ABSTRACT

A FOLDER ORGANIZATION MODEL FOR OFFICE INFORMATION SYSTEMS: EXPLORING ITS ARCHITECTURAL EXPRESSIVE POWER AND PREDICATE-BASED FILING

by
Simon Doong

This dissertation presents an Internal Folder Organization (I-ORG) which supplements the architectural deficiencies of the existing model - the User Folder Organization (U-ORG), to electronically model a person's filing system in the modern office environment. An I-ORG folder organization gives a logical representation of how documents of the same or different kinds are related and grouped into folders based on predefined premises. Our model is represented by a Rooted Direct Acyclic Graph (RDAG). Each node in the graph represents a folder; and folders are related by "subfolder relationship" (for capturing the "and" relation) and "virtual-folder relationship" (for capturing the "or" relation). Each folder in the organization has a criterion, specifying in terms of a local predicate, which governs the document filing for that folder. The dissertation also investigates how the new model demonstrates its architectural support in the four functional areas: (1) Construction - It reduces the complexity of predicate specifications; (2) Filing - It improves the performance of document distribution; (3) Retrieving - It facilitates system responsiveness to queries, especially for the documents which are frequently requested by the user; and (4) Reorganization - It reduces the volume of documents to be redistributed when the folder organization is modified. The justifications of our model in possession of critical architectural attributes to support the

above functions efficiently and effectively are presented throughout this dissertation, which lead us to draw an initial conclusion - our proposed model is architecturally superior over the other representative models.

In comparison with the I-ORG, which is operational more efficient, the U-ORG has its simplicity because it maintains only a single type of link. Therefore, the implementation of the system can have two models which represent the folder organization at two different levels: the user interface level (or the external representation using U-ORG), and the system execution level (or the internal representation using I-ORG). Interoperabilities between the two models needs to be well-coordinated and kept transparent to the user while the system optimizes its performance by utilizing the architectural strength from both models.

The dissertation also investigates the transformation between the two models and proposes a step-locked reduction algorithm to accomplish that task. This transformation capability provides to the user more flexibilities to specify predicates when his folder organization is created and represented by an U-ORG. This U-ORG is transformed and fine-tuned into a content-equivalent folder organization represented by an I-ORG, which optimizes the overall predicate structure to help improve the functional performance. In such a final representation, each folder in the organization only associates with a single atomic predicate.

A FOLDER ORGANIZATION MODEL FOR OFFICE INFORMATION SYSTEMS: EXPLORING ITS ARCHITECTURAL EXPRESSIVE POWER AND PREDICATE-BASED FILING

by
Simon Doong

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

August 1998

APPROVAL PAGE

A Folder Organization Model for Office Information Systems: Exploring Its Architectural Expressive Power and Predicate-Based Filing

Simon Doong

4/18/98

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

4/18/98

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

4/18/98

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

4/18/98

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

4/18/98

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

BIOGRAPHICAL SKETCH

Author: Simon Doong

Degree: Doctor of Philosophy

Date: August 1998

Undergraduate and Graduate Education:

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

- Master of Computer Science,
  State University of New York at Stony Brook, Stony Brook,
  New York, 1983

- Master of Mathematics,
  University of Southern Mississippi, Hattiesburg, Mississippi
  1980

- Bachelor of Mathematics,
  National Central University, Chung-Li, Taiwan, ROC, 1978

Major: Computer Science

Presentations and Publications:

Simon Doong, C.S. Wei, Xien Fan, D.C. Hung and Peter A. Ng, "A Folder Organization Model in the Office Environment", submitted to the 4th International Conference on Information Systems Analysis and Synthesis, Orlando, Florida, July 1998.

Simon Doong, Xien Fan, C.S. Wei and Peter A. Ng,"A Process for Constructing a Personal Folder Organization", submitted to the 4th International Workshop on Multimedia Database Management Systems, Dayton, Ohio, June, 1998.

Simon Doong, Ron Curtis and Peter A. Ng, "Document Filing Approaches Using an I-ORG Based Folder Organization", submitted to the 17th International Conference on Conceptual Modeling, Singapore, November, 1998.

TO MY BELOVED FAMILY

ACKNOWLEDGMENT

The author would like to take great pleasure in acknowledging his advisor, Professor Peter A. Ng, for his kindly assistance and remarkable contribution to this dissertation. He spent time and effort to review various drafts of the manuscripts and provided valuable comments and crucial feedback that influenced the final manuscript.

The author also likes to thank his committee members: Dr. D.C. Douglas Hung, Dr. Murat M. Tanik, Dr. Tina Taiming Chu, and Dr. Ronald S. Curtis for their continuous support in reviewing the research progress and providing feedback with valuable comments.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Background

In the modern electronic and multimedia office environment, there is significant demand of using office information systems in order to facilitate automated processing of office documentation [3,10,15,68,69,84,89,95]. The TEXt PROcessing System (TEXPROS) project has developed to foster and integrate many innovative concepts and research efforts in order to exploit and realize the construction of such an intelligent system [91].

The folder organization, as originally proposed in TEXPROS to model a personal filing system in the office environment, is a logical repository system that supports all other structured operations for document filing, storing [48,89,99], retrieving [49,46,47,49,50], reformatting [89,91] and organizational management activities.

In addition to the front-end documentation process where a document is converted into an electronic structured representation, the overall architecture of TEXPROS as shown in Figure 1 is comprised of the following functional components: (1) training, (2) classification [92-94], (3) extraction [28-32,92,94], (4) construction (of folder organization), (5) filing [99-101,103,104], (6) reorganization [48,91,99], and (7) retrieval [44-50,88,108].

Some components in the architecture are supported by a thesaurus [64]. For instance, during the information extraction process, the thesaurus needs to be consulted in order to deal with the ambiguous "kind of" relationships among synonyms. A thesaurus provides functions for clarifying semantically any ambiguities of a given keyword, phrase, or attribute [63]. For handling incomplete or insufficient information during information extraction, an advance and knowledge rich thesaurus is needed which supports a set of intelligent knowledge inferring and deriving capabilities. Although the thesaurus topic is not in the scope of this dissertation, we would like to point out that one of the challenges of the TEXPROS is how does the thesaurus assist to structure, clarify and summarize (extract information from the text pertinent to the users' significance) the unstructured part of a text (e.g., the content of a document in free text form).

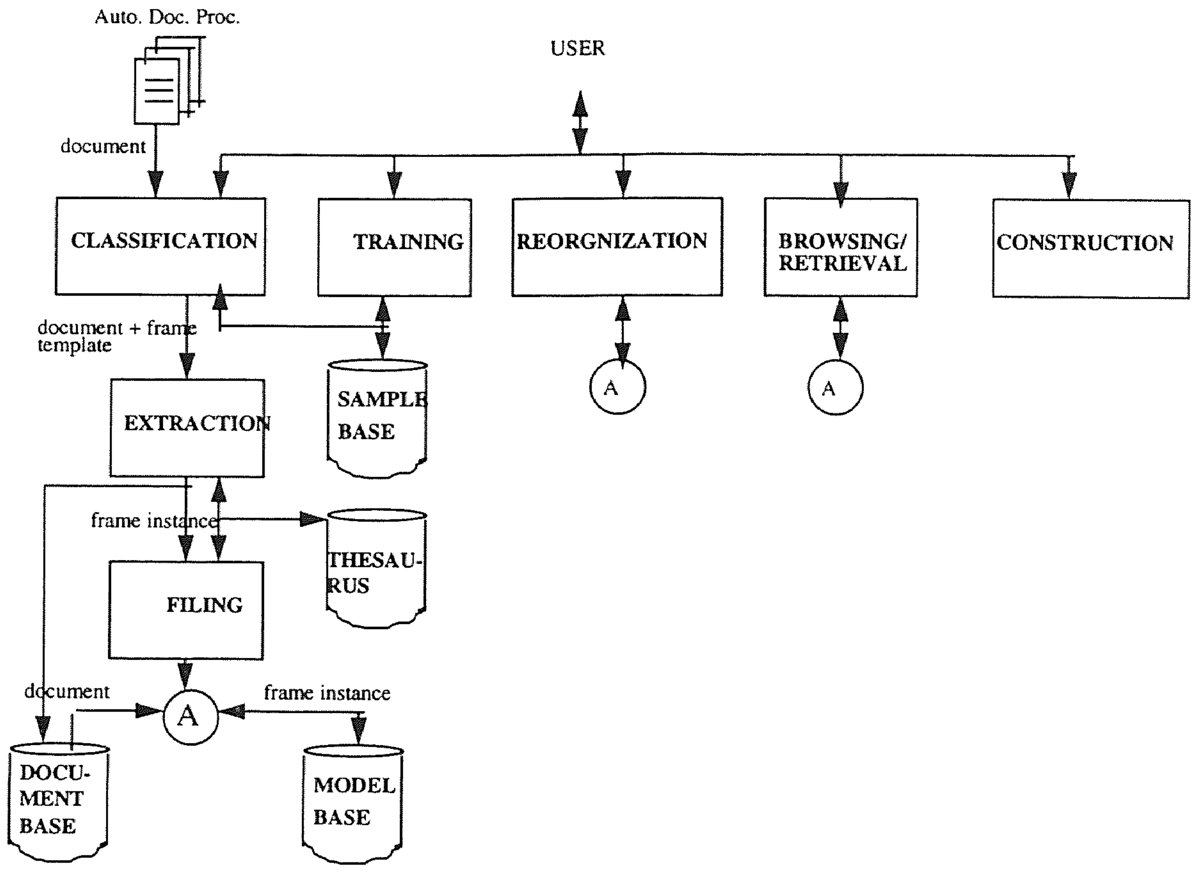


Figure 1 Architecture Overview of TEXPROS System

The process of intelligent document processing is described briefly as follows. A given document imaged with a scanner, is transformed into a tree-like logical structure by the "**classification**" subsystem [29]. This logical structure reflects the document's spatial relationships among identified blocks. Documents of the same type are represented by a collection of tree-like structures, which are stored in the sample base. Each document type is characterized by a set of attributes to form a frame template. To classify this document as a document type, the classification subsystem tries to find appropriate matches between its associated structure tree and the other structure trees in the sample base. Once a close match is identified, the system outputs the corresponding document type in terms of its associated frame template of the matched tree. If however, the system is unable to find a match, the user will be directed to the "**training**" subsystem [30,89] to help the system learn this type of document. Under this situation, the document is either a new type of document or an existing document type with a different format. Its associated tree-like structure is then stored in the sample base for future references.

A user-defined frame template describes the common information fields [76] (called attributes) for a given document type. Once a frame template for a given document is identified, the "**extraction**" subsystem [31] starts assigning each template-defined attribute with values extracted from the given document (filling the template).

The filled frame template becomes a frame instance[1] which is considered a synopsis of the document and is used to represent the original document. However the frame instance

[1] The most significant information is extracted from the original document to fill the frame template which was obtained from the classification subsystem. A filled frame template serves as a synopsis of the original document and is called a Frame Instance (FI). From this point, the original document can be represented by its corresponding frame instance. Each frame instance is stored as an object in the folder organization.

may or may not contain sufficient information (extracted from the original document) for filing. But every frame instance deposited into a folder must at least satisfy the folders' filing criteria of a path from the root to the folder.

The "**construction**" subsystem [48] is used to build a personal folder organization for storing the frame instances. For each folder, a user defines its predicate that serves as the filing criteria for determining the depository of frame instances into the folder. The construction subsystem is employed to define and construct the desired folder organization (including folders, relationships, and predicates), and performs the validity checks whether the folder organization has met the RDAG structure and is free from the predicate syntax error [106].

Instead of filing the *original document*, the predicate-based "**filing**" subsystem [48,91,99,101] processes and distributes its corresponding *frame instance* into appropriate folders in the folder organization. A frame instance will, in most cases, satisfy more than one folder's filing criteria and thus multiple copies of the frame instance may be logically deposited into the organization.

