *SRF* = <null, null, true, true, null>, which is called *Full Semantic Range*. It covers everywhere at any time. It is the default semantic range for thesaurus terms. The thesaurus of the level 1 has the full semantic range. The second one is *SRN* = NULL, which is called *Null Semantic Range*. It is not valid for any place at anytime. In other words, it is an empty range.

Let us see the example of SR1(US President -> William Jefferson Clinton) = <null, null, false, true, 1993-2000 > in Figure 19. Assume that there is a group of synonyms { US President, Bill Clinton, William Jefferson Clinton }, in which "William Jefferson Clinton" is the key term. The user attaches two semantic ranges to this synonym group, which are <null, null, false, true, 1993-2000 > and SRF, respectively. So, we have SR(US President -> William Jefferson Clinton) = <null, null, false, true, 1993-2000 > and SR(Bill Clinton -> William Jefferson Clinton) = SRF. It means, the "US President" is a value present in a document for the period of the year of 1993 to the year of 2000, and the "US President" and "William Jefferson Clinton" share the same meaning. For "Bill Clinton", because of the full semantic range, it always shares the same meaning with "William Jefferson Clinton".

In the example of SR2(Peter Ng -> Peter B. Ng) = <EE, null, false, true, null>, the semantic range SR2 is attached to the group of synonyms group {Peter Ng, Peter B. Ng}, in which Peter B. Ng is the key term. It means that these two words share the same

meaning if "Peter Ng" is a value appeared in a document, which is in the EE folder or its descendent folders.

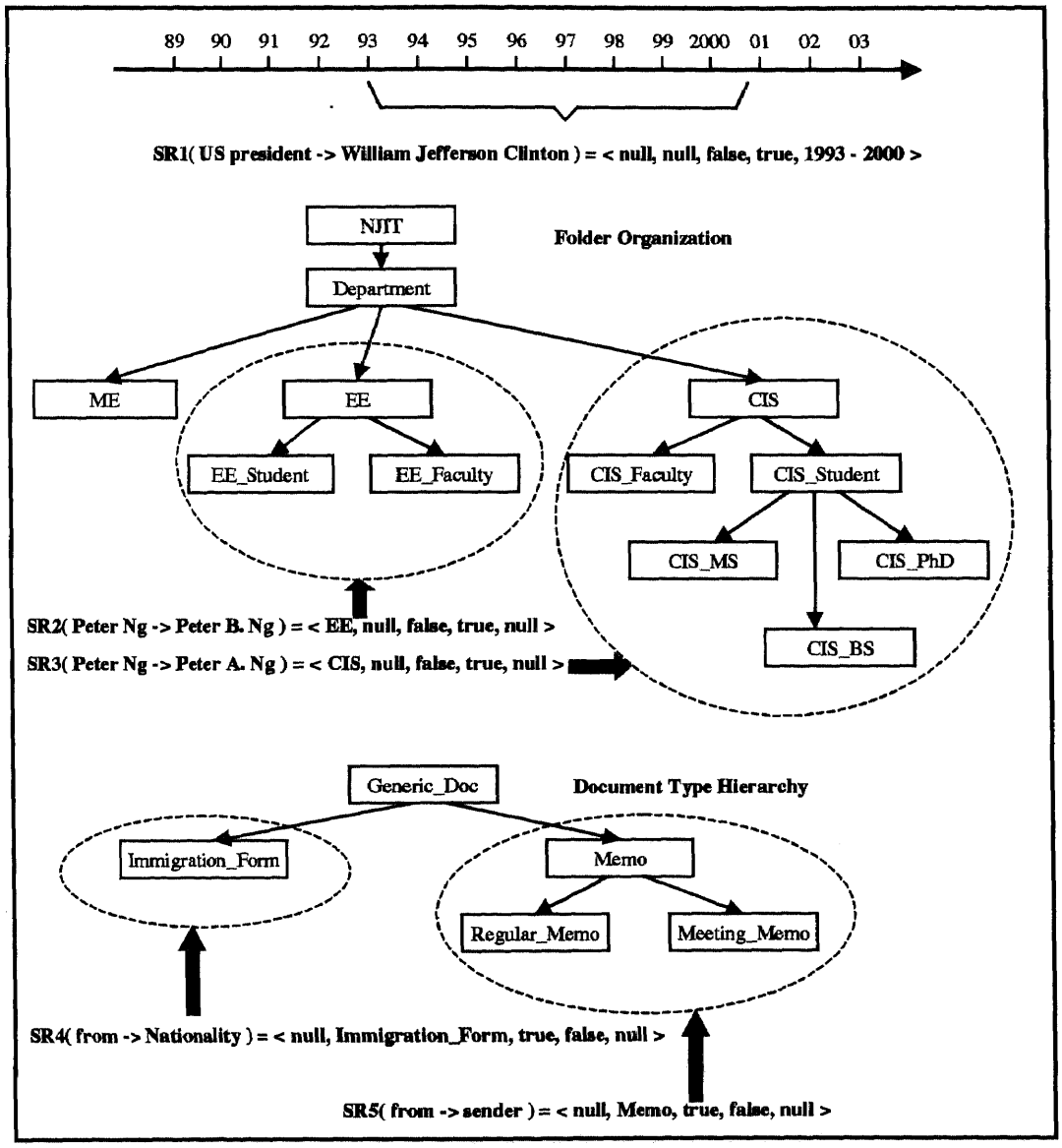


Figure 19 Semantic Range Examples

In contrast, the semantic range SR3(Peter Ng -> Peter A. Ng) = <CIS, null, false, true, null> has the same meanings as for the SR2, and the terms Peter Ng and Peter A. Ng

share the same meaning within the CIS folder and its descendent folders. So, using SR2 and SR3, we can represent that by the value type in folder range "EE", "Peter Ng" is referred to "Peter B. Ng", and in the folder range "CIS", "Peter Ng" is referred to "Peter A. Ng".

In the example of SR4(from -> Nationality) = <null, Immigration_Form, true, false, null>, this semantic range is attached to the group of synonyms {from, Nationality}, in which "Nationality" is the key term. It covers the frame template "Immigration_Form" and all its descendent frame templates. These two words share the same meaning if "from" is an attribute name in the frame template range "Immigration_Form".

In the example of SR5(from -> sender) = <null, Memo, true, false, null>, the semantic range SR5 is the same as the previous semantic range SR4, except that the frame template range of the semantic range SR5 is "Memo". Using SR4 and SR5, we can precisely interpret that the attribute name "from" is referred to the "sender" if the attribute name is in the frame template "Memo" or its descendent frame templates; and is referred to the "Nationality" if it is in the frame template "Immigration_Form" or its descendent frame templates.

There are three operators applicable to Semantic Range. These operators are +, - and &. The operator + is a binary operator for the semantic range. The result of this operation is a semantic range, which is the sum range of the two parameter ranges. The operator - is a binary operator for the semantic range. The result of this operation is a semantic range, which is the range in the first parameter range but not in the second range. The operator & is a binary operator for semantic range. The result of this operation is a semantic range, which is the range in both the first and second parameter ranges.

In the example of SR6 = <CIS, null, false, true, null> + <EE, null, false, true, null>, the semantic range SR6 covers the folders CIS and EE and their descendent folders. Also, it only works for the value type, but not for the attribute names.

In the example of SR7 = <null, Memo, true, false, null>-<null, MeetingMemo, true, false, null>, the semantic range SR7 covers the frame template Memo and its descendent frame templates, except the frame template MeeingMemo and its descendents. Also, it only works for the attribute names, but not for the value type.

In the example of SR8 = <CIS, null, false, true, null> & <EE, null, false, true, null> = <CIS_EE_JOINT_PROGRAM, null, false, true, null>, assuming that the CIS_EE_JOINT_PROGRAM is the only common child folder of both CIS folder and EE folder. The result of this & operation SR8 is the semantic range which covers the folder CIS_EE_JOINT_PROGRAM and its descendent folders.

There are several relationships between Semantic Ranges. They are *cover*, *join* and *disjoin*. If the Semantic Ranges SRi and SRj satisfy the condition SRi+SRj=SRi, then the Semantic Range SRi *covers* the Semantic Range SRj. We also say that the Semantic Range SRj is *covered* by the Semantic Range SRi. Specifically, the Full Semantic Range SRF covers any Semantic Ranges, and the Null Semantic Range SRN is covered by any Semantic Ranges. If the Semantic Ranges SRi and SRj satisfy the condition SRi & SRj = SRN, then both Semantic Ranges SRi and SRj are *disjoined*. Otherwise, we say that both Semantic Ranges SRi and SRj are *joined* . These relationships will be used in the semantic range evaluation, which will be discussed later.

Having the definition of Semantic Range, we can use it to handle ambiguous terms in our system. When we attach the Semantic Range to a group of synonyms, there are several rules that must be followed.

- In the thesaurus model, KeyTerm must be unique. But a synonym term can have more than one KeyTerm. (In other words, a synonym term can appear in different groups of synonyms.)
- Users can attach Semantic Range only to the level 2 thesaurus. If there is no Semantic Range attached (including the level 1 thesaurus), the default semantic range is the Full Semantic Range is SRF.
- The Semantic Ranges of a synonym term are different, if the term has more than one keyterms. In other words, within the same Semantic Range, a synonym term can have only one KeyTerm. The only exception is in the Full Semantic Range SRF. For this case, there are two KeyTerms kt1 and kt2, where the KeyTerm kt1 is in level 1 and the KeyTerm kt2 is in level 2. Then, kt1 is hidden by kt2.
- When a user defines a Semantic Range to attach to a synonym group, this Semantic Range can either be an atom Semantic Range, or several Semantic Ranges connected by the Semantic Range operators. For the latter case, the system will evaluate the operators operating on the Semantic Ranges, and give the result.
- For a synonym group, a user can define a set of semantic ranges attached to this synonym group. Each semantic range will correspond to the pair of a term in this group and the key term, respectively. The default semantic range is full semantic range.

4.3 Semantic Range Evaluation

In the previous section, the concept of Semantic Range was presented. We then described how it is used to represent the ambiguous terms in the thesaurus. In this section, we will present how the Semantic Range can be used to do the de-ambiguity and give stronger support to the functional sub-components.

Basically, there are two ways for handling the Semantic Range in TEXPROS:

1. A user supplies the semantic range information to help the system to do the de-ambiguity.
2. The system can uses the related unambiguous information to decide the semantic range, and then do the de-ambiguity on the ambiguous terms.

The first approach is to get help from users. This is very useful in browsing and retrieval. When a user gives a query to TEXPROS, the system wants the user to give the exact information to narrow the retrieval result. But, in most cases, the user doesn't know the exact terms used in the system. So, the thesaurus in TEXPROS is used to help users specify queries. However, it is very common to have the user give ambiguous terms in the query. For example, a user wants to find all the documents related to "Peter Ng". The user may or may not know that there are two professors who have the identical name, "Peter Ng" without knowing the middle initial of the name of "Peter Ng; one of the professors is in CIS and the other one is in EE. In this case, without semantic range, TEXPROS has no way but returns a bundle of documents to the user, some of the documents related to "Peter A. Ng" and the other documents related to "Peter B. Ng". But with the help of Semantic Range, the user can supply additional information to make the retrieval result

more precise and exact. For example, if the user knows this "Peter Ng" is in "EE" department, then during the retrieval, the user can supply a query such as "Peter Ng(folder range: EE)". Given this query, TEXPROS can do the de-ambiguity by replacing "Peter B. Ng" for the query "Peter Ng". This allows the system to give the user exactly what he/she wants.

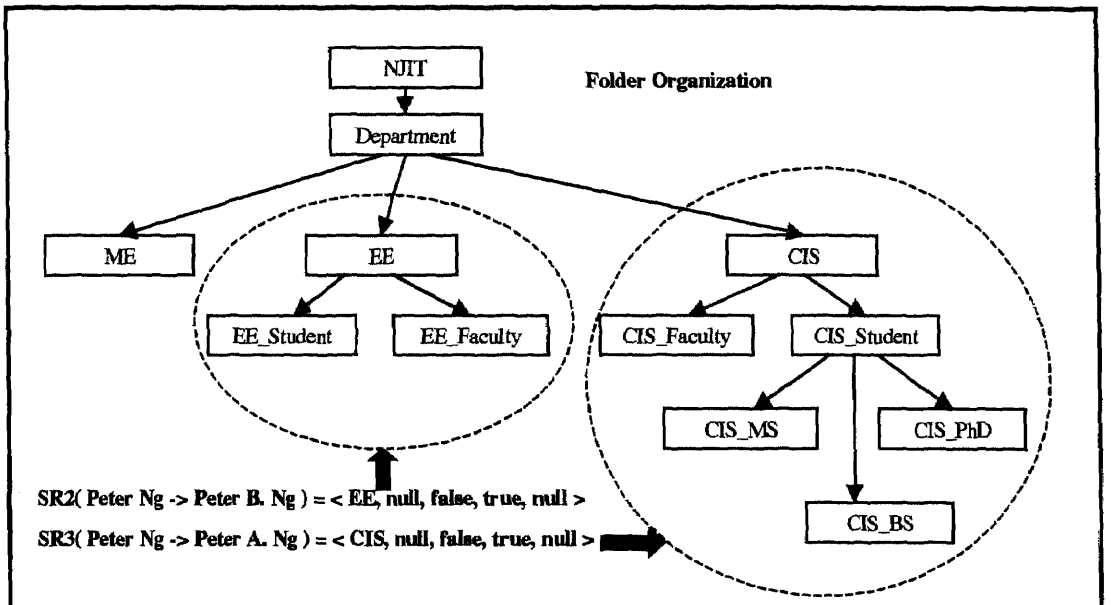
The second approach is to let the TEXPROS automatically handle the de-ambiguity. The basic idea is that TEXPROS do the de-ambiguity by using the related unambiguous information to decide the semantic range. The core of this approach is the Semantic Range Evaluation.

We shall give an example to show how the second approach works.

In the example of Figure 20, a user wants to file a frame instance of a department phone bill into the folder organization of TEXPROS. The folder organization is at the upper location of the figure. The original frame instance is located on the left side of the lower section. Before filing, the system creates the corresponding internal frame instance, which is on its right. The internal frame instance contains the Key Values for those unambiguous terms. For example, the Key Value for the terms "NJIT" and "CIS" are the "New Jersey Institute of Technology" and "Computer and Information Science", respectively. But till now, the internal frame instance has no Key Value for the ambiguous term "Peter Ng". The system will use the Semantic Range Evaluation to do the de-ambiguity.

Definition 9 (*Semantic Range Evaluation*) is an algorithm to evaluate the current working semantic range and candidates of possible semantic ranges for an ambiguous

term. It is used to narrow the candidates of the possible semantic ranges, step by step, until the target semantic range is found for the ambiguous term.



An frame instance of Department Phone Bill

Attribute	Value
university	NJIT
department	CIS
receiver	Peter Ng
date	April 30 1998
amount	32.5
credit	12
balance	20.5
due day	May 20 1998
comment	If you have any question, Pls contact Rose Mary

The corresponding inner frame instance

Attribute	Org Value	Key Value
university	NJIT	New Jersey Institute of Technology
department	CIS	Computer and Information Science
receiver	Peter Ng	?
date	April 30 1998	April 30 1998
amount	32.5	32.5
credit	12	12
balance	20.5	20.5
due day	May 20 1998	May 20 1998
comment	If you have any question, Pls contact Rose Mary	If you have any question, Pls contact Rose Mary

Figure 20 An Example of Semantic Range Evaluation for Document Filing

In document processing, such as TEXPROS, the target semantic range for an ambiguous term is not always a one step hit. In the example of Figure 20, the ambiguous term "Peter Ng" has two candidates of possible semantic ranges, <EE, null, false, true, null> and <CIS, null, false, true, null>. Before filing, we have no way to know which candidate is the target semantic range for this term. But we have the current working semantic range, <null, null, false, true, null>, which represents the term before filing. Using this current working semantic range, an internal frame instance is created. There is no problem in finding all other terms except "Peter Ng". The semantic ranges for this unambiguous term are the default SRF. Using this unambiguous information, the frame instance is filed into the folder "NJIT". Now, the current semantic range becomes <NJIT, Department_Phone_Bill, false, true, null>, which still cannot solve the ambiguous term. Continuously using unambiguous information to file into "Department", the current semantic range becomes <Department, Department_Phone_Bill, false, true, null> and is still no help at all. The system continues the proceeding of filing by using the unambiguous information to match the predicate of folder "CIS", "EE" and "ME". This frame instance can not be filed into folder "EE" and "ME", but folder "CIS". Now the current working semantic range becomes <CIS, Department_Phone_Bill, false, true, null>, which matches one of the candidates of the semantic ranges for "Peter Ng". So, the target semantic range for this term "Peter Ng" here is found to be <CIS, null, false, true, null>. We can use this target semantic range to do the de-ambiguity and get "Peter A. Ng" to update the internal frame instance. Then, the filing of this frame instance can proceed continuously, and this frame instance will go into the folders "CIS_Faculty" and "Peter A. Ng", etc.