Whether the frame instance of a given document can be deposited into the appropriate folders remains to be investigated. The filing subsystem should guarantee the uniqueness of the filing characteristics. A **loosely defined uniqueness** is that regardless the number of trials, filing the same set of m frame instances always results in depositing a total of n copies of these frame instances into the folder organization. On the other hand, a **tightly defined uniqueness** is that filing the same set of m frame instances into a given organization should produce exactly the same output (namely, the frame instance copies made for each deposited folder are uniquely determined).

We employ an agent-based [48,89,91,107] concept to handle the distribution of frame instances. Consider a folder as an "object" which has attributes and operations. Each folder in the organization is represented by a filing agent [48, 89,91, 100] which is a self-contained autonomous process in the system. Filing agents are activated via filing or reorganization requests, and remain active until the process of distributing (for filing) or redistributing (for reorganization) frame instances into folders is completed. For each folder organization, we shall call the deposit of a frame instance into a folder an object depository (or simply depository).

Since each agent may perform the filing functions independently, each relies on information exchange with the filing agents of other folders to update its current filing state and decide what has to be done next. Agent to agent communications are accomplished via message passing mechanisms.

After the filing is done (we defer the filing discussion until later sections), the user may (1) browse [45,88,108] or query the information via the "**retrieval**" subsystem [32], or (2) propose a reorganization of the existing folder organization via the "**reorganization**" subsystem [48,91,100].

Each filed frame instance with its represented original document must be easily retrievable from the object repository. This object repository may be split into two logically separated repositories, one (represented as a folder organization) for storing frame instances while the other (can be a file structure) is for storing the original documents [104]. Through a well-defined interactive graphical interface which synthesizes information components retrieved from the repositories, the user should be able to browse through either frame instances or any part of the original documents, place queries, and retrieve nearly anything of his/her interest from the documents.

For TEXPROS, any structural change of an existing folder organization is termed reorganization. Reorganizing an existing folder organization includes, but is not limited to, inserting a new folder into the organization and removing a folder from the organization. Inserting a new folder to the folder organization requires re-computing the global predicates of all of the children under this new folder and redistribute the existing frame instances from the parents and the children of this new folder. Removing a folder with or without its subsequent folders from the organization can be a simpler reorganization in comparison with the insertion of a new folder.

Note that any reorganization proposed by the user may be granted or rejected by the system for various reasons. For instance, the proposal of rehoming a folder from its current parent folder to one of its child folders will be rejected by the system since this reorganization proposal violates the Rooted Direct Acyclic Graph (RDAG) architectural constraint which allows no cyclic structure in the folder organization [48,91].

1.2 Folder Organization for Logical Representation

Since the filing, reorganization, browsing and retrieving functions all operate upon a common structural foundation - the folder organization - its representation model plays a key role of performance in the overall system architecture. In addition, this representation model must also model a personal filing system in the real world, help reduce the complexity of specifying folders' criteria, and support the automation for document processing.

A folder organization is a logical representation which gives the user a global view of how the documents are grouped. Each folder in the organization can contain documents of different types. A local predicate is specified by the user for each folder created. This predicate is local to the folder and it governs the filing decision of frame instances for that folder.

The current folder organization model of the TEXt PROcessing System (TEXPROS) was originally proposed in [48,91]. It is represented by a Rooted Direct Acyclic Graph (RDAG) [61,101]. Each node in the graph represents a folder. Each link connecting two nodes represents a potential filing path. The folder on the starting-end of the link is called the parent folder, and the folder on the arrow-end of the link is called the child folder.

A given document arrives at a parent folder A, traverses the filing path which interconnects A and B and deposits in the child folder B if the information extracted from the document satisfies the local predicate of the folder B. Otherwise, the document stops at the parent folder A. Therefore, the links in this organization define the folder relationships for the model. These folder relationships are simple and direct. We refer to this folder organization model as the User - ORGanization or U-ORG for simplicity. However, this existing model also presents some architectural constraints such as it is unable to model a more complicated folder organization without introducing the complexity of specifying predicates. For example, given a folder organization, if the user wants a new folder F to be added to the organization in such a way that the new folder only contains common documents collected from a set of the existing folders, say F1, F2 and F3, it requires the user to carefully specify the predicate for the folder F so the documents which are not common to the three parent folders would not be deposited into this new folder. Between F and any of its parent folders F1, F2 and F3, there exist three possible filing links, (F1, F), (F2, F) and (F3, F). Since a qualified document which is common to F1, F2 and F3, may flow into folder F via any of these three links, the predicate specification for the folder F needs to take into account the predicates of all of F's ancestral folders (i.e., F1, F2, F3 and all of their upstream direct or indirect parent folders). Therefore, it is difficult and complicate to specify the right predicate for the folder F. Moreover, the complicated predicates presented in the organization may hurt the system performance in both folder reorganization and document filing.

Our proposal supplements the only filing path relation link found in the U-ORG model with a subfolder relation link to form a new organization model. This new organizational model supplements the architectural deficiency of the existing model because: (1) it models the folder relationship in the previous example without creating unnecessary complexity; (2) it helps the user to simplify predicate specification inputs, and (3) it facilitates both sequential and parallel filing approaches to improve the document filing performance in a distributed operating environment [107]. In contrast to U-ORG, which is simple but limited in modeling capabilities, the proposed new model can be implemented as an Internal representation of a folder ORGanization (I-ORG). In doing so, a two-layer representational hierarchy, tightly coupling a U-ORG and its underlying supportive I-ORG, is proposed. This representational hierarchy constructs a common system foundation for processing, filing, managing, and retrieving documents in the electronic office environment.

Similar to the U-ORG, an I-ORG is also represented by a RDAG in which each node represents a folder, and each of its link represents either of the subfolder relationship or virtual-folder relationship. The subfolder relationship is an "and" relationship, which means that the frame instance of a document is "considered" for depositing into a folder if and only if every of its parent folder with the "and" relation contains a copy of the frame instance of the document. The virtual-folder relationship on the other hand, is the existing "or" relationship, which specifies that the frame instance of a document only needs to appear in at least one of the parent folders with the "or" relation.

For the I-ORG, filing a given frame instance from a parent folder into a child folder requires the frame instance satisfying the following two conditions: (1) it qualifies the child folder's predicate specification, and (2) it does not violate the relational constraint.

The relational constraint specifies that some of the parent folders of this child folder must contain a copy of the frame instance before the filing of the frame instance takes place for the child folder. This constraint is automatically enforced in a folder organization where the two relationships are used to link folders together. For performance reasons, since the filing agent of each folder is a fully autonomous entity (it is responsible for its own folder's filing process), the agent won't initiate the filing (evaluates the associated predicate) until the relational constraint of the folder is checked and passed. As an example, in Figure 2, filing a frame instance fi into a given folder F5, the following two conditions constitute the relational constraint between the parent folders F1, F2, F3 and F4 and the child folder F5:

(1) the folders F1 and F2 must contain fi, and

(2) either the folder F3 or the folder F4 contains fi.

Two major objectives behind the new model for designing such a structure for representing a folder organization are to reduce the complexity of specifying predicates and to achieve better filing and reorganizing performance. Although the model introduces an additional folder relationship, its contributions in reducing complexity of predicate specification by representing a good portion of complicated predicate specification in terms of proper links yields a superior architectural expressive power over the old model. With this enhanced folder organization model, users can easily create, attach or insert folders into their folder organization without too much concern whether they have translated correctly the desired filing rules for the folders into complicated predicates. In Chapter 2 and Chapter 3, we will further establish this statement by giving additional examples.

1.3 Filing System for Document Distribution

Many document processing systems have been developed in the past dealing with the topics of filing and retrieving [6,7,19,24,64,69,71,79,82]. We use the following example to describe the frame instance distribution (filing) in the folder organization which is represented by the existing model or the U-ORG. As shown in the Figure 3, the folder organization contains six folders, F1, F2, F3, F4, F5, and F6 with respective local predicates P1, P2, P3, P4, P5 and P6. A total of nineteen copies of the frame instances (f1, f2, f3, f4 and f5) are distributed in the folder organization with the rooted folder (F1) maintaining a copy of all frame instances. From the distribution, it is easy to see that the frame instance "f1" satisfies the predicates P1, P2, P4 and P6; therefore, a copy of "f1" can be found in each of the four associated folders F1, F2, F4 and F6; the frame instance "f2" satisfies the predicates P1, P2, P3, P5 and P6, so a copy of "f2" is found in each of the five associated folders F1, F2, F3, F5 and F6. Similarly, the frame instance "f3" satisfies the predicates P1, P2, P3 and P6, and a copy of "f3" is found in each of the folders F1, F2, F3 and F6, and so on for the remaining frame instances f4 and f5.

A file path from the rooted folder to a folder where a given frame instance is deposited simply indicates the frame instance flowing along the links from the rooted folder to the folder in the folder organization. For instance, the frame instances f1, f2 and f3 are filed into the folder F6 from a path starting from the rooted folder F1, following through F2 before reaching the folder F6. Frame instance f1 is the only frame instance filed into the folder F6 from a path starting from the rooted folder F1, following through the folder F4 and then the folder F6. The filing of frame instances along different paths are independent from each other.

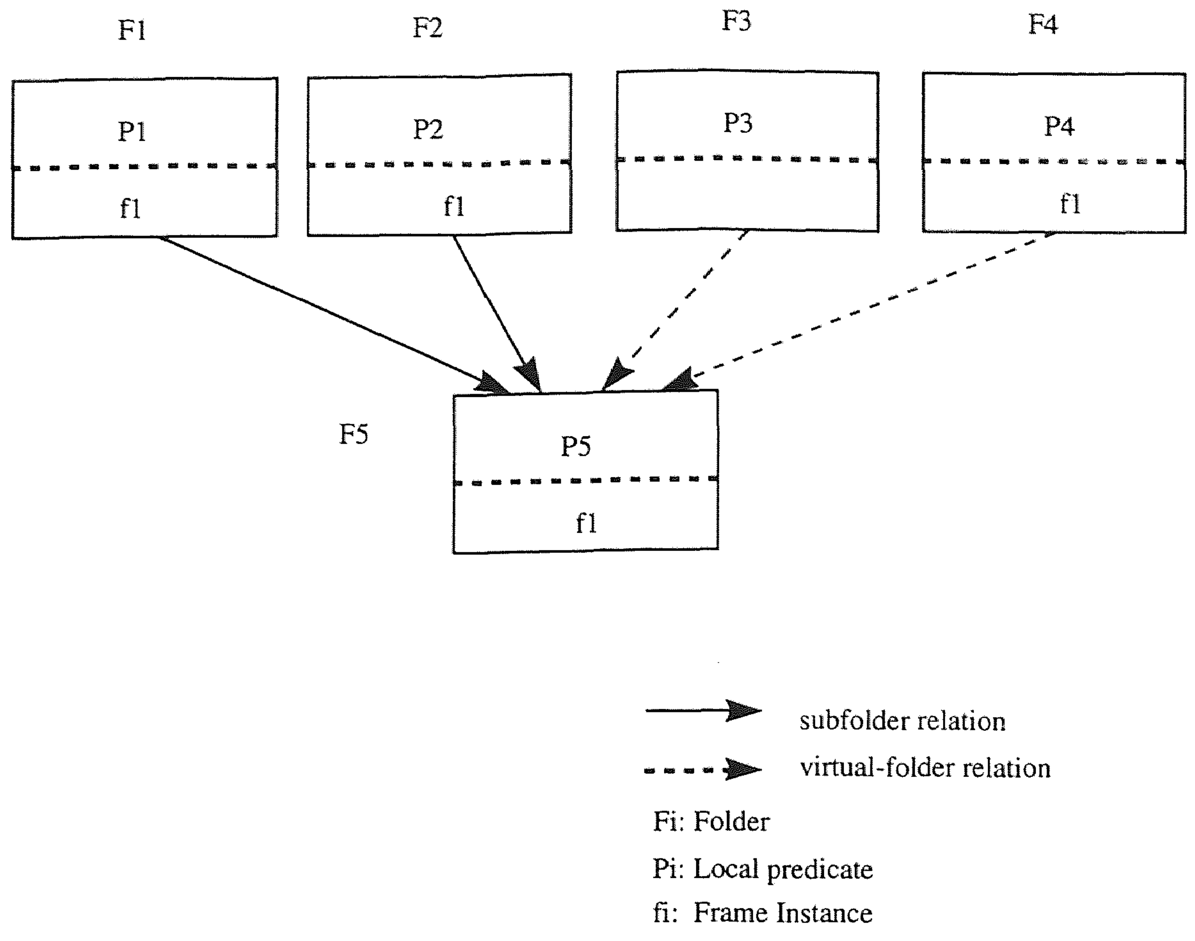


Figure 2 Relational Constraint

In contrast to Figure 3, Figure 4 gives an example of an I-ORG based folder organization. As shown in the figure, the folder organization contains the same six folders, F1, F2, F3, F4, F5 and F6 with respective predicates P1, P2, P3, P4, P5 and P6. There are only two links of the virtual-folder relationship, (F4,F6) and (F5,F6) in this organization, and the rest of the links are of the subfolder relationship. A total of sixteen copies of the frame instances (f1, f2, f3, f4 and f5) in this example are distributed in the folder organization. Likewise, the rooted folder (F1) keeps a copy of all frame instances. From the distribution, it is easy to see that the frame instance "f1" satisfies the predicates

P1, P2 and P4; therefore, a copy of "f1" can be found in each of the three associated folders F1, F2 and F4. The frame instance "f2" satisfies the predicates P1, P2, P3, P5 and P6, and therefore a copy of "f2" is found in each of the five associated folders F1, F2, F3, F5 and F6. Similarly, the frame instance "f3" satisfies the predicates P1, P2 and P3, and a copy of "f3" is found in each of the folders F1, F2 and F3, and so on for the remaining frame instances f4 and f5.

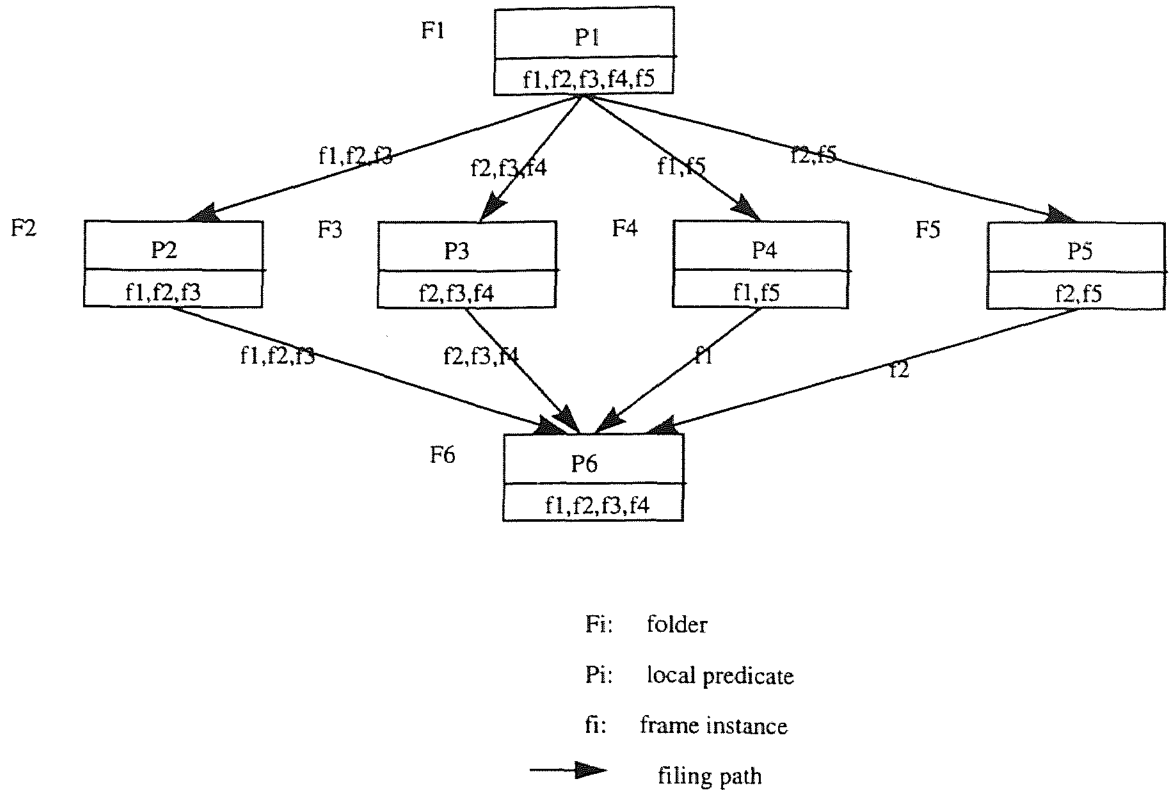


Figure 3 Frame Instance Distribution in U-ORG

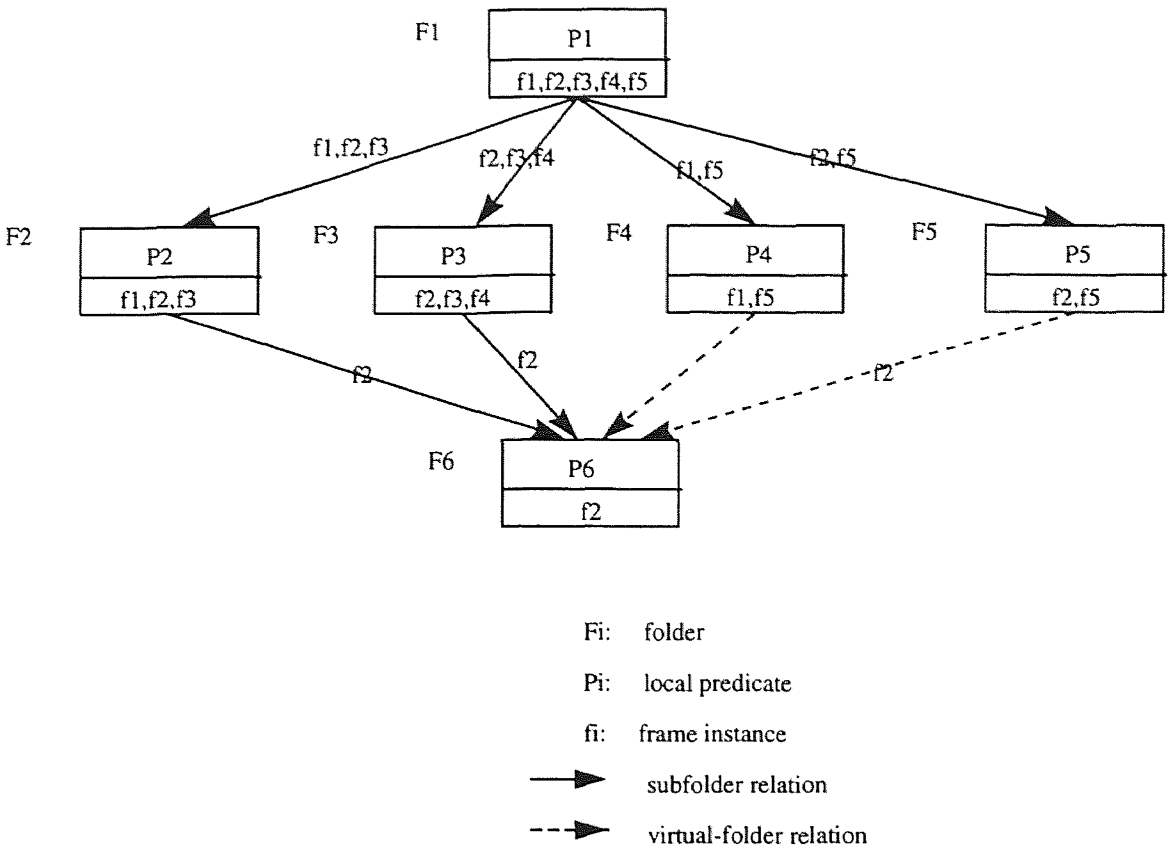


Figure 4 Frame Instance Distribution in I-ORG

However, the folder F6 contains only one frame instance (f2) instead of the four instances (f1, f2, f3 and f4) as shown in Figure 3. To deposit a frame instance in the folder F6, the frame instance has to meet the folder F6's predicate and the relational constraint of the folder F6 which states that the frame instance has to be in both folder F2 and folder F3, and in one of the folders F4 or F5.

To further support the architectural expressive power of the new organizational model, we propose three support models to minimize the predicate evaluation task as the filing objective. First of all, we propose a "complete-parent-first" filing algorithm which employs the well-known topological sorting and its underlying supportive Depth-First

Search (DFS) technique to accomplish the filing tasks. This filing algorithm is to do the filing of frame instances in an sequential manner. The processing and filing performance of this filing model are analyzed and evaluated. The other two filing models, namely the "broadcast model" and the "status relay model", are proposed to complete the filing of frame instances in a parallel fashion. The significance of these two filing models are twofold: firstly, the frame instance distribution process is more dynamic which relies more on the message (filing status and filing result) exchanges among folder agents.

Again, because each filing agent is a self-contained and fully autonomous process, it determines its own filing pace, and therefore, the filing order becomes less predictable. Secondly, the parallelism of filing perfectly fits into our overall object-oriented design architecture in which each of the basic components (folders, folder relationships, links, predicates, frame instances, documents, etc.) is treated as an object. Notable performance savings in parallel filing can be expected as the system is implemented, with object favor, in distributed environments.

Another objective for designing a filing system under the new organization model is to ensure that the frame instances are deposited into the right folders. It is not the focus of our research; nevertheless, it can be partially supported by redefining the predicates to prevent the frame instances from false dropping [103,104,105].

The rest of this dissertation is organized as follows: Chapter 2 defines the problems with the existing folder organization model. Chapter 3 describes an approach to modeling a personal folder organization by utilizing two types of folder relationships: the subfolder relationship and the virtual-folder relationship. Chapter 4 presents a sequential filing model which performs the filing on an I-ORG based folder organization. In addition, a process with its performance analysis of the filing algorithm "Complete-Parent-First" for the sequential filing model is discussed. Chapter 5 presents a simple performance model and explains the filing advantages of that the new model possesses. Chapter 6 describes the two new filing models which distribute frame instances dynamically. Chapter 7 explores the performance advantages of the new filing models. Chapter 8 explores the performance superiority of the enhanced model over the existing model in folder reorganization and document retrieval activities. Chapter 9 provides implementation guidelines which explain why the new models are more suitable as an application candidate in the object-oriented design. Chapter 10 investigates the transformation issues between U-ORG and I-ORG models. Chapter 11 summarizes what has been accomplished in this work. Chapter 12 discusses some issues observed during the present work which may provide a guideline for our future research directions.

CHAPTER 2

FOLDER ORGANIZATION: USER FOLDER ORGANIZATION (U-ORG)

As we mentioned in the previous chapter, our essential goal in designing an intelligent document processing system is to mimic robustly a personal filing system with the related operations in the office environment [5]. There are two major concerns: firstly, this intelligent system must provide a structured mechanism to better organizing office documents in a consistent view, and secondly, this intelligent system must support fast and correct information retrieving. Taking these two concerns into system design considerations, the establishment of a concrete document repository foundation or the introduction of a folder organization for supporting automated document processing and management, and the organizational and functional capabilities, such as folder reorganizating, filing and retrieving, becomes a critical success factor in developing and implementing such an intelligent system.