Sometimes, the semantic range evaluation is a one step hit. For example, a user wants to define an frame template "Immigration_Application_Form", which is a child of the frame template "Immigration_Form". In this frame template, a user defines an attribute name "from". In this case, the working semantic range is begun with <null, Immigration_Application_Form, true, false, null>, which hits directly at the <null, Immigration_Form, true, false, null>. Because the <null, Immigration_Application_Form, true, false, null> is covered by <null, Immigration_Form, true, false, null>, and it is disjoined with the frame instance <null, Memo, true, false, null>. At this time, we know that this "from" is "nationality", not "sender".

Also, this semantic range evaluation gives strong support to retrieval ranking. For example, consider a user query "CIS & Peter Ng". In this query, "Peter Ng" is an ambiguous term. Without the semantic range, all the documents about "Peter A. Ng" and "Peter B. Ng" have the same ranking for the ambiguous term "Peter Ng". But with the help of semantic range evaluation, the system will find that "CIS" is a folder name, which has a match with the folder range of "Peter A. Ng". So, this information can be used to help the ranking of retrieval results to yield more exact information for users. The algorithm of semantic range evaluation is depicted in Figure 21.

The system uses the Semantic Range Evaluation to do the de-ambiguity for most ambiguous terms. But in some cases, it still needs help from the user. From the algorithm in Figure 21, when a SRw gets stuck, the Semantic Range Evaluation is no longer able to solve the problem by itself, but needs help from the user.

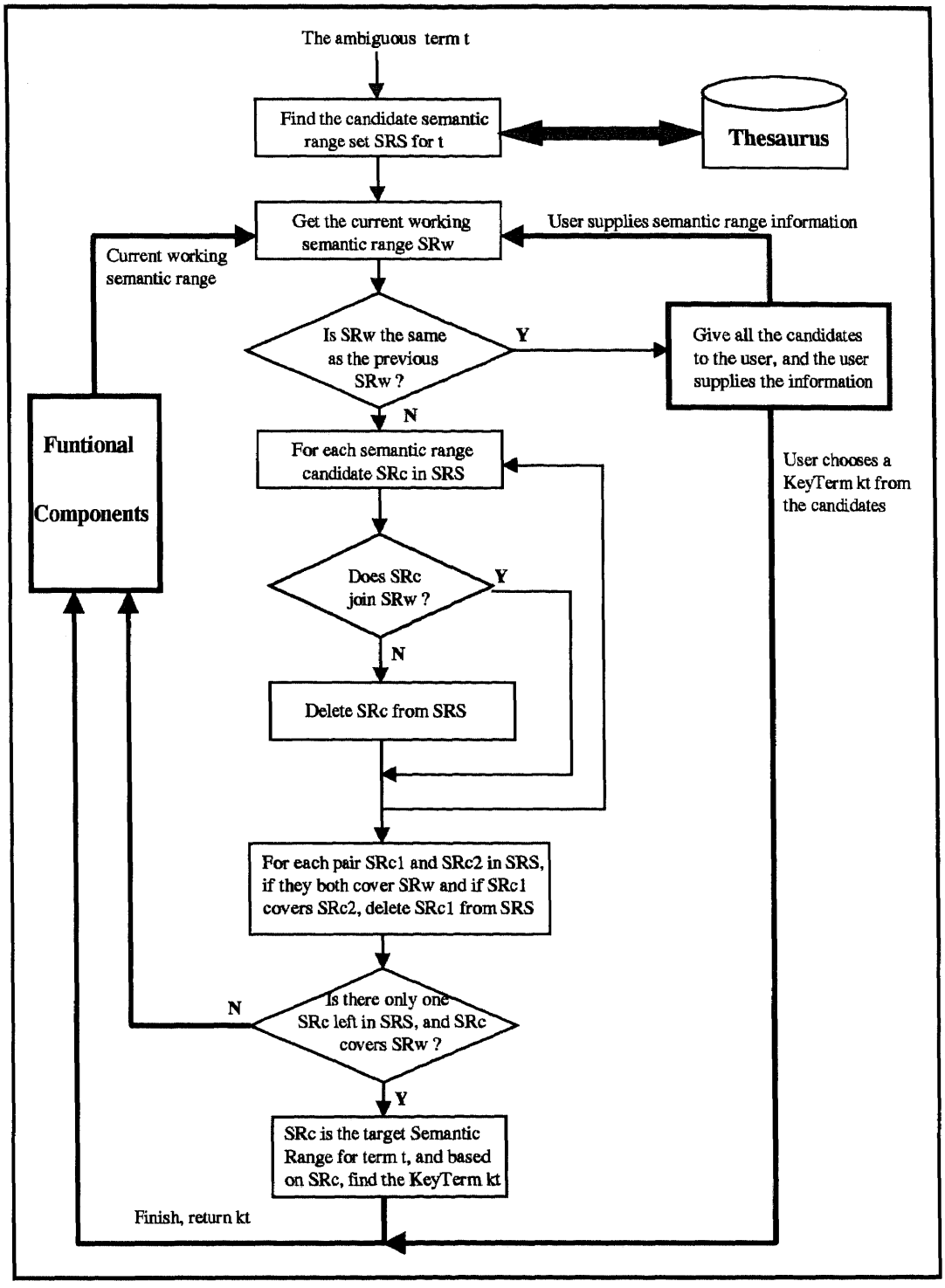


Figure 21 Algorithm for Semantic Range Evaluation

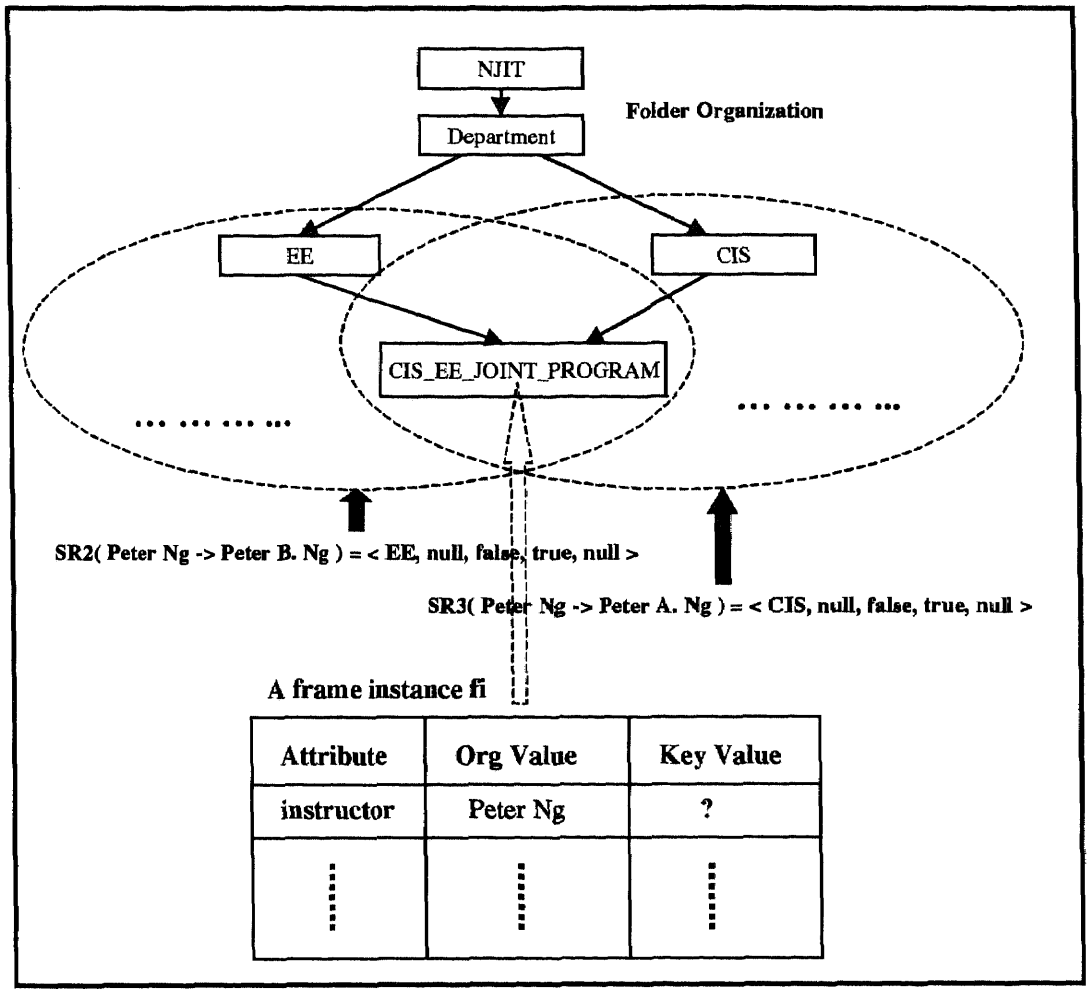


Figure 22 An Example of Ambiguous Case for Semantic Range

Consider an example in Figure 22. We use the semantic range evaluation to find the target range for "Peter Ng". But using the unambiguous information, the frame instance fi is filed into the folder "CIS_EE_JOINT_PROGRAM", which is the child folder of both "CIS" and "EE". And the current working semantic range is < CIS_EE_JOINT_PROGRAM, FT, false, true, null>, which is covered by both SR2 and SR3. Then the current working semantic range gets stuck and cannot proceed further. In

other words, this ambiguous term cannot be solved by the semantic range evaluation automatically, but needs help from the user.

An explanation for this case is given below.

- An author creates a document with a specific goal in his mind. He/she wants readers to understand his written document. If the document is clearly in the unambiguous semantic range (e.g., the memo is transmitted within the CIS department), the author can use ambiguous terms for they can be understood by the reader and the system. If the document is in the ambiguous semantic ranges (e.g., the memo was written by a third party, who does not belong to CIS or EE departments), the author would not use those ambiguous terms to confuse the readers and the system. As in the example in Figure 22, the author will directly use "Peter A. Ng" or "Peter B. Ng" instead of the ambiguous term "Peter Ng". So, the previous case would not happen for most cases.
- For many special cases, if the author uses the ambiguous term, it is always possible to find information from the structured text (e.g., the values for the attributes From, To, Date, and Subject) to imply the unambiguity of the ambiguous term. However, some of the information contained in the free text (i.e., unstructured text, such as the content of the memo), which can imply the unambiguity of the ambiguous term, is difficult for the system to recognize and understand. If this is the case, the system will give all the candidates of the semantic ranges to the user, who will pick the correct KeyTerm for the ambiguous term. Or if the user does not know the exact KeyTerm, he/she can give some additional semantic range information to help the system continue the evaluation until the problem has been solved.

In summary, to solve the ambiguous term, the system can receive help from the user or invoke the semantic range evaluation. These two approaches are tightly coupled and are performed in an interwoven fashion. With the help of semantic range evaluation, the user does not have to give the exact target range, in response to the request of additional range information by the system. On the other hand, the Semantic Range Evaluation can help the user to narrow the range information into more specific range, until the target range is reached. Also, when Semantic Range Evaluation gets stuck in the process of determining the unambiguity of an ambiguous term, the system can request help from the user to re-initialize the Semantic Range Evaluation to proceed the process.

CHAPTER 5

DATADOMAIN

TEXPROS is an intelligent document processing system. Its intelligent capability is derived from various functions for simulating human intelligence and capability to process documents, such as filing, extraction, browsing, etc. The system provides users with these functions. DataDomain is one of the important functional capabilities that TEXPROS gives to its users. DataDomain and thesaurus are complementary to each other. It can solve some of the problems that thesaurus can not solve; and thus it greatly enpowers the TEXPROS' sub-components from different views. Also, it gives strong support to let TEXPROS tailor to specific working (application) domains.

In this chapter, we shall introduce the concept of DataDomain into TEXPROS, and present the detailed structure of DataDomain and DataDomain Agent.

5.1 Motivation

TEXPROS is an intelligent system that is used to handle document processing. Since the primary source for TEXPROS is documents, what the system obtains from these documents are mostly words or phrases. In other words, information obtained from these documents are strings originally, which are insufficient to process the documents, at least, not intelligently. Several examples of the merging problems are given.

Consider a user wants to find documents that are related to the Master students, who have already completed more than 15 credit hours of coursework. But the system fails to find them. Why? Because to the system, although there are many documents that

are related to students, who have completed the number of credit hours, such as "12", "20", "18", etc. But, these numbers are simply words (i.e., strings of characters). To the system, the strings "18" and the string "United States of America" belong to the same data type, called string. Without DataDomain, the system doesn't know that "18" is a number which is bigger than the number "15"; but "United States of America" is a string of characters, called words, which cannot be compared with a number such as "12".

Here is another example. A user wants to find the acceptance letters of the Ph.D students, who were accepted before September 1995. But the system fails to find them. There are some acceptance letters of Ph.D. students in the system and these letters are dated as "January 1995", "December 1995", "September 1994" etc. However, the system doesn't know which values are dates and which are not. More importantly, the system has no way to compare the two dates, says "January 1995" and "September 1995", without having any knowledge of a calendar.

Here is the third example. The system wants to find any documents which contain a phone number "973 596-2990". But in the documents, there are several kind of forms for this number, such as "973-596-2990", "(973) 596-2990", "973/5962990", etc. Given a phone number "973 5962990", the system fails to find any documents that are related to this phone number, which can be represented in different forms. This problem cannot be solved using the thesaurus only, because there are millions of phone numbers in this world. It is impossible to store different forms for each phone number.

Also, when extracting information from a document, the system needs help to recognize the candidate string whether it belongs to a specific type. Without accessing the knowledge of DataDomain, the system has no way to recognize the structure of the string.

These problems arise nearly from all the components of TEXPROS. With this motivation in mind, we shall introduce the concept of DataDomain for solving the problems.

5.2 DataDomain Structure

DataDomain is a user-defined domain of data that is used to support the thesaurus functionality, data type operations and pattern recognition. It may be for the general working domain, such as "Date", "Number". It may be for the specific working domain, such as "SemesterId", "Grade". DataDomain must be self-described, so it is flexible for modification by users to tailor their specific working domain knowledge to TEXPROS.