2.1 The Logical Model

In [59,61,101], a folder organization is represented by a Rooted Direct Acyclic Graph (RDAG) where each node represents a folder and each directed link between two nodes represents a filing path folder relationship. A root is designated as the rooted node (also called the rooted folder) of the organization. The rooted folder has only outgoing links without any incoming link. A leaf folder is an end folder with only incoming links. An intermediate folder is a folder which possesses both incoming and outgoing links.

Each folder in a folder organization contains a collection of frame instances of various document types. Each folder is associated with a user defined filing criteria (predicate) which governs the filing decision for the folder. A frame instance from its parent folders can be deposited into this folder only if the frame instance satisfies this folder's filing criteria.

The rooted folder serves as a starting point connecting to the rest of the folders of the organization using links. This rooted folder contains a copy of every frame instance the user desires to keep in his/her folder organization. Its associated predicate is set to be "TRUE".

A filing path of a given folder in the organization is a path from the rooted folder to this folder where a particular frame instance is flowing through (making deposits at every stop along the path). The filing path of a given frame instance to be deposited into a folder from the rooted folder may not be unique. If a frame instance satisfies all the filing criteria along two different paths toward a common folder, the filing process should drop a copy of this frame instance at every stop of each respective paths except those common folders in which only a copy of the frame instance is deposited. In other words, if a folder has more than one parent folders, a given frame instance, which is deposited in any of these parent folders, can flow into this folder through its incoming links provided it qualifies the child folder's predicate. Once a copy of the frame instance is dropped into the child folder, the same frame instance coming from the other parent folders will not be re-deposited into the folder so only one copy of the frame instance is kept in the child folder.

On the other hand, it is also impossible to find a frame instance in a child folder without having a copy found in any of the parent folders. Therefore, if a document was found in a child folder, it is guaranteed that there is "at least" one filing path of the child folder where the frame instance flowed into the folder.

Consider again the folder organization as shown in Figure 3. We shall use it to explain the above folder organization model. Since the frame instance f1 satisfies the predicates P1, P2, P4 and P6, a copy of f1 can be found in each of the four associated folders F1, F2, F4 and F6.

The frame instance(s) flowing from the rooted folder along each link between folders in the folder organization simply indicates a filing path. For instance, the frame instance f1 is deposited into folder F6 via the filing path F1-F2-F6 or F1-F4-F6 because it satisfies $P1 \wedge P2 \wedge P6$ or $P1 \wedge P4 \wedge P6$ respectively. But this frame instance f1 does not go through the filing path F1-F3-F6 because f1 does not satisfy P3 (and therefore $P1 \wedge P3 \wedge P6$). The frame instance "f2" is deposited into folder F6 via the following filing paths: F1-F2-F6, F1-F3-F6 or F1-F5-F6 for it satisfies $P1 \wedge P2 \wedge P6$, $P1 \wedge P3 \wedge P6$ or $P1 \wedge P5 \wedge P6$. But it does not go through F1-F4-F6 for its does not satisfy $P1 \wedge P4 \wedge P6$.

Figure 5 depicts an example of a real world folder organization for an university administrator based on the U-ORG model. Under the rooted folder (folder NJIT), the administrator divides the folders into two branches, one for academic departments (folder Academic_Dept which is divided into folders named CIS, EE, etc.) and the other for university employees (folder Univ_Emp which is divided into folders Faculty, Staff, etc.). Under the CIS Department (folder CIS), the administrator categorizes the personnel records into two folders: the Staff folder and the Student folder. Furthermore, the administrator subdivides the student folder into three more folders: the B.S. folder, the M.S. folder, and the Ph.D folder depending on which program the student is enrolled. The folder Student may contain students who do not enroll in any of the given three programs, such as a student in a non-degree study program.

On the other branch of the folder organization, the user tracks all of the university employees' records. The personnel records are divided under two employee classes: the faculty class (folder Faculty) and the staff class (folder Staff). The faculty class is further divided into several other folders based on the hiring positions such as the Professor, Assoc_Professor and Asst_Professor folders. The staff class has folders such as Spec_Lecturer and TA (for special lecturers and teaching assistants, respectively). Finally, all employees whose ages are greater than 30 are gathered into a folder (called the folder Emp_of_Age_30+), which is created under the two employee classes, faculty and staff.

Each folder has a local predicate associated with it. For instance, the folder NJIT's predicate is school="njit"; the folder Academic_Dept's predicate is organization="academic_dept"; the folder Univ_Emp's predicate is role="univ_emp"; the folder CIS's predicate is dept="cis"; the folder EE's predicate is dept="ee", and so on.

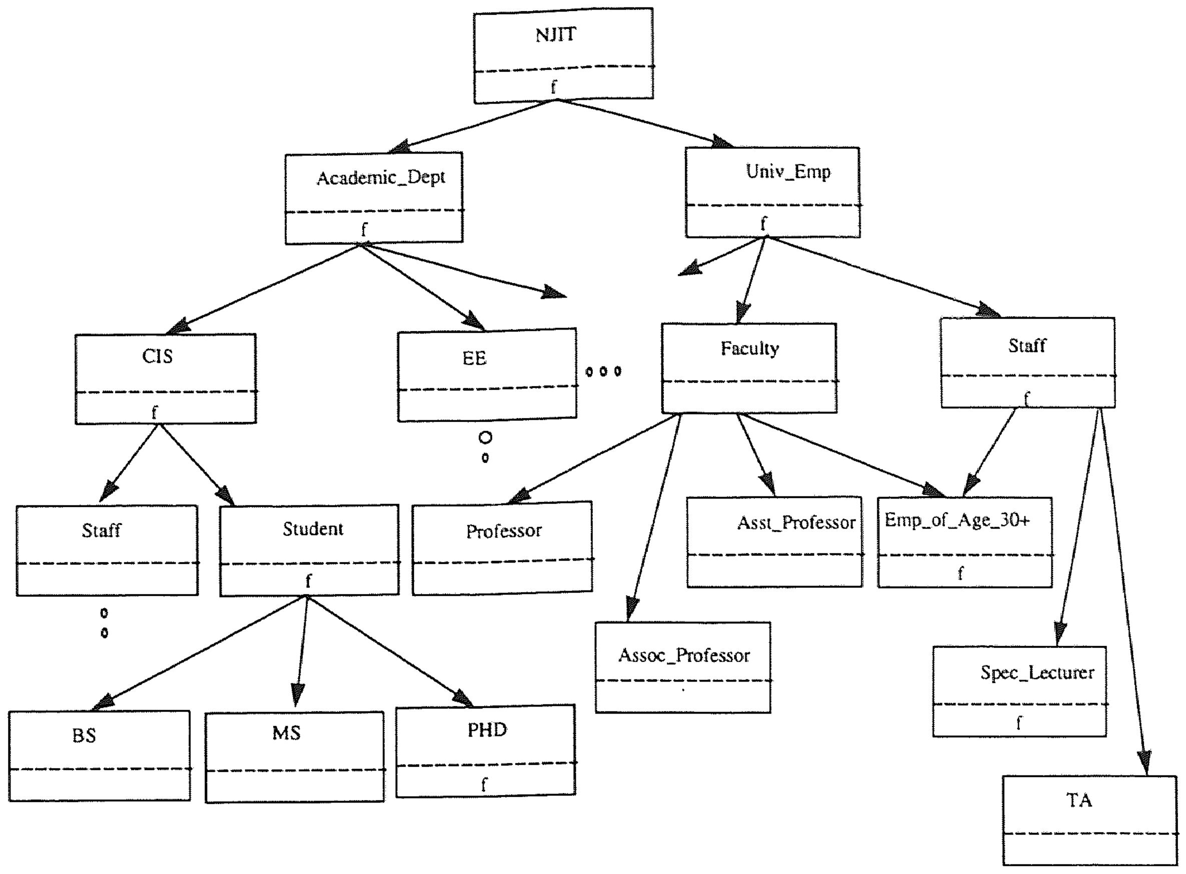


Figure 5 An University Folder Organization

Consider a frame instance f as shown in Figure 6. After filing, the given frame instance is deposited into the following nine folders: the NJIT, Academic_Dept, Univ_Emp, CIS, Student, PHD, Staff, Spec_Lecturer and Emp_of_Age_30+ folders. The information carried within the given frame instance must qualify the predicates of the nine associated folders. By observation, a copy of the given frame instance is deposited into the Emp_of_Age_30+ folder although it does not show in the Faculty folder which is one of the parents of the Emp_of_Age_30+ folder, but the Staff folder contains a copy of this frame instance. It is because this frame instance satisfies the predicate of the

Emp_of_Age_30+ folder and its filing path to the Emp_of_Age_30+ folder goes through the parent folder called Staff, rather than through the parent folder called Faculty.

However, if the frame instance f does not satisfy the predicate of the Staff folder, then it is impossible to find a copy of f in the Emp_of_Age_30+ folder even though f satisfies the Emp_of_Age_30+ folder's predicate. This is because none of the parent folders Faculty and Staff contains a copy of f.

The folder relationship as shown in the example is simple and straightforward. When the filing of a given folder is completed, the control moves to one of its child folders and continues the filing. Even if the folder has more than one child folders, filing on those child folders are independent from each other (i.e., filing result of one child folder does not affect the filing results of the other child folders of the same parent). On the other hand, if a child folder has more than one parent folder, the filing of a frame instance into this child folder is independent from the filing of the frame instance by the other parent folders of the child; that is, it follows its link down to the child folder for predicate evaluations, disregarding the existence of the other links. This folder organization is an User ORGanization or U-ORG for short. A user defines his/her folder organization in which every link simply indicates a possible filing path between the two connected folders.

Attribute	Value
school	njit
dept	cis
position	student
pgm	ph.d
employee class	staff
status	special lecturer
age	40
area of interest	database systems
organization	academic_dept
role	univ_emp
○ ○	

Figure 6 Frame Instance

2.2 Local Predicate and Global Predicate

Before we discuss the shortcomings of the U-ORG folder organization model, let us first review the "predicate".

An atomic predicate is defined as

<attribute> op <value>

where **"attribute"** is defined by the user and can be treated as a keyword. A predicate can be defined as an atomic statement, or a composite statement that comprises multiple atomic and/or composite statements, connected by logical operators such as "AND", "OR", and "NOT" to form a composite predicate. For example, both (dept=CIS) $\wedge$ (position=Faculty) and ((dept=CIS) $\wedge$ (position=Faculty)) $\vee$ (dept=EE) are composite predicates. These predicates consist of two and three atomic predicates, respectively. Attributes with the same or close semantic meaning can be grouped together and placed in a semantic network. For example, the two keywords "dept" and "division", which have the same semantic meaning, belong to the same keyword group. When an incoming frame instance does not have any information about to which department it belongs, but rather it includes the division information, then the division will be taken into consideration[2]. When a user inputs predicate specifications as part of constructing his/her folder organization, each attribute specified must also define a data type. For example, the data type of the attribute "age" is an integer, and the data type of the attribute "position" is a character string or an array of characters.

[2] The more challenging problem is how to define a set of inference rules to best serve situations in which an frame instance carries incomplete information for filing.

"op" is an arithmetic (e.g. "=", ">") or set (e.g. "IN", "BELONG TO") operation.

"value" gives data of the type for the attribute.

Every folder must have an atomic or a composite predicate which serves as its local filing criteria. A valid predicate (atomic or composition) can be derivable from the following production rules (a given grammar).

```
 <Predicate> --> <composition>
<composition> --> <composition> <OP> <composition> | (<composition>) <OP>
                  <composition> | <composition> <OP> (<composition>) |
                  (<composition>) <OP> (<composition>) | <atomic>
<OP> --> = AND | OR
<atomic> --> attribute <op> value
<op> --> = | > | < | . . .
```

where "Predicate" is the start symbol and terms without brackets are terminal symbols.