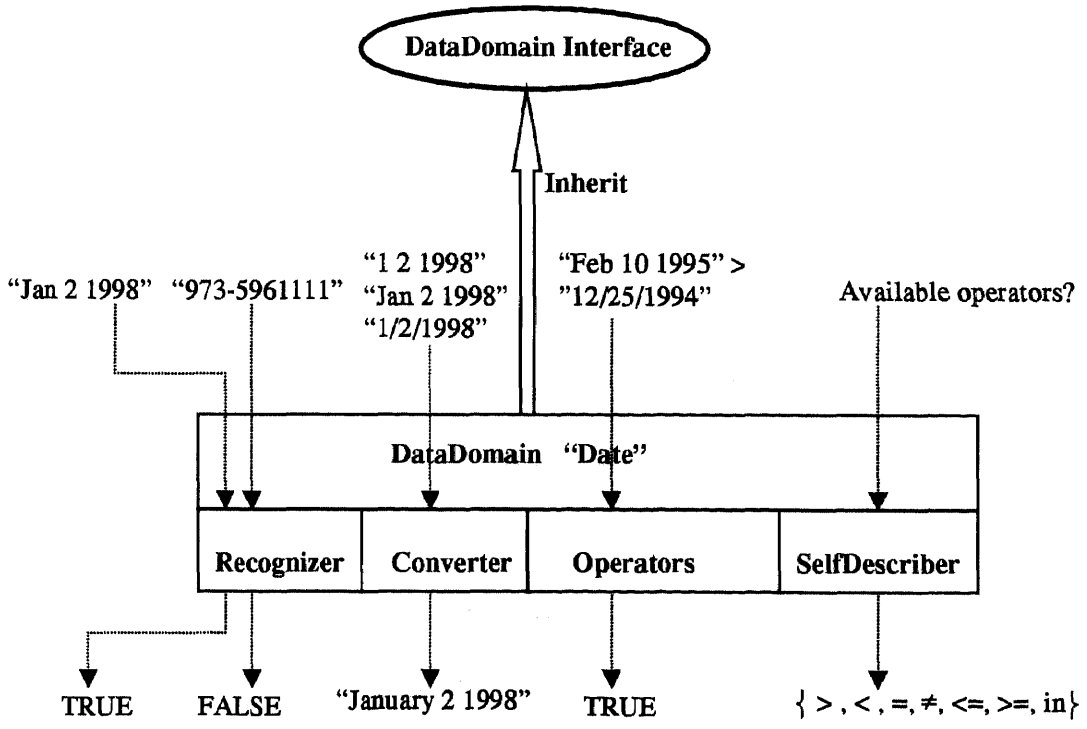


Figure 23 DataDomain "Date"

From object-oriented viewpoint, every DataDomain is a class. A DataDomain has four parts, namely, the recognizer, the converter, the operators and the self-describer. We shall define these parts and we shall use the DataDomain "Date", which is depicted in Figure 23, as an example to illustrate them.

The *recognizer* of a DataDomain is a function that is defined to recognize whether a term, irrespective of its form, belongs to this DataDomain or not. For example, "Date" is a DataDomain. We all know that "January 2, 1998" is a "Date", but "USA" is not a "Date". That means, a human is able to recognize a term whether it belongs to a DataDomain or not. We would like to have TEXPROS to recognize whether a term belongs to a DataDomain or not. Also, for a DataDomain, there may be more than one form for specifying the same value. For example, "January 2, 1998", "1/2/1998", and "Jan 2, 1998" are the same value of a "Date". So, the recognizer should be able to recognize a term irrespective of its possible forms.

The *converter* of a DataDomain is a function that is defined to convert a term of different forms into the KeyForm of this DataDomain. In general, a data, which belongs to the same DataDomain, may have several possible forms. For example, for a date, say, "January 2, 1998" in DataDomain "Date", there are some other forms, such as "Jan. 2, 1998", "1-2-1998", "1/2/1998", etc. Although all of them are in different forms, they share the same meaning (that is, they specify the same day, month and year). Since the domain of the Date contains an unlimited or extremely large data set, it is inefficient to put them as a group of synonyms into the thesaurus. To solve this problem, we introduce KeyForm of DataDomain into TEXPROS.

Definition 10 *(KeyForm)* is one of the possible forms of a DataDomain, which is designated as the internal representation form of the data in this DataDomain. The KeyForm with respect to a DataDomain is similar to the KeyTerm of a synonyms group.

Consider the "Date" case, for an incoming term which could be in the form of "Jan. 2, 1998", "1-2-1998" or "1/2/1998", the converter will transform it into "January 2, 1998", assuming that "January 2, 1998" is their KeyForm. Clearly the recognizer and the converter are important functions, which help the TEXPROS to recognize a data specified in various different forms and to process documents efficiently because of the unique internal representation of the data of the same value but different forms.

Here is a special case that we should concern. In most places and application domains, the date format is in "MM/DD/YY". At that case, we have no problem to convert "1/2/1998" to "January 2, 1998". But for some special application domains, users need to handle not only the format "MM/DD/YY", but also "DD/MM/YY". At that case, "1/2/1998" will be ambiguous, because we have no way to know whether it is "January 2, 1998", or "February 1, 1998". To handle these kinds of cases, here we give two solutions. One solution is that users can define two DataDomains, "MDY_Date" and "DMY_Date", to handle different formats. This solution is good in the case that different format date appears in different type of documents, whose attributes related to date can be defined as either "MDY_Date" or "DMY_Date" in the frame templates. In that way, all the values can be converted without ambiguity. Another solution is when we do the conversion, if we find the ambiguous date, a dialog will be pop up to users, let users decide which is the correct format for this Date value. After users' selection, the converter of DataDomain

will convert the term into its KeyForm, based on the format that users selected. So, in that way, we will only have one KeyForm of the original term after the conversion. These are the solutions to handle the special format cases, like "MM/DD/YY" and "DD/MM/YY". In fact, in the real application domains, users mostly will not need to handle the ambiguous format, because it is hard to recognize and convert even for human beings. So, in this dissertation, we will not consider "DD/MM/YY" as a possible format for "Date". Our discussion is based on the default format "MM/DD/YY".

The *operators* of a DataDomain are the optional functions that operate upon the data of this DataDomain. Some meaningful operations are available to manipulate the data of a DataDomain. We define operators for a DataDomain. For example, for the DataDomain "Date", there are several meaningful operations, such as the operators > (after), < (before), <= (no later than), etc. However, the operators for the DataDomains are different; some of the DataDomain may require no operators. So, the operators of DataDomain are optional.

Besides these three important parts, a DataDomain must be *self-described.* As we discussed before, one of its important functions is to help TEXPROS tailor to the user-specific working domain. However, the concept of DataDomains is not only used to handle the common ones, such as Date, Time, etc., but also designed to handle those special ones, which could be used for a specific working domain or for other special purposes, such as extraction, operations or retrieving. Thus, DataDomain must be flexible, allowing users to easily add, delete or change the DataDomain. We will give the detailed discussion later in Registration Center section. In fact, the *self-describer* allows a

DataDomain to define and describe its functionality itself, which makes it more independent and easier to change and handle.

Definition 11 *(DataDomain)* is a user-defined domain of data, which is used to support the thesaurus functionality, data type operations and pattern recognition in document processing systems. A DataDomain has the following basic data structure.

Data Members:

- operationList: It contains operations that are available to the DataDomain.
- helpMessage: It contains the help messages, which are used for online help

Methods:

- recognizer: It is used to recognize whether an incoming term belongs to the DataDomain or not.
- converter: It is used to convert the form of an incoming term into the KeyForm.
- getOperationList: It is used to indicate the available operations for the DataDomain.
- operation: It is used to deliver the incoming operator with its parameters to the corresponding handler.
- handler: It is optional. Each DataDomain can have several handlers, each of which corresponds to a specific operation.

5.3 DataDomain Agent

In the previous section, we presented the structure of DataDomains in TEXPROS. There may exist numerous different DataDomains in the system. It means that any sub-components must have sufficient knowledge about these DataDomains before they can

apply them. It is not good for the modular system structure, nor is it flexible for users to do modifications on DataDomains. To solve this problem, we introduce DataDomain Agent into TEXPROS.

A DataDomain Agent is an intelligent agent that interacts with the DataDomain base and all the TEXPROS' functional sub-components. It handles all the management and functional issues for DataDomains. Also, in cooperating with the Registration Center, which will be presented in chapter 8, it gives users a great deal of flexibility for to make modifications on DataDomains. In general, the DataDomain Agent is a logical component that hides the details of DataDomains at lower level, and creates a unified DataDomain environment for all the functional sub-components.

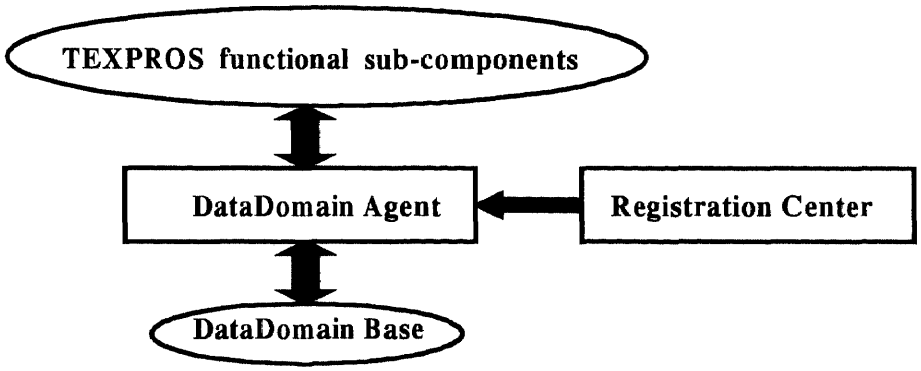


Figure 24 Logical Level of DataDomain Agent in TEXPROS

In TEXPROS, there are many different DataDomains, but there is only one DataDomain Agent. The DataDomain Agent satisfies the principle of information hiding, in such a way that the functional sub-components do not have to know the detailed knowledge about the DataDomains in the DataDomain base, before their applications. The

DataDomain Agent provides the interactions between the functional sub-components and the DataDomain base.

The functionalities of a DataDomain agent are twofold: it handles all the management and functional issues for DataDomains. The detailed structure of DataDomain Agent is depicted in Figure 25.

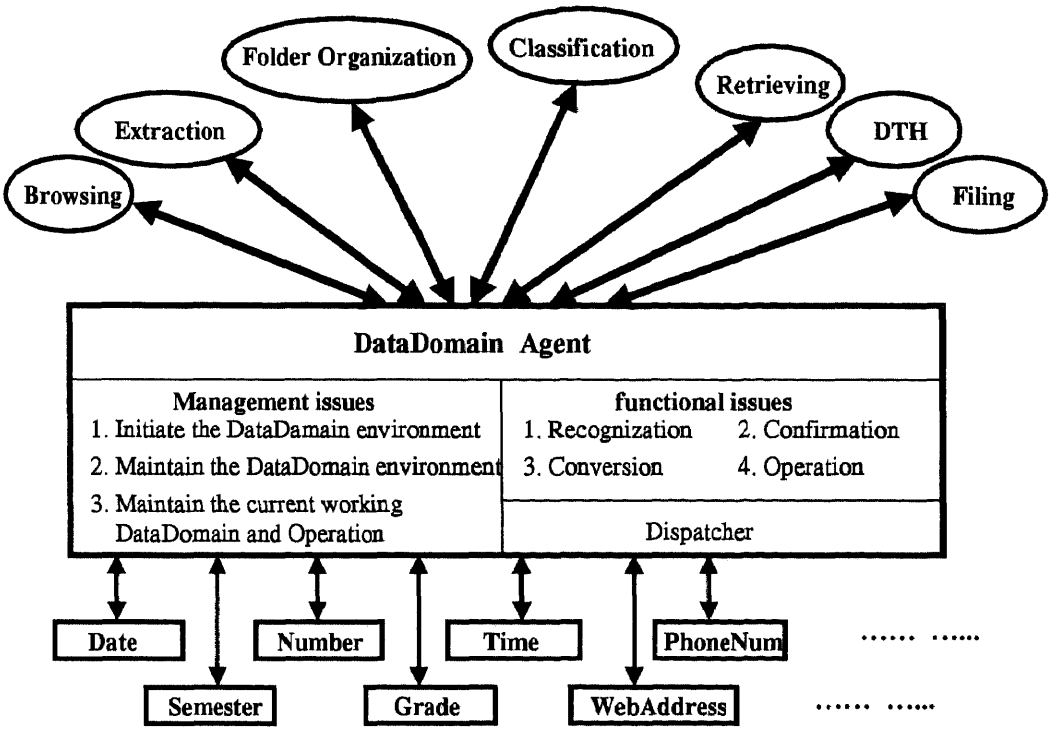


Figure 25 Detailed Structure of DataDomain Agent

The DataDomain Agent has three sub-functions to handle the management issues, namely, initializing the DataDomain environoment, maintaining the DataDomain environment, and maintaining the current working DataDomain and its operation.

- Initializing the DataDomain environment. When the system starts up, the DataDomain Agent begins to gather the necessary information and to initiate the DataDomain environment. As we discussed before, all the DataDomains are self-described to allow DataDomains flexible and easy for users to change, later. Therefore, the DataDomain Agent should dynamically get any relevant information from the binary coded representations of DataDomains. The DataDomain Agent should find all the possible DataDomains and check whether these DataDomains have the correct structures; in particular, each must have the three important parts, recognizer, converter and operators. Then, the DataDomain Agent stores these DataDomains in the available DataDomain list and initiates the standard error message and help message. After these, the DataDomain environment is initiated and wait for their applications to any functional sub-components of the TEXPROS.
- Maintaining the DataDomain environment. While the system is running, the DataDomain Agent is responsible for maintaining the DataDomain environment. A DataDomain is a user-defined domain of data that serves different purposes. TEXPROS allows users to do modifications. Changing DataDomain is fully controlled by the Registration Center (which will be discussed later). To cooperate with the Registration Center, the DataDomain Agent is able to dynamically maintain the environment by adding or deleting a DataDomain from the environment. Also, the DataDomain Agent is able to provide the DataDomain with the structural knowledge, such as, which DataDomain is available, or which operation is available on a DataDomain.

- Maintaining the currently working DataDomain and operation. Finally, the DataDomain Agent will dispatch its entire functional jobs to each specific DataDomain to handle. So, the currently working DataDomain and its operation handlers are dynamically created and deleted. This maintenance is also an important management job for DataDomain Agent.

The DataDomain Agent has four major components to deal with its functional issues. These components are recognition, conversion, confirmation and operations.

The function of the recognition component is to recognize an incoming term and determine which DataDomain it belongs to. It is widely used in browsing. Consider that a user gives a query "(973) 5962990". The process of browsing would not be able to process it, since there are possibly many different forms for the phone number. Without transforming it into a KeyTerm of its data type, we will definitely miss many documents, which contain the same phone number, but in other forms. So, the first step we should do is to transform it into the KeyTerm. However, we find no corresponding term in the thesaurus. At this point, we need help from the DataDomain base, which has a great variety of different DataDomains. The question is, how do we know which one should be used. Here, the DataDomain Agent will dispatch the item "(973) 5962990" to the recognizer of each available DataDomain, receive a bundle of "False" signals, except the "True" signal from the DataDomain "Date" and the system knows that it is a "Date". The DataDomain Agent does not guarantee that there is always one found DataDomain for a given term. There may be more than one possible candidates of DataDomains for a given vague term. Consider the term "1997". It could be a "Number" or a "Date"(year 1997). In

these situations, all found candidates of DataDomains will be returned to the user, who has to decide which DataDomain to use.

The confirmation component of this DataDomain is used to confirm whether the incoming term belongs to a specific DataDomain or not. This component is widely used in the process of document classification, in which the physical structure will decide the brief range and have a list of possible candidate frame templates. When going into the details to decide which one of these candidate frame templates is hit, it needs help from the semantic confirmation to make the decision. For example, for the "Memo" type document, the value of the "Date" usually appears in the upper right hand portion of the document. Suppose that the value "January 3, 1997" appears in the upper right hand portion of a document. Then the classification sub-component will send the request to DataDomain Agent to confirm whether "January 3, 1997" is a Date or not. DataDomain Agent will dispatch this request to the recognizer of "Date" and get the "True" answer. This answer returns to the classification sub-component and helps to decide that it is "Memo" type. If the incoming term is "CIS683", the DataDomain will confirm that it is not Date, which helps the classification sub-component to decide that the document is not the "Memo" type, and could be the "Transcript" type.

This function of the conversion is to convert an incoming term, which belongs to a specific DataDomain, to the KeyForm of this DataDomain. It is an extension of the thesaurus. As we described earlier, thesaurus is very importantly applicable in the whole system. But there are some problems that cannot be solved even with the aids of the thesaurus. For example, there are a variety of different forms for representing the "Date" and there are unlimited values for the "Date". It is almost impossible and certainly

impractical to store every value of the "Date" in thesaurus. However, this class of problems can be solved easily using the conversion of DataDomain Agent. When dealing with the value "1-2-1997" and the "Date", the DataDomain Agent dispatches the value to the converter of "Date", which transforms the value "1-2-1997" into the KeyForm "January 2, 1997" and returns to the source sub-component ready for use. Consider another example, the value "1997F" and "Semester". The DataDomain Agent dispatches the value "1997F" to the converter of "Semester", which transforms the value into the KeyTerm "1997 Fall". If the source sub-component is the extraction sub-component, then this transformed value will be assigned as the KeyValue. If the source sub-component is the browsing or retrieving sub-component, then this transformed value will be used to match against the KeyValues to find the result of query. If the source is a folder predicate, then this transformed value will be used to modify the predicate representation.

The operation function is used to manipulate the data type operations of a specific DataDomain. It is widely used in the processes of browsing, retrieving and folder organization predicate evaluation. For example, consider a query, such as "<send date> in September 1995". In this example, the user wants to find all the documents, which contains the send date in September 1995. If our system wants to process this query, it must have the knowledge how to handle the operator "in" for two date parameters. To solve this problem, the operation request and the parameters will be sent to the DataDomain Agent, which will, in turn, dispatch them to the corresponding DataDomain and their Operation handler. The corresponding DataDomain and Operation handler will process the operation and return the boolean result. Then, the DataDomain Agent will return this result to the source sub-component to help the processing continue.

Basically, both the thesaurus and the DataDomain base are used to handle certain specific data. The DataDomain base is an extension of thesaurus, for it is used to solve the problems that could not be handled using the thesaurus. It also supports the pattern recognition and data type operations. Users can use "DataDomain" to define any kind of domain of data. Some common DataDomains are number, date, etc. Some DataDomains are special for users' working domains; and they are such as grade, semester-id, course-id, which are defined by the users for various purposes, such as the extraction, the thesaurus support, the retrieving process. Users can use the "DataDomain" to define the preferred domain of data. This enpowers TEXPROS to be tailored to different specific working domains. Also, with the help of the DataDomain Agent, all the details of DataDomains are hidden from the sub-components. DataDomain Agent handles all the management issues and functional issues of DataDomain. Therefore, the sub-components don't have to know the detailed information about the DataDomains; they simply interact with the DataDomain Agent to get the strong support from DataDomain base.

CHAPTER 6

KEYTERM TRANSFORMATION COMPONENT

6.1 KeyTerm Transformation

In the previous chapters, we have presented a two-level thesaurus model, which allows the system to find the key term, KeyTerm, for any given user's term using the exact term matching or sense matching. We also presented the notation of KeyTerm and internal frame instance to speed up the document retrieval for TEXPROS. Based on the presented approach, when a document comes into our system, the classification and extraction components create its corresponding frame instance and then create the internal frame instance by transforming each original value into the obtained key value, and store it into the frame instance base. Later, for retrieving documents, the system will also transform the original query into a query, in which all the terms are KeyTerms. These KeyTerms are used to match the KeyValues of internal frame instances during the search for the related documents. Therefore, it is critical to determine a KeyTerm transformation procedure for transforming an original term to its corresponding KeyTerm. This KeyTerm transformation is a major procedure for implementing the sense matching for TEXPROS, which would be used by many other components of the TEXPROS.

In chapter 3, the "transformToKeyTerm" algorithm is presented. But that algorithm is only used for the thesaurus part. It is inadequate for the general KeyTerm transformation problem of TEXPROS. For example, finding all the documents with the date " 1/20/1997 ", the system will begin to transform this query into the KeyTerm form. But transforming " 1/20/1997 " into its corresponding KeyTerm requires more than a

thesaurus. In chapter 5, the concept of DataDomain is presented to solve this kind of problem. Also, for ambiguous terms, such as the term "Peter Ng", both the thesaurus and the semantic range are used to help solve the problem of ambiguous terms. In other words, we need a thesaurus, the DataDomain and Semantic Range to handle the general KeyTerm transformation problem of TEXPROS. For each of them, we have described our approaches in the previous chapters. In this chapter, a KeyTerm Transformation component is presented that will integrate the thesaurus, DataDomain and Semantic Range as an unit for finding the solution of the general KeyTerm transformation problem of TEXPROS. We shall discuss the detailed design of the KeyTerm Transformation component, the KeyTerm transformation process control and how the integration of the three major parts for solving the KeyTerm transformation problem. In addition, we shall define the KeyTerm transformation request and reply as unified input and output data structure for KeyTerm Transformation component.

Before we go into the details of the KeyTerm Transformation component, we shall extend the definition of KeyTerm. The TEXPROS' thesaurus consists of the definitions of KeyTerm. Given an original term, the KeyTerm Transformation component obtains its corresponding KeyTerm. In the following, we extend the KeyTerm definition from the viewpoint of KeyTerm transformation.

In KeyTerm Transformation component of TEXPROS, for an original term, OT, and its corresponding KeyTerm, KT,

- If OT appears in the thesaurus of TEXPROS, KT follows the KeyTerm definition of thesaurus of TEXPROS. For example, the KeyTerm of the term "CIS" is "Computer and Information Science".

- If OT belongs to a specific DataDomain, KT is the KeyForm for that DataDomain of OT. For example, the KeyTerm of the term "1/2/1998" is "January 2, 1998".
- If OT is an ambiguous term, KT is the target term, which is obtained after the semantic range evaluation. For example, the target term of the ambiguous term "Peter Ng" is "Peter A. Ng".
- If none of the previous conditions match, KT is the same as the OT. For example, the KeyTerm of the term "software" is itself.

6.2 KeyTerm Transformation Request and Reply

The major job of KeyTerm Transformation component is to get the request from other functional components, to transform an incoming original term to its corresponding KeyTerm, and to reply it to the source component. But, based on different types of requests, there requires different information for transforming a given term into its corresponding KeyTerm. Also, the information responded to the functional components is varied based upon the result of the transformation. To solve this problem, we define KT_Request and KT_Reply as the standard input and output data structure for KeyTerm Transformation component. With KT_Request and KT_Reply, we give the unified KeyTerm transformation request and reply interfaces, which can ease the communication between the KeyTerm Transformation component and functional components of TEXPROS.

6.2.1 KeyTerm Transformation Request

For different types of KeyTerm transformation requests, we define KT_Request as the standard input data structure of KeyTerm Transformation component.

```
KT_Request {
  request_type;
  original_term;
  DD_Name; (DataDomain Name)
  SRT_ID; (Semantic Range Task ID)
  SRw; (Current Working Semantic Range)
}
```

The KT_Request is used to handle three types of possible KeyTerm transformation requests. For the first type, DataDomainKnown type, the functional component already knows the DataDomain that the original term belongs to. In this case, the request_type is assigned with a value 1. The original_term and DD_Name have the original term and its corresponding DataDomain name as their values, respectively. And the other data members will be NULL. For the second type, OTOnlyKnown type, the functional component only knows the original term. In this case, the request_type is assigned with a value 2. The orginal_term is used to store the original term. And the other data members will be NULL. For the third type, AmbiguousTerm type, the requested original term has already been sent to KeyTerm Transformation component before. But KeyTerm Transformation component finds it is an ambiguous term; sends back a Semantic Range Task ID (SRT_ID) and needs more working semantic range information to help SRE(Semantic Range Evaluation) to solve the problem. In this case, upon having the

updated working semantic range information, the functional component sends the request again as the third type KeyTerm transformation request. So, for the third type of KeyTerm transformation request, the "request_type" is assigned with a value 3. The orginal_term, SRT_ID and SRw are used to store the original term, Semantic Range Task ID and the current working semantic range. And the other data members are NULL.

In KT_Request, we also give the following three constructors:

- KT_Request(original_term, DD_Name); //DataDomainKnown
- KT_Request(original_term); //OTOnlyKnown
- KT_Request(original_term, SRT_ID, SRw); //AmbiguousTerm

Using these three constructors, the functional components of TEXPROS can easily build the corresponding type of KeyTerm transformation requests and can send the necessary information into the KeyTerm Transformation component. In the later sections, we shall describe the details of the structures and process control of these types of KeyTerm transformation requests.

6.2.2 KeyTerm Transformation Reply

For different types of KeyTerm transformation reply, we define KT_Reply as the standard output data structure of KeyTerm Transformation component.

```
KT_Reply {
  key_term;
  status; (status is successful or ambiguous or failed)
  SRT_ID; (Semantic Range Task ID)
}
```

For most cases, our KeyTerm Transformation component will find the corresponding KeyTerm of the original term in the KT_Request. At that time, key_term is used to store the found KeyTerm; status is marked as successful and SRT_ID is NULL. If the KeyTerm Transformation component finds the original term is an ambiguous term, the KT_Reply will return the ambiguous status and a Sematic Range Task ID, from the data member SRT_ID. The other data members are NULL. Then the functional component is fully aware that the requested term is an ambiguous term and the KeyTerm Transformation component requires working semantic range information to handle the ambiguity. When the functional component has the current working semantic range information, based on the Semantic Range Task ID, the RT_Request of type 3 is constructed and the KeyTerm Transformation component is requested to find the KeyTerm for the ambiguous term. If the status is failed, the functional component constructs a request of type 1, and finds the original term does not belong to the DataDomain after the DataDomain recognition. In this case, the KT_Reply returns the status "failed". Then the functional component knows that some errors happened for that original term. A warning is given to the user, who should check the original document.

6.3 KeyTerm Transformation Component

In the previous section, we have defined KT_Request and KT_Reply as the standard input and output data structure for KeyTerm Transformation component. In this section, we shall describe the detailed structure and process control of KeyTerm Transformation component.

A KeyTerm Transformation component, in cooperating with the thesaurus and the DataDomain Agent, consists of ten sub-modules and a Semantic Range Task Buffer. The detailed structure is shown in Figure 26.

The first module is a Request Dispatcher. In the previous section, we defined three types of KeyTerm transformation requests. Their processes and control are different. When receiving the KT_Requests from the functional components, the Request Dispatcher module dispatches the requests to the corresponding sub-modules based upon their type. The KT_Request of type 1 (DataDomainKnown type) is dispatched to the DD_Term Conversion Controller module. The KT_Request to type 2 (OTOnlyKnown type) is dispatched to the Transformation Controller module and KT_Request of type 3 (AmbiguousTerm type) is dispatched to the De-ambiguity Controller module.

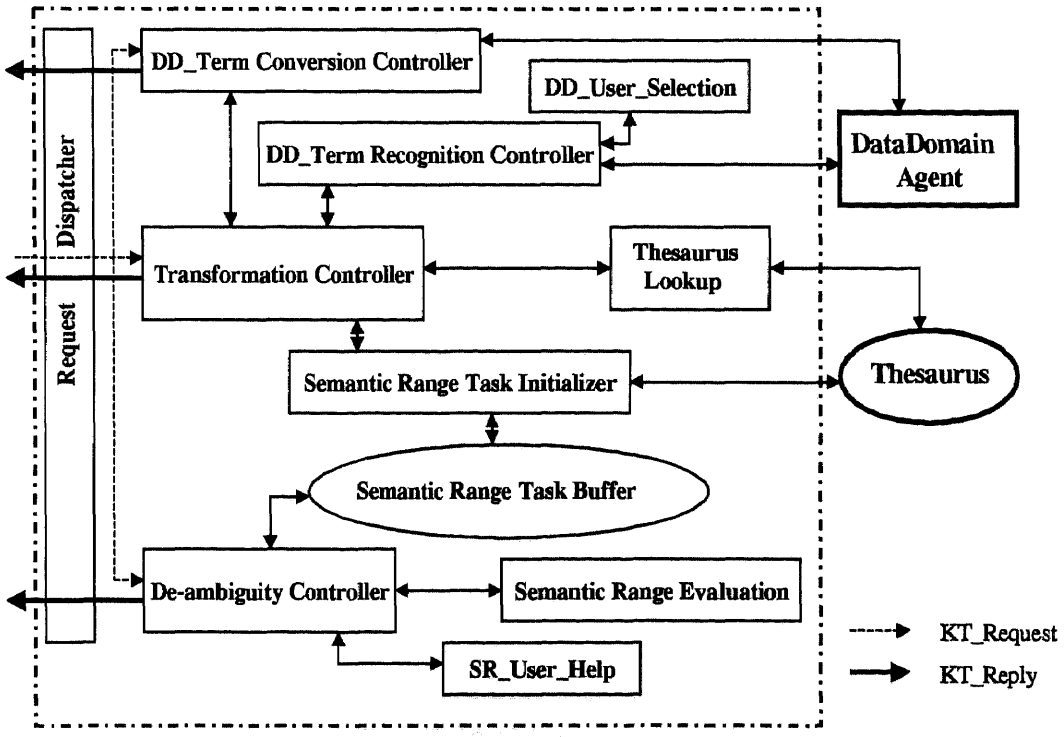


Figure 26 KeyTerm Transformation Controller Structure

Semantic Range Task Buffer is the buffer place for storing the semantic range task information. In chapter 4, we have presented the approach which uses the Semantic Rang and Semantic Range Evaluation to solve the sense ambiguity problem. Based on this approach, when the KeyTerm Transformation component cannot find the corresponding KeyTerm at once for an ambiguous term, the thesaurus is used for finding all the candidates for that ambiguous term. At that time, a semantic range task is constructed and is assigned with a unique ID. This task ID is returned to the functional component. Later, when the functional component gets the updated working semantic range information, the KT_Request is sent again, based on the task ID. Using this mechanism, the KeyTerm Transformation component can continuously get the current working semantic range information, and can use the Semantic Range Evaluation to narrow down the set of candidates until the corresponding KeyTerm is found. During this procedure, the Semantic Range Task Buffer is a temporary buffer, which is used by KeyTerm Transformation component to store the information of semantic range tasks. The structure of the Semantic Range Task Buffer is shown in Figure 27.

SRT_ID	Original_Term	Previous_SRw	Candidate_Set
.....	.....	.....	.....
.....	.....	.....	.....
20101	Peter Ng	SRF	Peter B. Ng , <EE, null, false, true, null> Peter A. Ng , <CIS, null, false, true, null>
20102	From	SRF	Nationality , <null, Immigration_Form, true, false, null> Sender , <null, Memo, true, false, null>
.....	.....	.....	.....
.....	.....	.....	.....

Figure 27 Semantic Range Task Buffer Structure

The Semantic Range Task Buffer is a table-like structure, which contains four columns, SRT_ID, Original_Term, Previous_SRw and Candidate_Set. Each semantic range task will be stored as a record in this buffer. SRT_ID is used to store the semantic range task ID, which is the unique ID for each semantic range task. Based on this unique ID, we can find all of the information related to a specific semantic range task. Original_Term and Previous_SRw are used to store the original ambiguous term, and the last current working semantic range that we obtained for this task, respectively. The latter information will be used to recognize the bottleneck during the Semantic Range Evaluation. Candidate_Set is used to store the remaining candidate set for the ambiguous term. During the process of sense de-ambiguity, the information in the last two columns will be continuously updated by the sub-module "De-ambiguity Controller".

For KeyTerm Transformation component, there are three types of KeyTerm transformation requests. The "Request Dispatcher" module will dispatch the requests of different types to their corresponding modules for their processing. In the following sections, we will present how to use KeyTerm Transformation component to handle the KeyTerm transformation requests of different types. At the same time, we will introduce the detailed functions of each sub-module and how to control the process flow between them.

6.3.1 First Type of KT_Request Handling

For the first type (DataDomainKnown type) KeyTerm Transformation Request, the functional component already knows the DataDomain that the original term belongs to. In this case, the functional component can use the constructor KT_Request(original_term,

DD_Name) to create the KT_Request of type 1 which is sent to the KeyTerm Transformation component. The Request Dispatcher module dispatches the request to the DD_Term Conversion Controller module for further processing. Figure 28 shows how the KeyTerm Transforamtion component handles the KT_Request of type 1.

Figure 28 First Type of KT_Request Handling

After receiveing the KT_Request, the DD_Term Conversion Controller module contacts the DataDomain Agent and send the original term and its corresponding DataDomain name to the agent. Based on the DataDomain name, the DataDomain Agent will try to convert the original term into its corresponding KeyForm, which is returned to the DD_Term Conversion Controller module. As the DD_Term Conversion Controller module receives the KeyTerm from the DataDomain Agent, the module creates the KT_Reply{ KeyTerm, successful, NULL }, which is returned to the functional component.

Here is a special case. Based on the DataDomain name, the DataDomain Agent cannot convert the original term into its KeyForm. In other words, the original term does not belong to that specific DataDomain. If this happens, the DD_Term Conversion Controller module will get the error message from the DataDomain Agent to create the KT_Reply { NULL, failed, NULL }, which is returned to the functional component.