Given a folder organization in which each folder has a local predicate, the global predicate of each folder can be derived from ANDing the folders' local predicates of a filing path from the rooted folder through the folder; and ORing the global predicates of the folders for all the filing paths from the rooted folder to the folder. Each local predicate of a folder can be in the form of either atomic or composite [8]. Before filing a frame instance into a folder, the filing process simply needs to evaluate whether the frame instance from at least one of the parent folders of the folder qualifies the folder's local predicate and draws a depository decision. On the other hand, unlike the local predicate that represents a folder's filing criteria in the local context, the global predicate of a folder, represents this folder's overall filing criteria of the filing path in the context of the entire folder organization. Therefore, the global predicate of a folder governs its content in the organization. This also implies that a folder's global predicate must be a part of any of its descendant folders' global predicates.

To compose a folder's global predicate, only incoming links to this folder affect its composition. The following rules apply.

1. The global predicate of the rooted folder is equal to its given local predicate.

2. Assume that p is the local predicate of a folder F which has n parent folders, F1, F2, .., Fi, .., Fn ($n \geq 1$). Let Pi be the global predicate of the folder Fi. Then the global predicate of the folder F is $p \wedge [(\vee Pi)\ (i=1, .., n)]$.

If we have a folder organization which consists of four folders F1, F2, F3 and F4, where F1 is the parent folder of both F2 and F3; and F4 is the child folder of both F2 and F3. Let P1, P2, P3 and P4 be the four local predicates for F1, F2, F3 and F4, respectively. From the above composition rules, we know that the global predicate of folder F1 is P1; the global predicate of folder F2 is $P1 \wedge P2$; the global predicate of folder F3 is $P1 \wedge P3$; and the global predicate of folder F4 is $P4 \wedge ((P1 \wedge P2) \vee (P1 \wedge P3))$ which is equivalent to $P4 \wedge P1 \wedge (P2 \vee P3)$.

2.3 Predicate Evaluation Procedures

In the previous section, we discussed how a predicate is currently defined. It is simple, though it may result in the false drop problem [105]. A false drop is a filing result where a frame instance is deposited into a folder to which it does not belong. Since defining a sound and complete predicate is an on-going research topic [105], our architecture work should be independent from the development of predicate evolution. In the other words, as long as the predicate-based filing concept is adopted, our architecture should be able to support the associated predicate evaluation work disregarding the format of predicate specifications.

If accepting the concept of how predicates are defined, their evaluation processes are straightforward. We will have more discussion in the later of this section. But regardless, whether the predicate of a folder is atomic or composite, its evaluation against the information extracted from a given frame instance always generates a Boolean result: True or False. If the result is True, then a copy of the frame instance is deposited into this folder, and the filing process proceeds downward along with this folder's filing paths. Otherwise, the filing process halts at the folder and skips all the descendant folders of this folder. Repeat the same filing (predicate evaluation) procedure for another selected folder (dependent on the filing algorithm).

Before a predicate is used as a filing criteria, we need to validate the correctness of its syntax [106]. The semantic correctness checking, which understands and verifies the semantic meaning of a given predicate (for instance, whether this predicate is redundant or conflicts with another predicate in the organization) [106], is beyond the scope of this dissertation. Nevertheless, we will include some exploratory discussions in Chapter 12.

In addition to the validation of the predicate's syntax, it may be helpful if we can decompose a complicated composite predicate into simpler representations (i.e. a set of atomic predicates) for the subsequent evaluation processes. "Predicate parsing", "evaluation ordering", and "predicate verification" constitute an important part of the construction subsystem and are usually done during the construction phase of a folder organization. We elaborate each of these three tasks as follows:

(1) Predicate parsing:

Predicate parser examines whether a given predicate is derivable from the following production rules for validating the correctness of its syntax. For instance, by applying recursively the rules given in Section 2.2, the composite predicate (dept=CIS) AND ((pos=student) OR (yr IN (94, 95)) can be derived as follows:

```
<Predicate> --> <composition>
          --> (<composition>) <OP> (<composition>)
          --> (<atomic>) <OP> (<composition>)
          --> (attribute <op> value) <OP> (<composition>)
          --> (attribute = value) <OP> (<composition>)
          --> (attribute = value) AND (<composition>)
          --> (attribute = value) AND ((<composition>) <OP> (<composition>))
          --> (attribute = value) AND ((<atomic>) <OP> (<composition>))
```

--> (attribute = value) AND ((<atomic>) <OP> (<atomic>))

--> (attribute = value) AND ((attribute <op> value) <OP> (<atomic>))

--> (attribute = value) AND ((attribute = value) <OP> (<atomic>))

--> (attribute = value) AND ((attribute = value) OR (<atomic>))

--> (attribute = value) AND ((attribute = value) OR (attribute <op> value))

--> (attribute = value) AND ((attribute = value) OR (attribute IN value))

The derivation tree of the predicate is given in Figure 7, which is generated by the depth-first search.

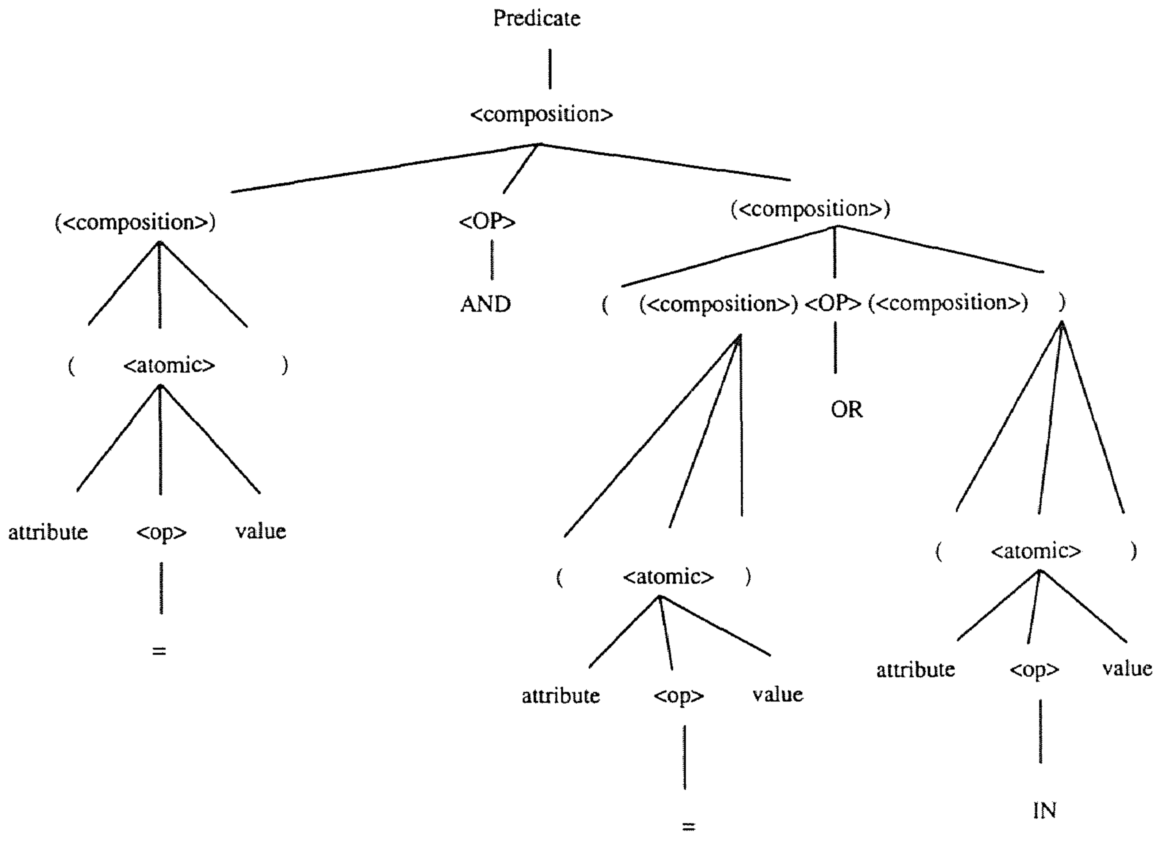


Figure 7 Part of Derivation Generated by Depth-First Search

*(2) **Evaluation ordering***:

The predicate parser examines whether a given predicate is syntactically correct. We are investigating here, a logical representation of the ordering of evaluation process for these atomic predicates of a given composite predicate. The ordering of evaluating atomic predicates has possible short-circuit effect for improved performance. There may exist different ways of organizing the atomic predicates; and some of these ways may give the ordering of evaluating atomic predicates an optimal performance.

The above example is decomposed into three atomic predicates: “dept=CIS”, “pos=student” and “yr IN (94, 95)” and is represented by the operational structure as shown in Figure 8:

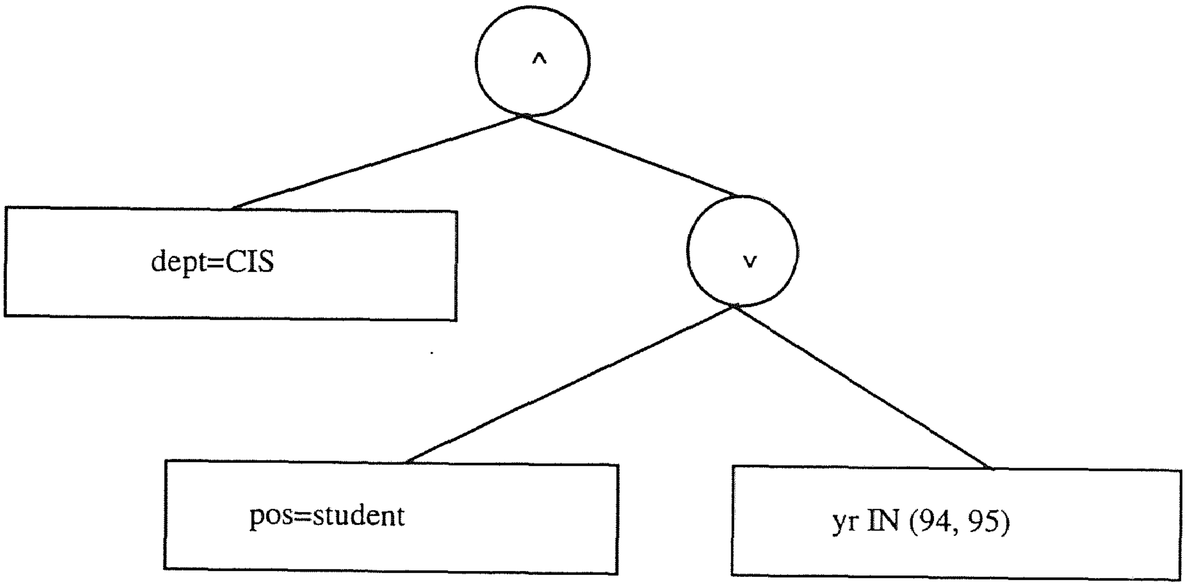


Figure 8 Logical Structure of Atomic Predicates

We develop the following algorithm to facilitate a general predicate evaluation process:

Algorithm:

```
Main
    Eval_order(root)      // start processing the rooted folder
End Main
Eval_order(node)
    switch(node)
        ∧: for each child node N
                Eval_order(N);
                if (return code="false")
                then exit with return code="false";
                end if
           end for
           return "true";

        ∨: for each child node N
                Eval_order(N);
                if (return code="true")
                then exit with return code="true";
                end if
           end for
```

```
            return "false";
        Predicate: evaluate the predicate & return either true or false depending
                on the verification result;
    End switch;
End Eval_order
```

The ordering for processing the three atomic predicates in the example is significant to the performance: That the leftmost predicate (dept=CIS) is evaluated first with a return "false", will cause the evaluation process to stop and return a "false" to the filing subsystem so that the filing subsystem will not put a copy of the frame instance into the current folder. Otherwise, the evaluation process proceeds to evaluate the other two atomic predicates and then evaluate the result with "AND" and "OR" operators for its completion if the current evaluation returns a "true" result. In contrast to this, the other approach evaluates first either of the predicates on the right hand side, and the evaluation process must continue disregarding the output of the previous evaluation. Therefore, from the performance perspective, we should evaluate the leftmost predicate first for this case.