6.3.2 Second Type of KT_Request Handling

For the second type (OTOnlyKnown type) KeyTerm Transformation Request, the functional component knows only the original term. In this case, the functional component can use the constructor KT_Request(original_term) to create the KT_Request of type 2, which is sent to KeyTerm Transformation component. The Request Dispatcher module dispatches this request to the Transformation Controller module for further processing. Figure 29 shows how the KeyTerm Transformation component handles the KT_Request of type 2.

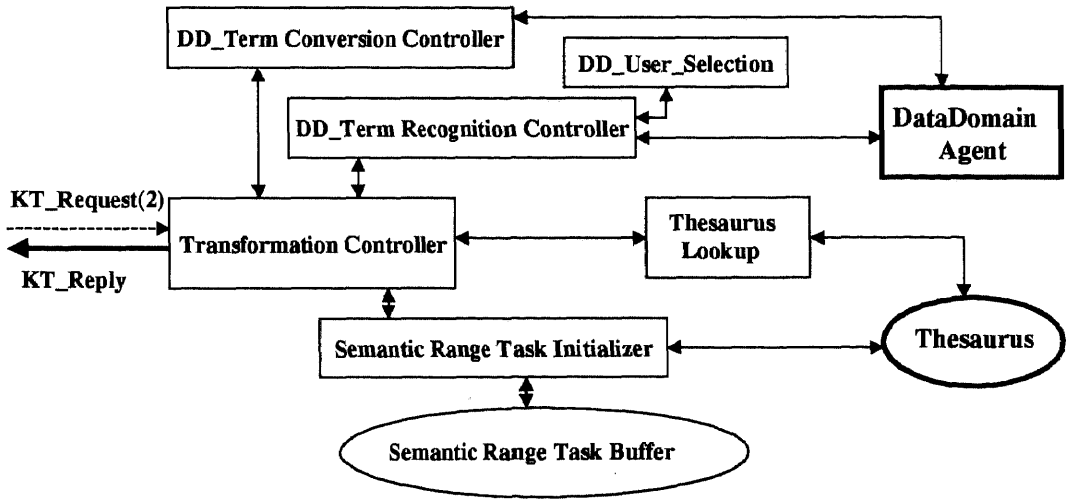


Figure 29 Second Type of KT_Request Handling

In TEXPROS, most KeyTerm transformation requests belong to the second type. The handling of the requests of type 2 is much more complex than the handling of the first type of requests. Six modules are employed to handle this type of KT_Requests. The Transformation Controller module is the central module here to control the transformation process. It is also responsible to receive the KT_Request and then to create and return

KT_Reply to the functional component. The Thesaurus Lookup module has the major responsibility for checking the thesaurus to find the KeyTerm for the original term. Its core algorithm is given in Chapter 3. The Semantic Range Task Initializer module has the major responsibility for creating a semantic range task in the buffer when an ambiguous term is found. The DD_Term Recognition Controller module is responsible for controlling the DataDomain recognition for an original term. When an original term has more than one DataDomain, the DD_User_Selection module will contact the user for making the selection. The DD_Term Conversion Controller module was introduced in the previous section.

The Request Dispatcher module will always dispatch the KT_Request of type 2 to the Transformation Controller module. Upon receiving a request, the Transformation Controller module will extract the original term, which is sent to the Thesaurus Lookup module first. Based on this original term, the Thesaurus Lookup module will check the thesaurus for finding its corresponding KeyTerm. There may have three possibilities here.

Firstly, the Thesaurus Lookup module finds the corresponding KeyTerm in the thesaurus, which is returned to Transformation Controller module. In this case, the Transformation Controller module will create a KT_Reply { KeyTerm, successful, NULL }, which is returned to the functional component. This ends the KeyTerm transformation procedure.

Secondly, the Thesaurus Lookup module cannot find that original term in the thesaurus and send this message back to the Transformation Controller module. From the viewpoint of thesaurus, if an original term does not appear in the thesaurus, it means it does not belong to any synonym group and its KeyTerm is defined by default as the

original term itself. But in KeyTerm Transformation component, our processing is a little different. In the previous section, we have extended the definition of KeyTerm. The KeyTerm Transformation component employs not only the thesaurus, but also the DataDomain and the Semantic Range. If an original term does not appear in the thesaurus, it may belong to a specific DataDomain that is not supported by the thesaurus. When this happens, the Transformation Controller module will not return the original term as default KeyTerm immediately. Instead, the Transformation Controller module will send the original term to the DD_Term Recognition Controller module. The DD_Term Recognition Controller module will contact DataDomain Agent to check whether this original term belongs to a specific DataDomain or not. There are three possible cases here.

Case 2.1: The DataDomain Agent finds that this original term belongs to a specific DataDomain. Then the DD_Term Recognition Controller module will get this DataDomain name and send it back to the Transformation Controller module. Then the Transformation Controller module will send the original term and that DataDomain name to the DD_Term Conversion Controller module, which will convert the original term to its corresponding KeyForm, and then return the KeyTerm to the Transformation Controller module. Finally, the Transformation Controller module will create the KT_Reply { KeyTerm, successful, NULL }, which is returned to the functional component. This ends the transformation procedure.

Case 2.2: The DataDomain Agent finds that this original term belongs to more than one DataDomain. In this case, the DD_Term Recognition Controller module will send the original term and all the candidate DataDomain names to the DD_User_Selection module for user help. The user has to select from all the candidates, which DataDomain this

original term belongs to. Then DD_Term Recognition Controller module will send this DataDomain name back to the Transformation Controller. The rest of the process is the same as Case 2.1.

Case 2.3: The DataDomain Agent finds that this original term does not belong to any DataDomain. In this case, the DD_Term Recognition Controller module will send this message back to the Transformation Controller module. Now the Transformation Controller module knows that this original term does not belong to any DataDomain or the thesaurus. Based on the KeyTerm definition, the Transformation Controller moduel uses the original term itself as the default KeyTerm and creates KT_Reply { KeyTerm, successful, NULL }, which is returned to the functional component. The transformation procedure is ended.

Thirdly, when the Thesaurus Lookup module finds the original term to be an ambiguous term and sends this message back to the Transformation Controller module. Then the process control will go to the Semantic Range Task Initializer module. Based on the original term, this module will extract its related information from the thesaurus; will assign a unique semantic range task ID and will create a new task record in the semantic range task buffer. Based on the semantic range task ID, the Transformation Controller module creates KT_Reply { NULL, ambiguous, SRT_ID }, which is returned to the functional component. This KT_Reply will inform the functional component that the incoming original term is an ambiguous term and the KeyTerm Transformation component needs additional working semantic range information to do the de-ambiguity. Processing of the ambiguous terms will be discussed in the next section.

6.3.3 Third Type of KT_Request Handling

For the third type (AmbiguousTerm type) KeyTerm Transformation Request, the functional component knows not only the original term, but also the semantic range task ID attached to it. It means this original term is an ambiguous term and the KeyTerm Transformation component needs functional component to provide current working semantic range information for doing the de-ambiguity. In this case, the functional component can use the constructor KT_Request(original_term, SRT_IC, SRw) to create the KT_Request of type 3, which is sent to the KeyTerm Transformation component. The Request Dispatcher module dispatches this request to the De-ambiguity Controller module for further processing. Figure 30 shows how the KeyTerm Transformation component handles the KT_Request of type 3.

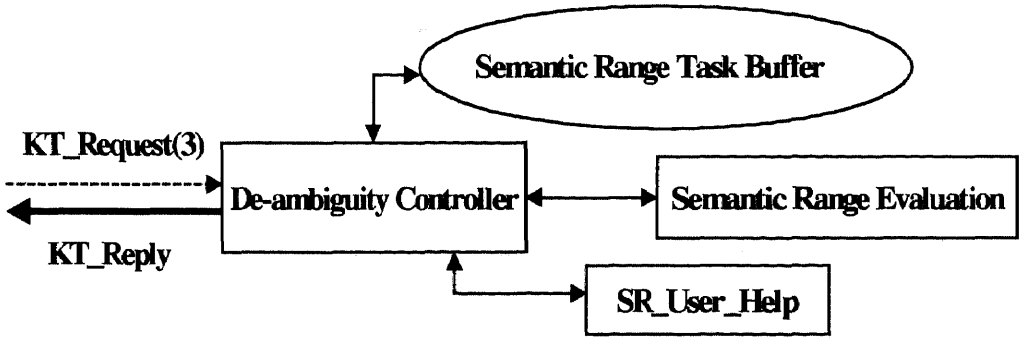


Figure 30 Third Type of KT_Request Handling

When the De-ambiguity Controller module receives the KT_Request of type 3, based on semantic range task ID, this module extracts the corresponding task record from the Semantic Range Task Buffer to get all the related information. Firstly, the De-ambiguity Controller module will check whether the current working semantic range of the KT_Request is the same as the previous working semantic range. If the current

working semantic range is the same as the previous one, it means that the functional component cannot give further information to help us solve the ambiguity. In other words, we find the bottleneck that our semantic range evaluation cannot solve. In this case, the SR_User_Help module will be used to give all the candidates information to the user for help. If the user gives more semantic range information, the De-ambiguity Controller module will use the current working semantic range to continue the processing. If the user directly picks a candidate as the KeyTerm of the original term, the De-ambiguity Controller module will use this KeyTerm, to create the KT_Reply { KeyTerm, successful, NULL }, which is returned to the functional component. So, the ambiguity is solved and the corresponding semantic range task record will be deleted from the buffer.

If the current working semantic range is different from the previous one, the De-ambiguity Controller module will send the current working semantic range and all the candidates information to the Semantic Range Evaluation module. The details of the Semantic Range Evaluation module were described in Chapter 4. The Semantic Range Evaluation module will use the current working semantic range to evaluate the candidates and try to narrow down continuously the candidate set. If the candidate set is reduced by the evaluation module and the target KeyTerm could not be found, the new candidate set will be sent back to the De-ambiguity Controller module. The De-ambiguity Controller module will use the new current working semantic range and candidate set information to update the corresponding semantic range task record in the buffer. Also, it will create KT_Reply { NULL, ambiguous, SRT_ID }, which is returned to the functional component. The functional component will be asked, in turn, for further working semantic range information for doing the de-ambiguity. If the evaluation module narrows down the

candidate set and at last finds the target KeyTerm. The De-ambiguity Controller module will use this KeyTerm to create the KT_Reply { KeyTerm, successful, NULL }, which is returned to the functional component. So, the ambiguity is solved and the corresponding semantic range task record will be deleted from the buffer. This ends the transformation process.

KeyTerm transformation is a critical procedure for implementing the sense matching for TEXPROS. It gives strong supports to all the functional components and is frequently used during the document processing of TEXPROS. To handle this problem, we need the combination of the thesaurus, the Semantic Range and the DataDomain. In this chapter, we presented the KeyTerm Transformation component, which integrates the thesaurus, the Semantic Range and the DataDomain for solving the general KeyTerm transformation problem of TEXPROS. We have discussed the detailed functions of ten modules of KeyTerm Transformation component, the structure of the Semantic Range Task Buffer, and how to control the transformation process between them. Also, we defined the KT_Request and KT_Reply as the standard input and output data structure.

With the KeyTerm Transformation component, we hide the details of KeyTerm transformation procedure from all the functional components of TEXPROS. The functional components can use the KT_Request and the KT_Reply as the standard interfaces, allowed to have an easy communication with the KeyTerm Transformation component. The KeyTerm Transformation component will handle the detailed transformation procedure to find the KeyTerm, which corresponds to the original term requested by a functional component. In this way, the functional components do not have

to know the detailed information about the transformation control, the thesaurus, etc.; they simply interact with the KeyTerm Transformation component to get the KeyTerm transformation support. Therefore, the KeyTerm Transformation component provides the solution for the common problem in all the functional components. This greatly eases their burden and allows them to focus on their specific areas of the document processing of TEXPROS.

CHAPTER 7

APPROXIMATE TERM MATCHING

Currently, upon receiving a user's term, the exact term matching is used in TEXPROS to find its matched terms which are used in the system. Although the thesaurus, the DataDomain and the Semantic Range can help us to conduct term matching and sense matching, we still need the first term to be exactly matched. However, in some cases, without other accessory components, the system would fail to provide the users with cooperative answers.

For example, a user wants to find some documents related to "software tester". Unfortunately, he mis-types it as "software taster", which doesn't match any documents or terms in our system. The system gives the user a NULL answer, without any further assistance. Another example, a user wants to find some documents related to "Jianshun Hu". But this name is difficult to remember and spell. However, the user can give an approximate spelling "Jinsun Hu". For this case, our system surely cannot find terms and documents related to the user's query and therefore gives a NULL answer.

Till now, when this sort of cases happens, the system cannot give users further assistance. It therefore depends on the users to find the correct term. This system fails to provide users with cooperative answers, especially in document browsing and retrieval. For some special cases, like the "Jinsun Hu" example, without any help from the system, the user can hardly find the correct term that he wants.

With this motivation, we designed an "approximate term matching" module to help users find the terms used in the system. In this module, the users can give an original term

to request the approximate term matching. TEXPROS will go through the terms used by the system to find the approximately matched terms, rank them and give them back to users. In this way, the system can assist users to find the system terms that they want.

Currently, there are some research and commercial software which support the similar functionality. A famous one is the "spelling check" in Microsoft Word. All of these, including our "approximate term matching", are from the approximate matching viewpoint to give users assistance. But these systems, which are functionally different, have their won application domains. For example, the "spelling check" targets at helping users to correct spelling errors during the document creation. Because of this goal, the "spelling check" will only be automatically triggered when the users spell the words wrongly. Also, it will only check the word dictionary and some accessory terms inserted by the users. But the "approximate term matching" targets at helping users to find the term used in the system, not for the spelling correction. Hence, the original term can be a single word or a complex term (such as, "software taster"). The original term may be spelled wrongly or correctly. The users can apply the "approximate term matching" in all of these cases. Also, unlike the "spelling check" nearly only searches in dictionary, the "approximate term matching" will search all the system terms, which include folder names, frame template names, attribute names, values, and thesaurus terms. Finally, the same core algorithms may be used in different software, but the approximate term matching is a time consuming job to accomplish different targets in different software environment. We need to design our own processes to speed up and ease the matching procedure.

In TEXPROS, we employ two existing popular techniques, pattern matching and edit distance comparison, to do the approximate term matching, and help users to find the

approximate matched terms used in the system. In this chapter, we will present the details of "approximate term matching" and some accessory processes we used to speed up and ease the matching procedure.

7.1 Pattern Matching

Pattern matching is a popular technology used in the approximate term matching area. The core idea is that the system supplies a group of pattern operators. When a user knows to spell part of a term or wants to find a group of similar spelling terms, he can use the pattern operators to help create the original term pattern. The system will find all the matched terms, based on the original term pattern.

In TEXPROS, we provide two pattern operators to the users. One is "/*", which is used to substitute zero or more characters. The other is "/?", which is used to substitute one character. For example, if a user gives the original term pattern such as "software /*er", the system will find the matched terms like "software tester", "software engineer" and "software manager", etc. If a user gives the original term pattern such as "Ji/?ns/*n Hu", the system will find the matched terms like "Jianshun Hu", etc. The core algorithm for the pattern matching is already implemented by Java classes and SQL "like" operator of Oracle, which can be easily used by the prototype of TEXPROS.

In the "approximate term matching" of TEXPROS, when the "pattern matching" is used, the system provides its users three optional selections: letter case sensitive, blank space sensitive and category selection. The "letter case sensitive" option is to tell the system whether the user takes the upper case and lower case of letters into consideration. The "blank space sensitive" option is to tell the system whether the user cares the blank

space in the terms or not. The "category selection" option is given because sometimes the user is only interested in part of all the system terms. For example, when a user gives the original term pattern such as "Ji/?ns/*n Hu", he is only interested in folder names, values and thesaurus terms, but not the others. Hence, the "category selection" option allows users to select one or more from the five categories: folder name, frame template name, attribute name, value and thesaurus term. The default one is for all the categories.

When proceeding the pattern matching, a user can use the pattern operators to define the original term pattern and decide to use any selection from the three optional selections. Upon receiving the information, firstly, the system will create the candidate set. Based on the category selection, the system terms are selected from these categories to create the candidate set. The candidate set does not contain any duplicate terms. If the user wants the letter case to be insensitive, for all the terms in the candidate set and user's original term pattern, the system will convert them into upper case form. If a user wants the blank space to be insensitive, for all the terms in the candidate set and user's original term pattern, the system will convert all the blank spaces in the middle of a term into one blank character, and cut off all the blank spaces before or after the term. However, the original forms of the candidate terms are still kept after the case processing and blank space processing. At last, for each term in the candidate set and user's original term pattern, we do the pattern matching to find all the matched terms, which are returned to the user.

7.2 Edit Distance Comparison

In the "approximate term matching" component, if the pattern matching is failed to achieve their objectives for some reasons, TEXPROS provides users with another approach, which is called the "edit distance comparison". The basic idea is that a user can give an original term and a matching threshold, which is represented by a positive integer number. The system will automatically find all the similar system terms, whose edit distance from the original term is below the matching threshold; will rank them and return to the users. In this way, users allow to give an approximate spelling without defining any patterns, and the system can help users find the system terms they want.

In the "edit distance comparison", the core problem is how to determine the edit distance between two strings as measured by the minimum cost sequence of "edit operations" needed to change a string into the other, which is referred to as the "string-to-string correction problem" in computer science. In [76], an algorithm is presented, which calculates the edit distance between two strings. The edit operations they considered are: (1) changing one character to another character; (2) deleting one character; (3) inserting one character. They also define Υ as the arbitrary cost function which assigns to each edit operation $a \rightarrow b$ a nonnegative real number $\Upsilon(a \rightarrow b)$.

In TEXPROS, this algorithm [76] is used as the core algorithm for our "edit distance comparison". However, this algorithm is for the general "string-to-string correction problem". For applying this algorithm in our TEXPROS, we define their cost function as : $\Upsilon(a \rightarrow b)$ is 1 if a != b; is 0 if a = b. In other words, the cost for the three edit operations are all 1. Based on this cost function in TEXPROS, the edit distance between two strings is the minimum number of edit operations needed to change from one string into the other.

Also, based on our cost function, we give a simplified version of this algorithm, which is shown as follows:

```
ED_ALGO( term sub_st, term sub_ct)
1. declare two dimensional array D[length(sub_st), length(sub_ct)];
2. D[0,0]=0;
3. for i=1 to length(sub_st)   D[i,0]=i;
4. for j=1 to length(sub_ct)   D[0,j]=j;
5. for j=1 to length(sub_ct)
      for i=1 to length(sub_st)
         if(sub_st<i> == sub_ct<j>)
           then D[i,j] = min(D[i-1,j-1], D[i-1,j]+1, D[i,j-1]+1);
         else
           D[i,j] = min(D[i-1,j-1]+1, D[i-1,j]+1, D[i,j-1]+1);
6. return D[length(sub_st), length(sub_ct)];
```

This ED_ALGO algorithm is used as the core algorithm for the "edit distance comparison" in TEXPROS. This algorithm calculates the edit distance between two strings. In other words, the degree of similarity between two terms can be determined in terms of edit distance between two terms. For example, the edit distance between "software taster" and "software tester" is 1. The edit distance between "software taster" and "software manager" is 4. Therefore, the string "software taster" is more similar to the string "software tester" than the string "software manager". In this way, we can calculate

the edit distance between the system terms and the original term supplied by a user; find those similar system terms, rank them based on the edit distance and return them to the users.

If the "edit distance comparison" is applied, the user needs to supply the original term and a matching threshold. The matching threshold is the maximum edit distance allowed between the original term and result terms. Also, the three optional selections for "pattern matching": "letter case sensitive", "blank space sensitive" and "category selection", can also be applied in the "edit distance comparison". Similar to the "pattern matching", based on the three optional selections, the system will firstly create the candidate set. Secondly, the length restriction will be used to narrow down the candidate set. Generally speaking, the users provide the matching threshold, which is the maximum edit distance allowed between the original term and result terms. Based on our algorithm, if the matching threshold is mt, the length of result term must be in the range of [length(orignal term) - mt, length(original term) + mt]. So, using this length restriction, we can delete the candidate terms, which do not fit in the length range. Thirdly, we shall do the exact matching between the candidate terms and original term. There may be some candidate terms, that differ from the original term only by the letter case and blank space, such as "software tester" and "Software Tester". If we find such terms, we will delete them from the candidate set and directly put into the result term list. In other words, they satisfy the approximate matching with the highest priority. After all these, we will use ED_ALGO algorithm to calculate the edit distance between the original term and each term in the candidate set. For a candidate term, if the edit distance is larger than the matching threshold, then this term does not satisfy the users' request and will be deleted

from the candidate set. If the edit distance is not larger than the matching threshold, then this term satisfies the users' request and will be kept in the candidate set. Finally, all the terms left in the candidate set will be ranked by their edit distance and put into the result term list. In this way, the system can return the result list to users and help them find the system term that they want. For the previous "Jianshun Hu" example, a user can use the "edit distance comparison" to get help from the system. Say, a user supplies the original term as "Jinsun Hu" and a matching threshold 3, the system will return a resulting list as: "Jianshun Hu", "Jinsheng Hu", etc. In thais way, a user can find the correct term that he wants is "Jianshun Hu".

In the "edit distance comparison", there is another important feature, called the history buffer. TEXPROS is a personal document processing system. Because of the typing habit and term-usage habits, the approximate term matching, for a specific application domain, can be possibly localized. For the "Jinsun Hu" example, even the system found the correct term "Jianshun Hu" for the user, he still may not remember the correct spelling. For the next time, this user is still using "Jinsun Hu". However, the most possible result of the approximate term matching for "Jinsun Hu" is "Jianshun Hu", which is the same as the last time. Although the "edit distance comparison" can find the correct term for the user, it is a time consuming job. If we can store user's "edit distance comparison" matching result in history buffer, for some cases, we can directly provide users with suggestions from the history buffer. Without calculating edit distance, it will greatly improve the system performance with a very small amount of overhead cost.

With this motivation, we design the history buffer for the "edit distance comparison" of TEXPROS. The history buffer stores a list of (original term, result term) pairs. Each

time when users uses the "edit distance comparison", the original term and result term that the user selected will be automatically added into the history buffer, if the pair does not exist in the buffer. For example, the pair (Jinsun Hu, Jianshun Hu) will be automatically stored into the history buffer. The usage frequency and latest usage time of these pairs will be also stored for each pair, which is used to rank the list of the pairs in the history buffer. With the history buffer, when user wants to use the "edit distance comparison", the user's original term will be first used to exactly match the original term of each pair in the history buffer. If we find some pairs in the history buffer contains the same original term as user supplies this time, we will get their result terms, create the history list and return the list to the user. For most cases, it will directly hit the user's request. If the user selects one from the history list, the usage frequency and latest usage time of the corresponding pair in the history buffer will be automatically modified. If the user does not want any term of the history list, the system will ignore the history list and continue the edit distance comparison.

In summary, our "approximate term matching" is based on two popular existing techniques, the pattern matching and the edit distance comparison. It can assist users to find the system terms that they want. In this way, when the system cannot find the related terms and documents with respect to the users' original query or the users do not know exactly what the terms they are looking for, TEXPROS can do the approximate term matching and give users cooperative answers. Also, with the accessories processes presented in this chapter, we can let users more precisely define their request, speed up and ease our approximate matching procedure.

In this dissertation, we only present the design of the "approximate term matching", which focuses on the normal term cases. It is not extended to the special kinds of data yet. In TEXPROS, the DataDomain handles the knowledge of special kinds of data, which is not supported by the thesaurus. To implement the approximate matching in DataDomain, the basic idea is to define a [0,1] similarity function for the recognizer of every DataDomain. Therefore, we can tell the similarity between the original term and the DataDomain, instead of a "True/False" answer. However, TEXPROS targets at the general working (application) domain. Users can define any kind of DataDomain. So, we will exploit in our future research how to define a general [0,1] similarity function or several kinds of functions to cover all these cases. Also, our current implementation is based on two existing popular techniques: pattern matching and edit distance comparison. There exist some other techniques, such as soundex, which may improve our current implementation. In our future research works, we will also exploit these techniques to improve the approximate term matching for TEXPROS.

CHAPTER 8

REGISTRATION CENTER

TEXPROS is an intelligent knowledge-based document processing system. From the system viewpoint, its architecture can be divided into an organization of three levels. At the highest level, it has an inference engine, which includes several functional components, such as SCAN/OCR, Classification, Extraction, Filing, Browsing, and others. The knowledge base is at the lowest level, which includes system catalog, thesaurus, DataDomains, etc. Between these two levels, there is a knowledge management level,

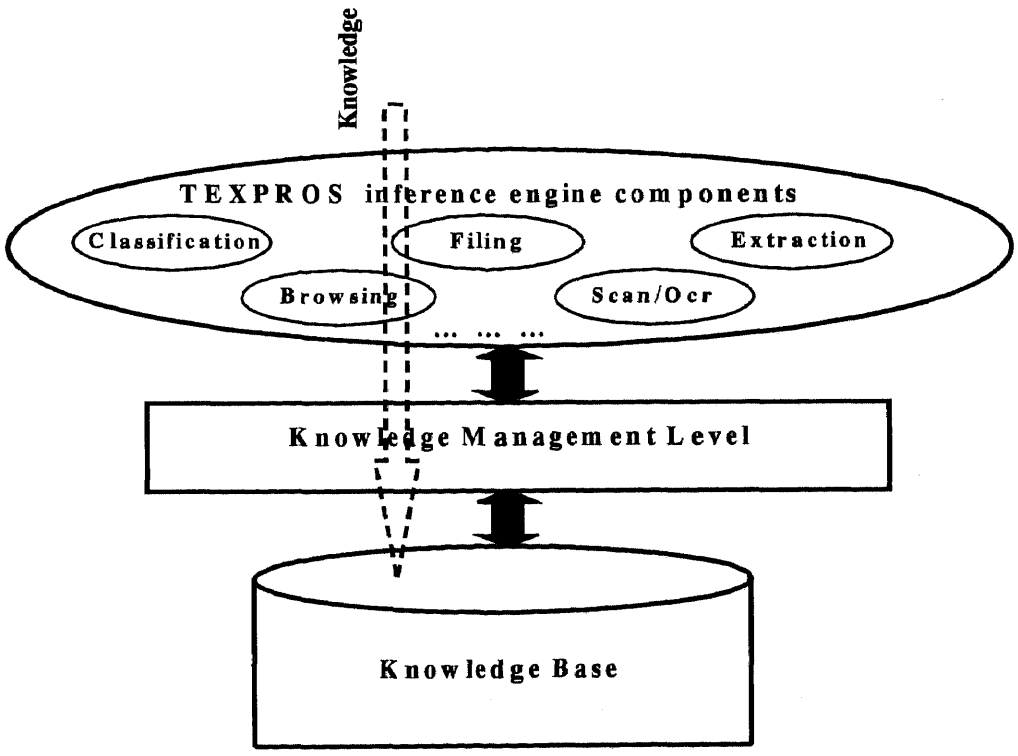


Figure 31 Current System Architecture for TEXPROS

which is used to create a unified knowledge environment to support the inference engine components. The three-level system architecture is shown in Figure 31.

Based on this system architecture, users are allowed to add or delete knowledge from the knowledge base of TEXPROS only through the inference engine components. Adding or deleting knowledge in or from the knowledge base is a kind of one-by-one (or, one instance at a time) behavior. For example, to create a university folder organization, a user must go to the Folder Organization and Filing component to create folders one by one and then link them together. The Folder Organization and Filing component is a part of the TEXPROS' inference engine. The knowledge about the folder organization is created by the user in the inference engine and then stored into the knowledge base. To create the document type hierarchy, a user should go to the DTH component, which is a part of the inference engine, to create frame templates, one at a time, and then store them into the knowledge base. Even to create the thesaurus (namely, the user defined second level thesaurus), the user has to go to the inference engine to create them one at a time. From these examples, we observe that TEXPROS fails to provide users with the flexibility to handle large amounts of knowledge or knowledge packages. In other words, this type of knowledge management structure in TEXPROS is a relatively closed structure.

There are a number of research or commercial knowledge-based systems that employ the closed knowledge management structure [11, 13]. Most of these systems are for a specific working domain. The knowledge for a specific working domain is known and pre-defined before the system is implemented and delivered to the users. In other words, when the users use the system, most of the knowledge is already built into the knowledge base. They need only process the knowledge from the base; and they do not

need to make any changes to it. Even though users need to add in some knowledge, the amount of the knowledge is relatively very small. In this type of applications, the closed knowledge management structure is perfectly suitable for the system.

But, in reality, this is not always the case for TEXPROS. TEXPROS targets at document processing for general working domain. That is, TEXPROS is designed for processing documents in any applications of general working domain. The system is not limited to a specific area. Users can mount TEXPROS from one working domain to another. Generally speaking, the application of TEXPROS to a specific working domain is not as attractive as those systems designed for that specific working domain. For these systems designed for a specific working domain, the working domain knowledge is already pre-defined and therefore the systems are ready to be booted and run. But for TEXPROS, before the system can be used, users must spend a significant amount of time creating, step by step, the specific working domain knowledge. The reason is that TEXPROS employs the closed knowledge management structure. All the knowledge must be added, one by one, into the knowledge base, through the use of the inference engine components, which are basically designed to process documents but not to handle knowledge, especially large amounts of knowledge or knowledge packages. For some specific knowledge, such as DataDomains, it is even very difficult for users to add additional knowledge into the knowledge base through the use of the inference engine components after the system is running.

Now, if TEXPROS requires to be for general working domain, which is a very good feature, the major problem is how to make it easy to add the large amounts of knowledge

or knowledge packages into the knowledge base. We will refer to this problem as the knowledge package add-in problem.

To solve this problem, an easy "+" port is proposed. This easy "+" port is a back door for users to add or delete a package of knowledge into the knowledge base. So, from this viewpoint, TEXPROS is the core system for document processing. Users can use this easy "+" port to add knowledge packages for any specific working domain.

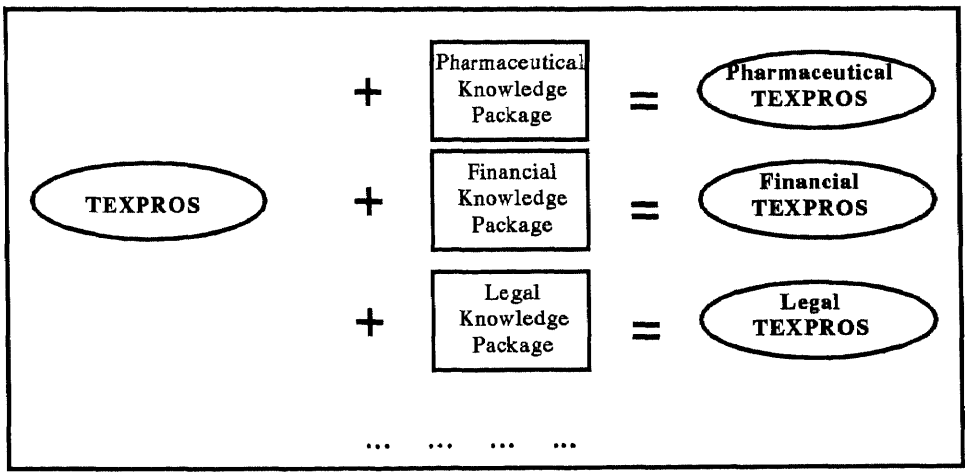


Figure 32 Easy "+" Port for TEXPROS

Consider an example in Figure 32. Before TEXPROS is used by a pharmaceutical company, the users can use this easy "+" port to add a knowledge package of the pharmaceutical working domain into the knowledge base. Then, this TEXPROS becomes an effective system that is applied in the pharmaceutical working domain. Before a law firm uses TEXPROS, we can create a knowledge package for the legal documents environment, and allow the attorneys and legal staffs to add the knowledge package through the use of this easy "+" port. Further more, users can use this easy "+" port to add multiple knowledge packages into the same TEXPROS. For the Legal TEXPROS

example, users can further add a financial knowledge package into the same TEXPROS through the easy "+" port. Then this Financial + Legal TEXPROS can fit in both the financial and legal document environment. Most significantly, the procedure for adding knowledge into the knowledge base uses this easy "+" ports to accomplish the addition without going through the inference engine components. This requires only a little time, possibly just a click. The procedure for adding knowledge via the easy "+" port is designed to be user-friendly so users can easily handle it, without any need of assistance from TEXPROS designers. With this easy "+" port, TEXPROS is flexible to be used for general working domain. In addition to that, it can better compete against other available systems for the specific working domain. Since this easy "+" port targets on the knowledge management, this port should remain in the knowledge management level, not in the inference engine.