(3) Predicate verification:

Predicate verification (evaluation) is a process which evaluates whether a given frame instance satisfies a predicate. It is done during the filing phase. The filing subsystem needs to first extract the necessary information from the given frame instance and compares them with the sorted atomic predicates of the target folder. The comparison is done one after the other or per atomic predicate basis (algorithm dependent) until a filing decision can be made.

For determining whether the information extracted from a given frame instance satisfies a given atomic predicate, we need to extract the "right" information from the frame instance for predicate verification. If the attribute of an atomic predicate does not appear at any part of the frame instance, then we may need to consult the thesaurus for possible keyword matching, which, in turn, provides us linkages between the given attribute and those relevant terms stored in the thesaurus. If any of those general terms in the thesaurus can be located in the frame instance, we may subsequently extract from the frame instance the associated value part of this term (which is considered as equivalent to the predicate's attribute). Verifying the value part should be straightforward depending on the operator and the data type. For instance, from the frame instance, if a person's age that we extracted out for verification is 50, verifying this information against the target predicate "age>30" should result in a true output. As we mentioned above, the ordering of evaluating atomic predicates is a dominant factor in achieving performance optimization.

The number of atomic predicates need to be evaluated before a filing decision may be made should be kept to a minimum. Therefore, when the system finishes evaluating each individual atomic predicate contained in such a set and a filing decision can be drawn after applying the logical operators (which link the atomic predicates) upon the results from those atomic predicate evaluations, then the evaluation procedure for the predicate is said completed. Otherwise, the evaluation process continues for another set of atomic predicates until a filing decision finally can be made. A generic structured predicate evaluation process is still under investigation which should not be locked into the way the predicate format is currently defined.

2.4 Problem Definition

The structure of a personal folder organization in the office environment plays a major role in supporting the capabilities of fast and correct filing and retrieving. For instance, to achieve the effectiveness of the fast and correct filing, we minimize, as many as possible, the number of predicates to be evaluated as they are part of the underlying folder organization. Evaluating a single predicate can be a time-consuming process. The process of evaluating whether a given frame instance qualifies a predicate involves the following two tasks: firstly, searching and extracting the right information from the given frame instance; this information may be obtained directly from the frame instance or through the use of a thesaurus. Secondly, determining the match of the extracted information against the local predicate associated with the target folder. Therefore, a simple specification of predicates for any folder organization is desired.

The existing organizational model U-ORG has some architectural constraints: it is difficult to model a complicated folder organization without creating the complexity of the specification of predicates, and consequently, it cannot meet the objective of fast filing.

A complicated folder organization can be an organization created with special user requirements, which may have logical relationships embedded in the folders. For instance, add a new folder into a given folder organization, which contains only the common documents from its selected parent folders. By adding a folder into the folder organization, the user is required to specify carefully the associated filing predicate for the new folder in order to ensure the correctness of the subsequent filing. The predicate specification of this new folder must take into account the predicates of all of the related ancestral folders. Modeling such a folder relationship within the existing framework requires the user to provide a condition-equivalent filing predicate for the folder. In doing so, predicate specification work becomes a nightmare for the user.

Assume that a new folder is created to contain all common frame instances from a set of selected folders. How do we present such a folder relationship between this new folder and its parent folders? Now, let us generalize it into a complicated case. Let us divide the desired parent folders of this newly created child folder into two groups, namely, group-A and group-B. How do we present a relationship so that this new child folder contains common frame instances found from every folder of the group-A, "and" from at least one folder of the group-B? It is difficult to model this kind of folder requirements with the existing folder relationship. The child folder requires a predicate which specifies some constraints for monitoring the distribution of frame instances from its parent folders. This creates a burden for users to construct such a predicate for meeting the requirements. In this dissertation, we shall investigate a structural way of simplifying the specification of the predicate.

Consider the example given in Figure 5, let create a new folder which contains frame instances regarding the "special lecturers" of "age>30" who are either enrolled in "MS" program in "CIS" or the "PhD" program in "CIS". Assume that this new folder has linked to each of the four respective parent folders. The required predicate for this new folder must take into consideration all of the following: (1) the status is "special lecturer"; (2) age is older than "30"; (3) attends either the doctorate program or master's program; (4) the employee class is either "faculty" or "staff"; (5) the position is "student"; (6) the department is "cis"; (7) the organization is "academic_dept"; (8) the role is "univ_emp"; and (9) the school is "njit".

For the U-ORG model, the user must specify correctly the following local predicate for such a folder:

(a) role	= "univ_emp"	AND
(b) employee class	= "staff"	AND
(c) age	> "30"	AND
(d) status	= "special lecturer"	AND
(e) organization	= "academic_dept"	AND
(f) dept	= "cis"	AND
(g) pos	= "student"	AND
(h) (pgm	= "ms"	OR
pgm	= "ph.d")	

In the hierarchy of an U-ORG folder organization, any non-rooted folder inherits all of its ancestral folders' predicates. Those ancestral folders may lie on the different filing paths of this non-rooted folder and these predicates must be included in the predicate specification of the folder. Therefore, the specification of predicate for a newly created folder in such an environment is complicated because it is easy to create redundancy or miss any part of the required predicates.

As shown in Figure 9, the predicate specification for the new folder is based on the configuration in which this folder is attached to the four related parent folders: MS folder, PHD folder, Emp_of_Age_30+ folder and Spec_Lecturer folder. Since a frame instance of this newly created child folder may come from any of the four parent folders, we must take into account the global predicates of these parent folders, when specifying an appropriate local predicate for the new folder. Therefore, the local predicate for the folder must include the atomic predicates of (e), (f), (g) and (h) to ensure that any frame instance arriving from the Emp_of_Age_30+ folder from the Spec_Lecturer folder into the new folder must be a CIS student, who enrolled in either the MS or PHD program. This is because frame instances which do not qualify some filing criteria on the "Academic_Dept" branch, could still arrive at the new folder from either of these two paths. Similarly, this predicate also needs to include the atomic predicates (a), (b), (c) and (d) to ensure that any frame instance containing information regarding a special lecturer whose age is above 30 can be deposited into the new folder via the MS folder or PHD folder. It is certain that a frame instance will not flow into the new folder from the Faculty folder, and therefore, the Faculty folder's predicate can be excluded from the predicate specification of the new folder.

If, for some reason, this new folder is connected to only MS folder and PHD folder, then the predicate for this new folder would be changed accordingly. The predicate specification under this configuration should be simpler in comparison with the previous case, since there are fewer possible filing paths from the rooted folder to this new folder.

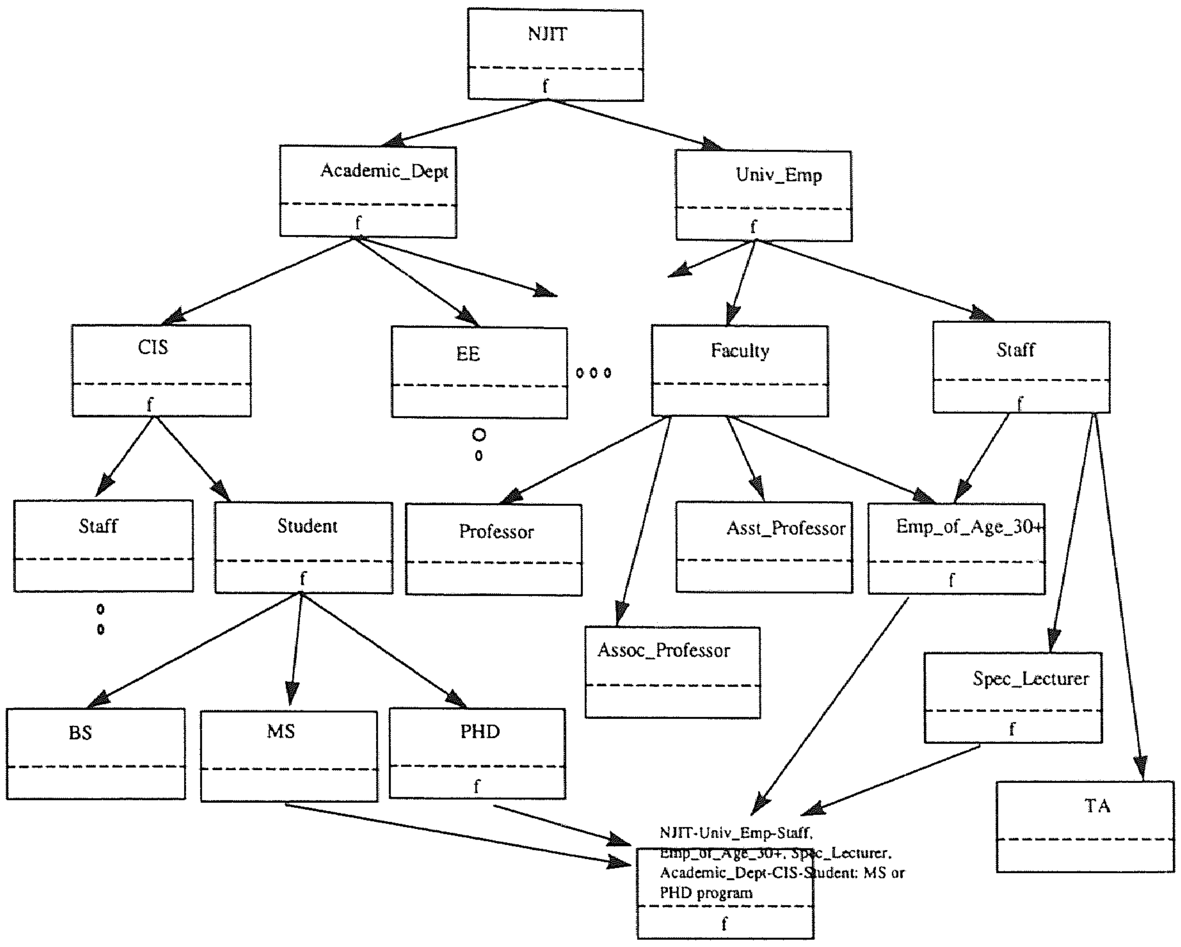
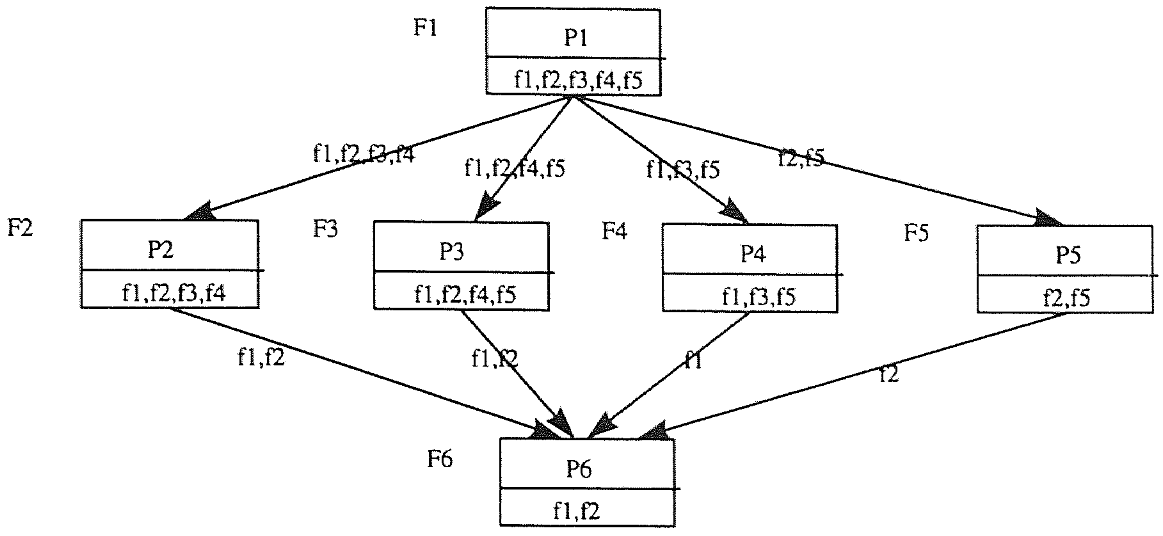


Figure 9 Creating a New Folder

A simplified model as shown in Figure 10 summarizes the above problem: a new folder (F6) is added to the existing folder organization where it accepts only the frame instances which are from both the folders F2 and F3; and must also be from one of the folders F4 and F5. As shown in the figure, Folder F6 contains a copy of "f1" and "f2". The frame instance "f1" satisfies the local predicates P1, P2, P3 and P4. The frame instance “f2” satisfies the predicates P1, P2, P3 and P5. The frame instance “f3” satisfies P1, P2 and P4, but not the required P3; the frame instance “f4” satisfies P1, P2 and P3, but neither P4 nor P5; and the frame instance “f5” satisfies P1, P3, P4 and P5, but not P2. The local predicate P6 of the folder F6 can be specified as follows: $P6 = G2 \wedge G3 \wedge (G4 \vee G5)$ where Gi represents the global predicate of the folder Fi.