Based on the previous discussion, we present the registration center as a component of TEXPROS.

8.1 Registration Center Concept

Registration Center is a component which is used in the knowledge management level of TEXPROS. It allows users to directly add, delete or port a knowledge package into or from TEXPROS. Cooperating with other agents in the knowledge management level, the registration center can dynamically create the unified knowledge environment to support the inference engine components of TEXPROS. Therefore, the registration center is the easy "+" port that we need for TEXPROS. As shown in Figure 33, with this registration center, knowledge can be added into, deleted from, and obtained directly from the

knowledge base by going through the inference engine components, but also directly from the registration center in the knowledge management level. Most importantly, the registration center implements the open knowledge management structure for TEXPROS.

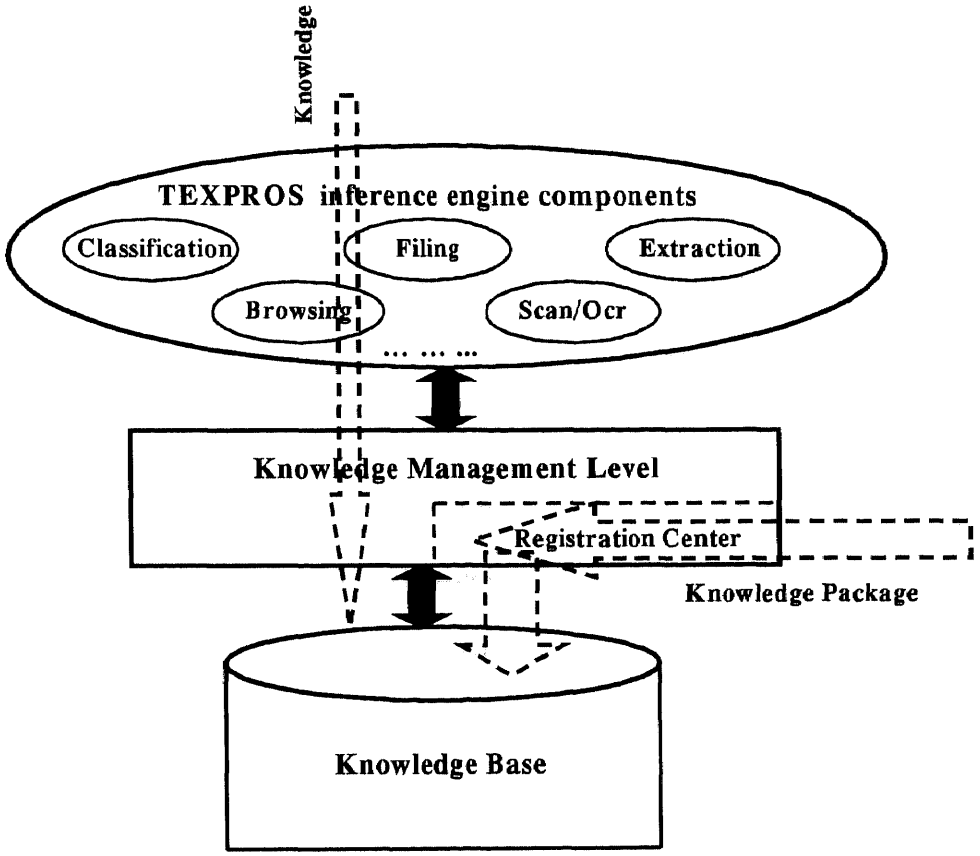


Figure 33 Registration Center in TEXPROS

With the registration center, the overall view of TEXPROS is changed to a core open system.

Here, the "core" means, with the general working (application) domain knowledge, such as the first level thesaurus, all the components of the inference engine, such as Scan/OCR, Classification, Extraction, Filing and Browsing, are combined to form the heart of intelligent document processing system for any working domain. But TEXPROS

is only a core. Using this core, TEXPROS can be operated, but inconvenient, for a specific working domain. But through the registration center, if we add in a pharmaceutical knowledge package, then TEXPROS becomes a PHARMACEUTICAL TEXPROS; if we add in a legal knowledge package, then TEXPROS becomes a LEGAL TEXPROS. In other words, TEXPROS is the bare bone system (or is an engine), which is flexible to tailor to any specific working domain through the use of the registration center.

Here, the "open" means the knowledge management structure for TEXPROS is a kind of open structure. Through the use of the registration center in the knowledge management level, users can directly add or delete a knowledge package into or from the knowledge base. The core TEXPROS provides users with a back door, which is always open to any knowledge packages to be added or deleted conveniently, and gives users great flexibility to tailor the system to any specific working domain.

In TEXPROS, the registration center can manage all kinds of knowledge, such as the knowledge about the folder organization and the document type hierarchy, the thesaurus, the DataDomains. Besides the knowledge packages, the targets of the registration center will be extended to the resources, which include all kinds of knowledge or accessory functional components used to support the work of the core TEXPROS.

Besides the knowledge base, there are some other accessory functional components which can be used to support the work of the core. If we extend the applications of the registration center to these other components, users will have great convenience to control and enpower the system. For example, we can use the registration center to allow users to control several scanners for one system at the same time. Users can use the registration center to update the OCR software. Using the registration center, users can plug

accessory components, such as the "voice recognition component" into TEXPROS. These capabilities give users great convenience to control our core open system. This is the reason why we wish to extend the applications of the registration center from knowledge package to other resources. But basically, the registration center mainly focuses on the control of knowledge packages. In the remainder of this chapter, we will restrict our discussion in the area of knowledge for the registration center.

8.2 Structure of Registration Center

The Registration Center is a back door mechanism, which allows users to add knowledge packages into the system or to delete any knowledge packages from the system. For the convenience reason, it is directly used by the users. The end users of TEXPROS are naïve users who may make mistakes when using the Registration Center. For example, A user wants to register a DataDomain, which in fact may be an irrelevant computer game, into TEXPROS. Or a thesaurus user wants to register a thesaurus package, which is of the form that cannot be recognized by the system. To prevent the problems of this sort, the system must provide users with certain agreements. The requests to add the knowledge into the system or delete the knowledge from the system must be clearly and well defined in TEXPROS. Based on these requests, the system can determine whether the candidate knowledge is acceptable for the system. In TEXPROS, this kind of requests is called a knowledge protocol.

In TEXPROS, a knowledge protocol is a set of requests for its defined kind of knowledge in an acceptable form of TEXPROS. Any knowledge source, which satisfies the knowledge protocol, will be regarded as providing knowledge in an acceptable form of

TEXPROS. The knowledge protocols can be knowledge items stored in database, or protocol functions, depending upon the different protocol requests and their sophisticated nature.

In TEXPROS, there are several kinds of knowledge, such as the folder organization, the document type hierarchy, the DataDomain and the thesaurus. For each kind of knowledge, there must be a knowledge protocol. These knowledge protocols are contained in a knowledge protocol base, which is used to support the Registration Center. This knowledge protocol base is pre-defined and fixed to users. Besides the knowledge protocol base, another important part of the Registration Center is the management services component. There are three parts of the management services of the Registration Center. They are the protocol matching, the consistency checking, and the knowledge updating.

Protocol matching is the first step, if a user wants to register in a knowledge source. The Registration Center will get the proper knowledge protocol from the knowledge protocol base, based upon the type of the knowledge. Then the management service will interpret the protocol and make contact with the knowledge source to confirm whether the coming source is in the acceptable knowledge form (i.e., whether it satisfies the protocol). If it does not satisfy the protocol, then the request of the Registration Center will be rejected and error information will be responded to the user.

Even if the source is in an acceptable knowledge form, it doesn't mean that it can be added into the system or deleted from the system. The knowledge management level is responsible for creating a unified knowledge environment for the inference engine components. The add-in or delete-from operation of knowledge may result in

inconsistency in the current knowledge environment. The consistency checking part of the management service is responsible for checking whether the knowledge environment will still keep the consistent after the operation.

One of the great features of the Registration Center is that it provides users with flexibility and convenience to handle the knowledge base; and, therefore, a unified knowledge environment should be kept dynamically, by means of the knowledge updating management service. After the add-in or delete-from operation of knowledge, knowledge updating management service will cooperate with various parts of the knowledge management level, to update the knowledge base and recreate a unified knowledge environment. Also, when a user wants to export a knowledge package, the knowledge updating service will be responsible for exporting the knowledge and creating the knowledge package.

Besides the management services and the knowledge protocol base, the Registration Center also provides users with an interface. This Registration Center's user interface consists of four parts. A register interface is used to let a user register the knowledge source into TEXPROS. A de-register interface is used to let a user delete some knowledge from the TEXPROS. An export interface is used to let a user export some knowledge for creating a knowledge package. Finally, a testing interface is used to let a user test the current knowledge environment. Figure 34 shows the structure of the Registration Center.

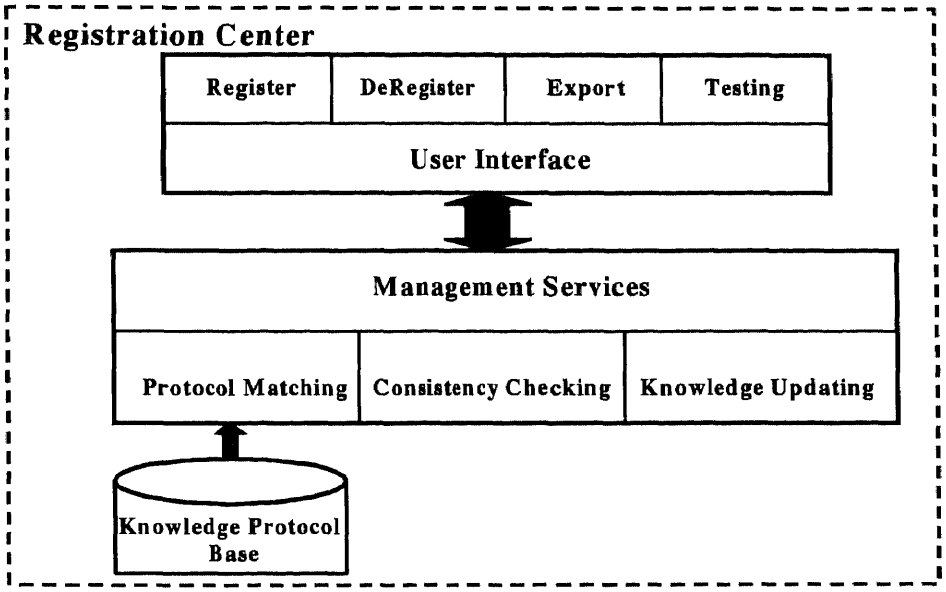


Figure 34 Registration Center Structure

Because there are several kinds of knowledge used in TEXPROS, in the following section, we will use DataDomains as an example to show how the Registration Center works.

8.3 Registration Center for DataDomain

DataDomains can be referred to as a kind of knowledge in TEXPROS, which is used to support the extended thesaurus functionality, pattern recognition and data type operations. In this section, we shall use DataDomains as an example to show how to use the Registration Center to control a specific kind of knowledge. We will focus on the heart of the Registration Center, namely the knowledge protocols and the management services for adding and deleting DataDomains. The discussion here is based on the prototype of TEXPROS, in which we use Java to implement the Registration Center and DataDomains.

8.3.1 Knowledge Protocols for DataDomains

The knowledge protocol for DataDomain consists of five items (namely, the recognizable language, the file extension, the implemented interface, the data member and the root interface).

- Recognizable language: Java
- File extension: .class
- Implemented interface: DataDomain
- Data member: operationList, helpMessage
- Root interface: DataDomain

```
public interface DataDomain
{
   public  boolean  recognizer(String s);
   public  String   converter(String s);
   public  Vector   getOperationList();
   public  boolean  operation(String operator, Vector parameters);
}
```

These five items can be divided into two groups. The first two items are used to confirm that the incoming source of knowledge is written in Java language and is the binary executable code. The last three items are used to define the class of a DataDomain in Java language. Each DataDomain should support the functionality of the thesaurus, the pattern recognition and the data type operations. This can be done by the Java Interface "DataDomain", which is defined in item 5. In this DataDomain's root interface, we define four methods, which are used to support the necessary functionality of DataDomains.

Also, in item 3 the implemented interface, we force that the incoming source of knowledge must inherit all the properties from the DataDomain's Root interface, which enforces the incoming source of knowledge to have these functionalities (namely, the recognizer, the converter, the getOperationList and the operation). In addition, the DataDomain must be self-described. So, in item 4, we enforce the incoming source of knowledge to have these two data members (operationList and HelpMessage), which are used for the self-description.

This knowledge protocol defines what kind of DataDomain source is acceptable by TEXPROS. It is used to support the protocol matching management service to confirm whether a user's registration request satisfies the DataDomain knowledge protocol. Also, it gives users a clear description what is an acceptable DataDomain in the system. Without knowing the detail of the inside of the TEXPROS, the users can develop DataDomains themselves and register them into TEXPROS, simply based on the knowledge protocols. This gives the users great flexibility to tailor TEXPROS to their specific working domain, and therefore enpowers TEXPROS.

8.3.2 Management Services for DataDomains

For DataDomains, there are three parts of management services in the Registration Center, namely the protocol matching, the consistency checking and the knowledge updating. These management services will cooperate with the DataDomain Agent, based on the DataDomain knowledge protocols and the DataDomain base, to dynamically create a unified DataDomain environment for the TEXPROS inference engine components.

8.3.2.1 Protocol Matching for DataDomains: This management services component supports user's request for registering a new DataDomain. When registering a new DataDomain, the user will supply the information of the name and current source location of the new DataDomain. Based on the name and location supplied by the user, the management services component will first check the existence and accessibility of the source of the new DataDomain. If there is no problem, the management services component will load the DataDomain knowledge protocols from knowledge protocol base and begin to match the DataDomain against the protocols. First, the management services component will check whether the file extension of the DataDomain is a ".class", and it will dynamically create the Java class from the source of DataDomain. This will confirm that the source is a Java binary file. After obtaining the Java class for the source, the management services component is able to get all the necessary information for confirming whether the DataDomain source is satisfied with the last three protocol items. If all these have no problem, it means the incoming source is an acceptable DataDomain to TEXPROS. Then the management services component will proceed to the consistency checking of registration. If there is a problem on any of these steps, the registration procedure will stop and error information will be forwarded to the user.

8.3.2.2 Consistency Checking for DataDomains: This management services component supports user's request for registering a new DataDomain or de-registering an existing DataDomain.

When registering a new DataDomain into TEXPROS, the management services component checks whether the new DataDomain is already in the current DataDomain

base, based on the name of the new DataDomain. If it does not exist in the DataDomain base, then proceed the registration procedure to the knowledge updating. If it is already in the DataDomain base, the management services component will ask the user whether he/she wants to use the new DataDomain to overwrite the old one. If the user refuses to overwrite the old one, then the registration procedure is stopped. Otherwise, this information will be forwarded to knowledge updating service.

When de-registering an existing DataDomain from TEXPROS, the management services component uses the name of a DataDomain to check whether it exists in the current DataDomain base. If so, the service component will further check whether there are some other knowledge related to this DataDomain. For example, some attributes in a frame template may use this DataDomain, or some folders' predicates may use this DataDomain. If there is other knowledge that this DataDomain refers to, the system will tell the user these related knowledge and reject the user's request to de-register this DataDomain. Until the user first deals with these related knowledge, he can go back to de-register this DataDomain.

8.3.2.3 Knowledge Updating for DataDomains: This management services component is used to support users' request to register a new DataDomain or de-register an existing DataDomain or export an existing DataDomain.

When registering a new DataDomain, the management services component will check whether it would overwrite an existing DataDomain. If it is a purely new DataDomain, the management services component will copy the source of the new DataDomain into the DataDomain base; and then contacts the DataDomain Agent to add

it into the current DataDomain environment. If it is an overwriting request, the services component will first copy the source of the DataDomain over the old DataDomain, which has the same name, in the DataDomain base. The old DataDomain will be kept a copy in a temporary directory for the case that users still need it. Then the service component proceeds to the knowledge base, finds those knowledge which is related to the overwritten DataDomain, and updates the knowledge using the new DataDomain to keep consistency. Finally, the management services component will contact the DataDomain agent to update the current DataDomain environment.

When de-registering an existing DataDomain, the management services component will delete it from the DataDomain base and contact with DataDomain agent to update the current DataDomain environment. The deleted DataDomain will be kept a copy in a temporary directory for the case that users still need it.

To export an existing DataDomain, the management services component will find this DataDomain from the DataDomain base, make a copy and save it as the file that users request.

CHAPTER 9

CONCLUSION AND FUTURE WORK

In today's information era, intelligent document processing systems have drawn much attention in research and development. The systems are considered to be used for processing, managing and retrieving documents in an office environment and on the Web. The use of Artificial intelligence (AI) technologies in developing document processing systems is to support the intelligent behaviors of the systems. To incorporate the AI technologies into the systems, the central issue is how to create a powerful knowledge base, and to represent and manage different kinds of knowledge to support the systems. Thus, knowledge management itself has become a critical part in document processing systems.

In this dissertation, we presented a new approach to creating the knowledge base and managing different kinds of knowledge for the intelligent document processing systems. Our research target is to design a powerful and applicable TEXt PROcessing System (TEXPROS), which is a knowledge-based intelligent document processing system for general working domain. TEXPROS employs the concept of dual models, which consists of the folder organization and the document type hierarchy, for representing and managing the documents. We introduce a new System Catalog structure to represent and manage the knowledge of the dual models for the TEXPROS. We also present an approach to creating the frame template base and the frame instance base, which are used to represent and store the physical frame templates and frame instances, respectively. To compare with other document processing systems, the presented approach allows not only

classifying the documents based on their conceptual types, but also helps users to categorize (that is, file) the documents into a real-world document storage organization. Also, the unified approach is used to represent and store both the operational and system level information. This gives users greater flexibility to retrieve not only general document information, but also the meta-data information.

We employ an enhanced thesaurus model to support the TEXPROS' functional capabilities. To apply TEXPROS to the general working (application) domain, the new thesaurus model is a two-level structure, which allows users to create and update the knowledge tailored to a specific working domain. The concept of KeyTerm is discussed to improve the performance of TEXPROS. Furthermore, we introduce several operations, which operate on the thesaurus in TEXPROS, and then introduce the internal frame instances to solve the inconsistency problem arising during the thesaurus operations. The concept of Value Composition is also defined, which allows the use of the composite attributes. The use of the composite attributes could be a critical issue in the frame template of TEXPROS, if it is not treated properly.

Ambiguity of senses is a common AI problem encountered in document processing systems. A lot of research works focus on the use of pre-defined working domain knowledge or the statistical approach, which analyze words to solve this problem. This statistical approach becomes less effective for a personal system, such as TEXPROS, which does not have a large collection of testing data for specific working domains. In this dissertation, we present an approach called the "Semantic Range" for solving the sense ambiguity problem. This new approach is based on the dual models of TEXPROS, which represent the real-world context environment of documents from the viewpoint of users. It

employs the document level knowledge to solve the sense ambiguity problem without requiring a large collection of testing data. We then present the algorithm of "Semantic Range Evaluation".

Documents are composed of a bundle of text strings. In additional to the string operations, the users need much more support to handle special kinds of data. Most document processing systems lack this kind of support. In this dissertation, we present a new structure "DataDomain" to solve this problem. DataDomain is employed to handle the special kinds of data, and to support the extended thesaurus' functionalities, pattern recognition and data type operations. Furthermore, users can employ DataDomain to tailor the system to specific working domain knowledge. We also present a new agent structure "DataDomain" Agent to hide the details of all DataDomains and maintain a complete DataDomain environment to support the inference engine components. This observes the information-hiding principle.

KeyTerm transformation is a major procedure of the implementation of sense matching of TEXPROS. In this dissertation, we present a KeyTerm Transformation component, which cooperates with thesaurus, Semantic Range and DataDomain, to solve the general KeyTerm transformation problem of TEXPROS. Also, we define KT_Request and KT_Reply as the standard input and output of KeyTerm Transformation component. In that way, TEXPROS' functional components can easily communicate with the KeyTerm Transformation component and get KeyTerm transformation support for their functionalities. In this dissertation, we also investigate the use of two existing popular techniques: pattern matching and edit distance comparison, to implement the approximate

term matching for TEXPROS. Some accessory processes are also given to speed up and ease the approximate matching procedure.

TEXPROS is aimed to be used in a general working domain. A major problem is how to help users port knowledge packages and tailor the system to a specific working domain. To solve this problem, we introduce a component, "Registration Center" into TEXPROS. Based on the "Registration Center", we change the knowledge management structure of TEXPROS from a closed structure to an open structure. It gives users not only greater flexibility to handle knowledge packages, but also helps system developers to empower our system with the capability of integrating various components into our system.

The prototype of TEXPROS is implemented using Java language on Window NT platform. The testing of the prototype proves that the presented knowledge management approach is applicable. It successfully manages the various kinds of knowledge employed by TEXPROS, and greatly supports the intelligent behaviors of system functional components. In the future, we will continue to improve our implementation and do the performance testing based on large amounts of documents. However, there also exist some limitations. In the following, we will discuss these limitations and future research works to resolve them.

In this dissertation, we presented a two level thesaurus model that focuses on the synonym relationship, which is a critical part in document processing systems. However, we need to investigate some other relationships, such as narrow term (NT), broader term (BT) and related term (RT). These thesaurus relationships are also very helpful in document filing and retrieving. We intend to extend the thesaurus model to cover these

relationships. Also, the current two level thesaurus model is designed to support TEXPROS as a personal document processing system. When TEXPROS is used for multi-user environment, the two-level structure is not enough. In our future research, we will also investigate the multi-level thesaurus model to support TEXPROS as a multi-user document processing system.

Currently, when dealing with ambiguous terms in TEXPROS, we only consider the "system-ambiguous" cases. In other words, users are always responsible for giving the correct KeyTerm in case system fails. But in the real world, in some cases, terms may be also ambiguous to users, which we called "user-ambiguous" cases. How to handle these "user-ambiguous" cases is another direction of our future research works.

Approximate term matching is employed to give users cooperative assistance to find the system terms that they want. Till now, our design of "approximate term matching" focus on the normal term cases. It is not extended to the special kinds of data yet. To implement the approximate matching in DataDomain, the basic idea is to define a [0,1] similarity function for the recognizer of every DataDomain. Therefore, we can tell the similarity between the original term and DataDomain, instead of a "True/False" answer. But the problem is that TEXPROS targets at the general working (application) domain. Users can define any kind of DataDomain. So, we will exploit in our future research how to define a general [0,1] similarity function or several kinds of functions to cover all these cases. Also, our current implementation is based on two existing popular techniques: pattern matching and edit distance comparison. There exist some other techniques, such as soundex, which may improve our current implementation. In the future, we will also investigate these techniques to improve the approximate term matching for TEXPROS.

REFERENCE

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

2.E. Bertino, F. Rabitti and S. Gibbs, "Query Processing in a Multimedia Document System," *ACM Transactions on Office Information Systems*, vol. 6, no. 1, pp. 1-41, January 1988.

3.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.

4.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.

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

6.S. Christodoulaskis, M. Theodoridou, M. Ho, and A. Pathria, "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and System," *ACM Trans. On Office Information Systems*, vol. 4, no. 4, pp. 345-383, 1986.

7.W.W. Chu, Q. Chen and R. Lee, "Cooperative Query Answering via Type Abstraction Hierarchy," in S.M. Deen, editor, *Cooperating Knowledge Based Systems*, pp. 271-292, Elseview Science Publishing Co., Inc., New York, 1991.

8.W.W. Chu, Q. Chen and A. Hwang, "Query Answering via Cooperative Data Inference," *Joint Intelligent Information System*, vol. 3, pp. 57-87, 1994.

9.W.W. Chu and Q. Chen. "A Structured Approach for Cooperative Query Answering," *IEEE Transaction on Knowledge and Data Engineering*, vol. 6, no. 5, pp. 738-749, October 1994.

10. D.M. Cleal and N.O. Heaton, *Knowledge-Based Systems: Implications for Huamn-Computer Interfaces*. Ellis Horwood Limited, West Sussex, England, 1988.

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

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

13. B.V. Dobrov, N.V. Loukachevitch and T.N. Yudina, "Conceptual Indexing Using Thematic Representation of Texts," in E.M. Voorhees and D.K. Harman, editors, *NIST Special Publication 500-240: The Sixth Text Retrieval Conference (TREC-6)*, pp. 403-414, Department of Commerce, National Institute of Standards and Technology, 1998.

14. S. Doong, *A Folder Organization Model for Office Information Systems: Exploring its Architectural Expressive Power and Predicate-Based Filing*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, May 1998.

15. X. Fan, *Knowledge-Based Document Filing for TEXPROS*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, May 1998.

16. 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.

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

18. R. Fikes and T. Kehler, "The Role of Frame- based Representation in Reasoning," *Communications of the ACM*, vol. 28, no.9, pp. 904-920, 1985.

19. 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.

20. 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.

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

22. X. Hao, J. Wang, M. Bieber and P.A. Ng, "A Tool for Classifying Office Documents," in *Proceedings of the 5th International Conference on Tools with Artificial Intelligence*, Boston, MA, pp. 427-434, November 1993.

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

24. X. Hao, J. Wang and P.A. 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.

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

26. G. Hirst, *Knowledge Representation Problems for Natural Language Understanding*, Springer-Verlag, New York, 1988.

27. C. Huser, K. Reichenbeger, 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.

28. D.D. Jaco and G. Garbolino, "An Information Retrieval System Based on Artifical Intelligence Techniques," in *Proceedings of the 1986 ACM Conference on Research and Development in Information Retrieval*, Pisa, Italy, pp. 214-220, September 1986.

29. Y. Jing and W. Bruce Croft, "An Association Thesaurus for Information Retrieval," 1994, Available: *http://cobar.cs.umass.edu/pubfiles/jingcroftassocthes.ps.gz* [July 2, 1998].

30. K. S. Jones and D.M. Jackson, "The Use of Automatically-Obtained Keyword Classifications for Information Retrieval," *IP&M*, vol. 5, pp. 175-201, 1970.

31. K. S. Jones and R.M. Needham, "Automatic Term Classification and Retrieval," *IP&M*, vol. 4, pp. 91-100, 1968.

32. A.R. Kaye and G.M. Karam, "Cooperating Knowledge-Based Assistants for the Office," *ACM Transactions on Office Information Systems*, vol. 5, no. 4, pp. 297-326, October 1987.

33. L. Kerschberg, "Expert Database systems: Knowledge/Data Management Environments for Intelligent Information Systems," *Information Systems*, vol. 15, no. 1, pp. 151-160, 1990.

34. X. Li, J. Hu, X. Fan, C.Y. Wang and P.A. Ng, "Automated Document Filing and Retrieval System: An Overview," in *Proceedings of 3rd Biennial World Conference on Integrated Design and Process Technology*, Berlin, Germany, pp. 231-241, July 1998.

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

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

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

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

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

40. E. Lutz, H. Kleist-Retzow, and K. Hoernig, "MAFIA – An Active Mial-Filter-Agent for an Intelligent Document Processing Support," in S. Gibbs and A. A. Verrijn-Stuart, editors, *Multi-User Interfaces and Applications*, pp. 16-32, 1990.

41. T. Malone, K. Grant, K. Lai, R. Rao and D. Rosenblitt, "Semistructured Messages are Surprisingly Userful for Computer-Supported Coordination," *ACM Trans. On Office Information Systems*, vol.5, no. 2, pp. 115-131, 1987.

42. J. Martin, *The Wired Society: A Challenge for Tomorrow*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

43. B. McCune , R. Tong, J. Dean and D. Shapiro, "RUBRIC: A System for Rule-Based Information Retrieval," IEEE Trans. on Software Engineering, vol. SE-11, no. 9, pp. 939-945, September 1985.

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

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

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

47. Y. Qiu and H.P. Frei, "Concept Based Query Expansion," in *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Pittsburgh, PA, pp. 160-169, June 1993.

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

49. R. Rada and B.K. Martin, "Augmenting Thesauri for Information Systems," *ACM Trans. on Office Information Systems*, vol. 5, no. 4, pp. 378-392, October 1987.

50. A.R. Rao and R. Jain, "Knowledge Representation and Control in Computer Vision System," *IEEE Expert*, pp. 64-79, Spring 1988.

51. V.J. Rijsbergen, D.J. Harper and M.F. Porter, "The Selection of Good Search Terms," *IP&M*, vol. 17, pp. 77-91, 1981.

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

53. G. Salton, "Automatic TermClass Construction Using Relevance - A Summary of Word in Automatic Pseudoclassification," *IP&M*, vol. 16, no. 1, pp.1-15, 1980.

54. G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, Reading, MA, 1988.

55. G. Salton, C. Buckley and C.T. Yu, "An Evaluation of Term Dependence Models in Information Retrieval,"in *Lecture Notes in Computer Science*, pp. 151-173, Engl Springer-Verlag, London, 1983.

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

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

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

59. P. Schone, J.L. Townsend, T.H. Crystal and C. Olano, "Text Retrieval via Semantic Forests," in E.M. Voorhees and D.K. Harman, editors, *NIST Special Publication 500-240: The Sixth Text Retrieval Conference (TREC-6)*, pp. 761-774, Department of Commerce, National Institute of Standards and Technology, 1998.

60. P. Shoval, "Principles, Procedures and Rules in an Expert System for Information Retrieval," *Information Processing & Management*, vol. 21, no. 6, pp. 475-487, 1985.

61. S. Small and C. Rieger, "Parsing and Comprehending with Word Experts (a theory and its realization)," in W. Lehnert and M. Ringle, editors, *Strategies for Natural Language Processing*, pp. 89-147, LEA Press, Mahwah, NJ, 1982.

62. P.J. Smith, S.J. Shute, D. Galdes and M.H. Chignell, "Knowledge-Based Search Tactics for an Intelligent Intermediary System," *ACM Transactions on Information Systems*. vol. 7, no. 3, pp. 246-270, July 1989.

63. M. Sneoeck 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.

64. P. Szolovits, "Knowledge-Based Systems: A Surey," in M. Brodie and J. Mylopoulos, editors, *On Knowledge Base Management Systems*, pp. 339-352, Springer Verlag, New York, 1986.

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

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

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

68. 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.

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

70. W. Ulrich, "Introduction to Electronic Mail and Implementation Considerations in Electronic Mail," in *AFIPS Conference Proceedings*, Arlington, VA, pp. 485-492, 1980.

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

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

73. C.Y. Wang, Q. Liu and P.A. Ng, "Intelligent Browser for TEXPROS," in *ISATED Proceedings of International Conference on Intelligent Information Systems Technology*, Grand Bahama, Island, Bahamas, pp. 388-398, December 1997.

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

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

76. R. Wanger and M. Fischer, "The String-to-String Correction Problem," *Journal of the Association for Computing Machinery*, vol. 21, no. 1, pp. 168-173, January 1974.

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

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

79. M.E. Williams, "Electronic Databases," *Science*, pp. 445-446, April 1985.

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

81. S.K.M. Wong and W. Ziarko, "A Machine Learning Approach to Information Retrieval," in *Proc. of the 1986 ACM Conf. on Research and Development in Information Retrieval*, Palazzo dei Congressi, Pisa, Italy, pp. 228-233, September 1986.

82. C.T. Yu, C. Buckley, K. Lam and G. Salton, "A Generalized Term Dependence Model in Information Retrieval," *Information Technology: Research and Development*, vol. 2, pp. 129-154, 1983.

83. Z. Zhu, *Document Filing Based upon Predicates*, Ph.D. dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, October 1994.

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

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