Redundancy or duplication can arise if the predicate of the folder F6, P6, is specified in terms of Gi’s because each of the Gi contains P1, the local predicate of the rooted folder. Such redundant or duplicate parts should be removed from the composite predicate P6 to improve the performance efficiency. Very often we specify a very complicated predicate for a newly created folder in which, even if the user’s requirements remain unchanged, different configurations used for adding the same folder may result in different predicate specifications. This means that the predicate specification for a new folder depends on how this folder is linked to the rest of the folder organization.



P6 = G2 ^ G3 ^ (G4 v G5)

Fi:	folder
Pi:	local predicate
Gi:	global predicate
fi:	frame instance
⟶	filing path

Figure 10 Complexity of Predicate Specifications

CHAPTER 3

FOLDER ORGANIZATION: INTERNAL FOLDER ORGANIZATION (I-ORG)

In this chapter, we investigate an approach to enhancing the existing folder organization model. The primary motivation for designing an enhanced folder organization model is to help the user specify the local predicates for folders in a simpler form. This local predicate specification, which is part of constructing a folder organization, must have the following characteristics:

(1) eases user's definition input,

(2) greatly reduces the processing time for document distributions and retrievals, and

(3) correctly governs the deposition decision of the incoming document objects.

In this dissertation, we propose the following two types of links to model the folder relationships defined on an organization: the subfolder link and the virtual-folder link. A subfolder link captures the "and" relationship among links between the child folder and its parent folders. On the other hand, the virtual-folder link, which is the same as the only link defined in the old U-ORG model, captures the "or" relationship among links between the child folder and its parent folders. Our focus here is on the child folder and not the parent folder because we are employing the top-down filing approach which visits a parent folder first before the child folder. The relationship between a parent folder and any of its child folder are twofold: firstly, it shows a possible filing path; and secondly, before a deposition is made into this child folder via any filing path, all of its parent folders need to be examined to ensure that the relational constraint is maintained. The relational constraint specifies that certain parent folders must contain the document before this document can be filed against their common child folder. The relational constraint should be checked prior to filing a document to a folder takes place.

With this new representation, it eases the folder creation task by reducing the complexity of predicate specification, and achieves superior performance in filing and reorganization over the existing model. It is because the two types of links not only represent complicated logical folder relationships, but also introduce a new relational constraint which, in turn, facilitates the filing efficiency (checking relational constraint via lookup flags and/or bit patterns check is much simpler than evaluating a predicate which requires finding and comparing information). A folder organization represented in this model, is called an Internal ORGanization or I-ORG in short. For the concern of the expressive simplicity, the ideal system should allow the user to create his/her personal folder organization with fewer restrictions. In other words, the system should present to the user a very simple and comprehensive organizational structure. This structure allows the user to configure the folders and specify predicates freely but keeps all complexity transparent to the user. If the U-ORG folder organization is the model considered for that purpose and being implemented in the user interface layer, then the I-ORG should be considered as the underlying support implementation for its operational efficiency.

Meaning that once the creation of a folder organization is done, the subsequent operation and management activities should be transparent to the user and should not be based on this external representation due to the performance considerations.

With the proposal to separate U-ORG from the I-ORG representation, the interoperability between the two models becomes a big issue. For instance, as we know that every folder in the user-created folder organization or U-ORG has its predicate defined, and the predicates of the folder organization, collectively form a predicate hierarchy. How a given U-ORG predicate structure could be simplified and transforms into an equivalent I-ORG based structure becomes a challenge. We will investigate the issue and present some preliminary ideas in Chapter 10.

3.1 Descriptive Definition

In this section, we give a semantic description of the folder relationships, and then follow by a formal definition:

A. Subfolder relationship (connected by a subfolder link called AND-link): Every frame instance of a child folder also belongs to its parent folder.

B. Virtual-folder relationship (connected by a virtual-folder link called OR-link): A virtual-folder relationship between two folders simply means that the frame instances in the child folder may or may not come from its parent folder. This virtual-folder relationship is equivalent to the filing path relationship of the U-ORG model.

Given two folders with the virtual-folder relationship (connected them by an OR-link), where does a frame instance in the child folder come from if it does not come from the parent folder. Apparently, it must come from the other parent folders. If we accept this assertion, then how do we define the virtual-folder relationship formally? It does not make any sense if we limit the scope of viewing a virtual-folder relationship only between the two related folders interconnected by an OR-link. Instead, we view it in a broader scope, all of the virtual-folder relationships between a given folder and its parent folders as a whole in the filing environment.

In other words, if there are multiple parent folders connecting to a common child folder via virtual-folder links, it simply means that every frame instance found in the child folder must come from at least one of its parent folders. Therefore, if a folder has only one incoming virtual-folder link, then this link has no difference from a subfolder link. Consequently, we may impose a design rule for the folder construction phase, namely the number of virtual-folder relationships of a child folder in the folder organization must not be equal to one; otherwise, the system should convert automatically the link of virtual-folder relationship into a subfolder link.

3.2 Formal Definition

Let C be a child folder and P be a parent folder of C. Both folders C and P contain their respective frame instance objects. Let f_P be a mapping function which maps objects in folder C into 0 or 1, depending on whether the objects are contained in the folder P (i.e., f_P: C $\rightarrow$ {0, 1}).

The object mapping function f_P is defined as follows: For any $x \in C$

$$f_P(x) = 1, \text{ if } x \in P$$
$$= 0, \text{ otherwise}$$

To complete this formal definition, we also need to cover the following two situations:

(1) if the domain of the function f_P (i.e., the object set C) is an empty set, and

(2) if none of the objects (not an empty object) in C can be mapped into objects in P.

The first situation states that the child folder does not contain any frame instances, when the folder organization was first constructed or simply because none of the filed frame instances can reach C and the second situation is that none of the frame instances in P qualifies C's local predicate.

We therefore extend the folders C and P to be sets of objects including the empty object $\varnothing$. By default, the empty object $\varnothing$ belongs to both of them. Therefore $f_P(\varnothing) = 1$, since $\varnothing \in C$ and $\varnothing \in P$; or the function f_P maps the "empty" object in C into the "empty" object in P.

Now, we can use function f_P to define the relationships between two folders C and P as follows:

A. Subfolder relationship: (For any $x \in C$) [$f_P(x) = 1$].

B. Virtual-folder relationship: (There exists an object $x \in C$, possibly an empty object) [$f_P(x) = 1$].

The above definition of folder relationships covers all kinds of organizational scenarios. For instance, given a child folder C which has two parent folders S1 and S2 connected by subfolder relationship and two other parent folders V1 and V2 connected by virtual-folder relationship. Assume that both S1 and S2 contain a frame instance f1; Vi contains a frame instance f1, and V2 contains a frame instance f2. Any frame instance found in the child folder C must come from both S1 and S2, and from either V1 or V2. Therefore, each of the four links appeared in this part of the folder organization satisfies the above definition:

(1) S1-C: $\forall f_i \in C, f_{S1}(f_1)=1$, since $f_1 \in S1$.

(2) S2-C: $\forall f_i \in C, f_{S2}(f_1)=1$, since $f_1 \in S2$.

(3) V1-C: $\exists f_1 \in C \ni f_{V1}(f_1)=1$, since $f_1 \in V1$.

(4) V2-C: for $\varnothing \in C \ni f_{V2}(\varnothing)=1$, since $\varnothing \in V2$.

3.3 Predicate Composition

Given the definitions of folder relationships, how do we compose the global predicate of a folder which has both subfolder and virtual-folder links? For instance, given a folder with local predicate p, the folder has n parent folders, of which the first k parent folders are subfolder parents and the other n-k folders are virtual-folder parents ($n-k \neq 1$). Let Pi be the global predicate of the parent folder Fi. Then the global predicate of this folder is equal to $p\wedge [(\wedge Pi)\ (i=1,..k)] \wedge [(\vee Pi)(i=k+1,..n)]$.

In Figure 11, we revisit the previous example (as shown in Figure 10) by representing the folder organization with our new model. As shown in the figure, the folder organization contains six folders and a total of twenty frame instance copies. Note that the specification of the predicate for F6 is much simpler (P6=true) in contrast to the complicated predicate specification in the previous example shown in Figure 10. This is because the user requirements for creating the folder F6 is embedded in the four links that connecting the four parent folders F2, F3, F4 and F5 to their common child folder F6. In other words, the two types of links are used to replace some complicated predicates without losing the desired folder relationships (but they need to be differentiated when composing the global predicates for folders). The filing against this organization needs to take into account the relational constraint. To deposit a copy of a frame instance into the folder F6, this frame instance must appear in both folder F2 and folder F3. In addition, the frame instance must also appear in either folder F4 or folder F5. For example, the frame instance f1 is filed into the folder F6 because it meets the above constraint while the frame instance f3 is not filed into folder F6 because the folder F2 does not contain a copy of f3.

3.4 Example of Creating a New Folder with I-ORG Representation

In the example as shown in Figure 9, it is difficult to specify the local predicates for a new folder. The corresponding example where the new folder is constructed under the I-ORG model is given in Figure 12. The local predicate specification for this new folder is true and the original predicate can be represented by its folder relationships.

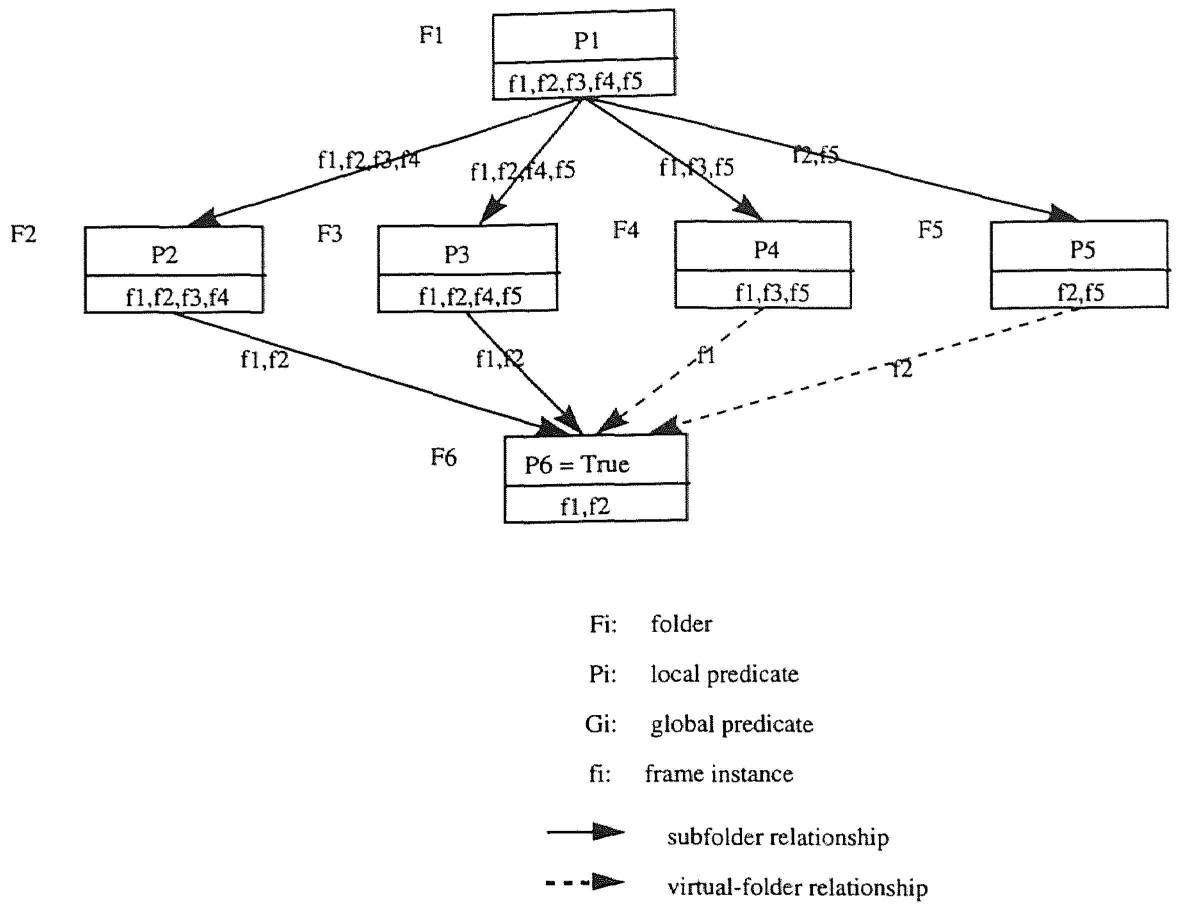


Figure 11 Folder Organization Represented by the New Model

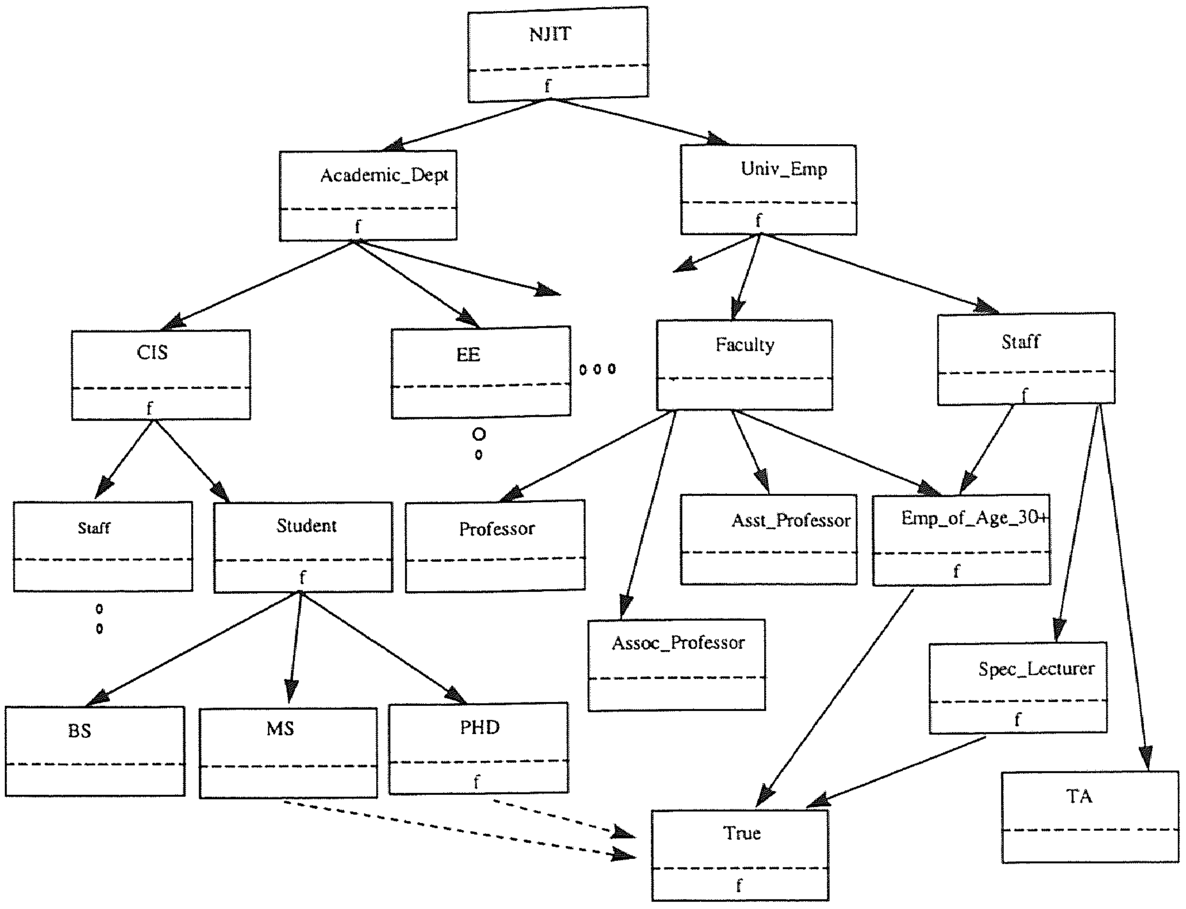


Figure 12 Creating a New Folder with I-ORG Representation

CHAPTER 4

SEQUENTIAL FILING MODEL

The design of the filing under the new organization model should capture some architectural attributes of the model in order to gain performance advantages. For instance, the filing model should perform the predicate evaluations as few as possible, because the predicate evaluation process is considered as a time-consuming task and can be avoided if the check of relational constraint fails.

The input of the filing process consists of (1) a given frame instance fi to be filed, (2) the target folder Fc and its associated local predicate, (3) the folder relationships between the folder Fc and its parent folders (forming the base of the relational constraints), and (4) the filing statuses and results of fi in those parent folders. The output of the filing determines whether a copy of fi is deposited into the folder Fc.

The third and fourth input items, namely the folder relationships between folder Fc and its parent folders, and the filing status and results of these parent folders provide the filing process with the information to (1) check whether the frame instance fi qualifies the relational constraint, and (2) determine the filing time for filing the frame instance fi against the folder Fc. At the folder Fc, once the frame instance fi passes the relational constraint check, the predicate evaluation process carries out the following two tasks: (1) searching and extracting the right information from the given frame instance fi, and (2) examining the extracted information against the local predicate of the folder Fc. Since the new organization model helps reduce the complexity of predicate specifications, performance saving in filing frame instances may be significant because now we should have fewer atomic predicates to be evaluated. For predicate evaluation, we need only to extract the minimum relevant information from the frame instance which is required to be compared against the local predicate. This implies that the simpler the predicate is specified, the less information is required to be extracted from the frame instance. Therefore, the volume of information to be extracted is really predicate dependent. In brief, since the specification of a predicate for a folder may be simpler by representing a good part of its predicate with links, and the time required for checking the relational constraints becomes insignificant in contrast to evaluating a complicated predicate, a filing model can be designed to take this architectural advantage into considerations.

In this chapter, we propose a basic filing approach designed for the new I-ORG folder organization - the sequential filing model which performs the filing in a sequential manner.

Sequential filing performs a filing task by traversing in a sequential order through the entire folder organization. Starting from the rooted folder and following the links that interconnecting folders, the filing system visits folders one after the another to determine if the filing time of a given folder has been reached (thus sequentially). This ordering of traversal allows to synchronize the filing time between a given child folder and its parent folders. Because of this synchronization requirements, the ordering of the sequential filing approach becomes important. The synchronization rule requires the filing to be performed at all parent folders before their common child folder can be filed. This ordering requirement can be represented by a sort result as produced by the topological sorting algorithm, which in turn, utilizes the well-known Depth-First Search (DFS) algorithm to support the sorting task. Topological sorting on a RDAG is the reordering of its nodes (folders) along a horizontal line so that all directed links go from left to right indicating the filing order. Therefore, we employ the topological sorting algorithm in our sequential filing model to help construct a list of sorted folders which gives the desired filing order.

"Complete-Parent-First" algorithm takes the sequential approach to distribute frame instances into the I-ORG folder organization. The algorithm recursively traverses the entire folder organization and determines which folders should get a copy of the frame instance. The algorithm starts the filing process from the rooted folder, for each stop at a folder, the algorithm returns to the folder's immediate parent folder if any of the parent folder has not yet completed the filing. Otherwise, the algorithm performs the filing on the folder and continues the process on its child folders. Filing a frame instance into a folder, the algorithm first checks: (i) whether the frame instance has not already existed in the folder; (ii) whether the folder relational constraint is satisfied (to do this, the system needs to know which parent folders have link connections to this folder and what are the link types); and (iii) whether the frame instance qualifies this folder's local predicate. If either condition (i) or condition (ii) is not met, the filing process does not need to proceed to (iii) and consequently, the frame instance is not deposited into the folder. Therefore, (i) and (ii) can be considered as the pre-conditions before the condition (iii) is examined. However, if both conditions (i) and (ii) are met, before the deposition decision (whether a copy of the frame instance is deposited into the folder) is made, the filing process needs to ensure the information contained in the frame instance satisfies the folder's local predicate,

Figure 13 gives the process transition diagram of the "Complete-Parent-First" filing algorithm which consists of three states:

(1) "Searching" - discovers the next candidate node to be filed,

(2) "Synchronizing" - checks all parent folders' filing status, and

(3) "Filing" - checks the relational constraint and evaluates the local predicate if needed.

The filing process terminates if the "Searching" of the next candidate folder to be filed detects that all of the folders have already done their filing (all folders' associated flags are raised to indicate their "filing done" status). Otherwise, the process identifies the next candidate folder and enters into the "Synchronizing" state which in turn, determines if the folder is ready for filing (i.e., checking the filing status of all of its parent folders). If the folder is not ready for filing, the process returns to the "Searching" and continues looking for another candidate folder. On the other hand, if the folder is ready for filing, the system performs the "Filing" against the folder (i.e., checking relational constraint and comparing the content of frame instance with the folder's local predicate), raises the filing flag for the folder, then returns to the initial "Searching" state and repeats the same process cycle.

Note that the "Searching" follows the DFS algorithm discovering procedure and find the folder to be visited next. Once the folder is identified and returned from the "Searching", it is only considered as a candidate folder for filing. The "Synchronizing" subsequently checks this folder's readiness for filing by examining its parent folders' filing status. The folder enters into the "Filing" state if all of its parent folders complete the filing. Otherwise, the process returns to the "Searching" and proceeds (again, follows the DFS discovering steps) to another folder. The local predicate of the folder is evaluated against the information extracted from the frame instance as the folder enters in the "Filing" state. When the predicate evaluation is done, the filing of the folder is completed and the associated filing flag of the folder is raised disregarding the filing result, the process returns to the "Searching" state, and the filing continues on the folders that are not yet visited or filed. Each folder should be visited at least once. The number of times for visiting at a folder depends on how this folder's relationship is defined.

The folder organization shown in Figure 14 contains twelve folders. Assume that, for each folder, the traversing order of its outgoing links goes from left to right. The number shown in the folder node indicates the filing order and the number shown on each link indicates the link traversing order. Links accompanied by more than one number simply means those links have been traversed more than once. From this example, we know that the rooted folder is the first folder to be filed, its left child folder is the second folder to be filed, its right child folder is the fifth folder to be filed, and so forth.

The above filing order is topologically sorted. If we change the link traversing order from right to left instead, the revised filing order matches exactly the order as shown in Figure 15(a). In the figure, the number contained in each folder node indicates the filing order of that folder. The pair of numbers for each folder represents that folder's discovery time and finishing time in accordance to the Depth-First Search algorithm [18].

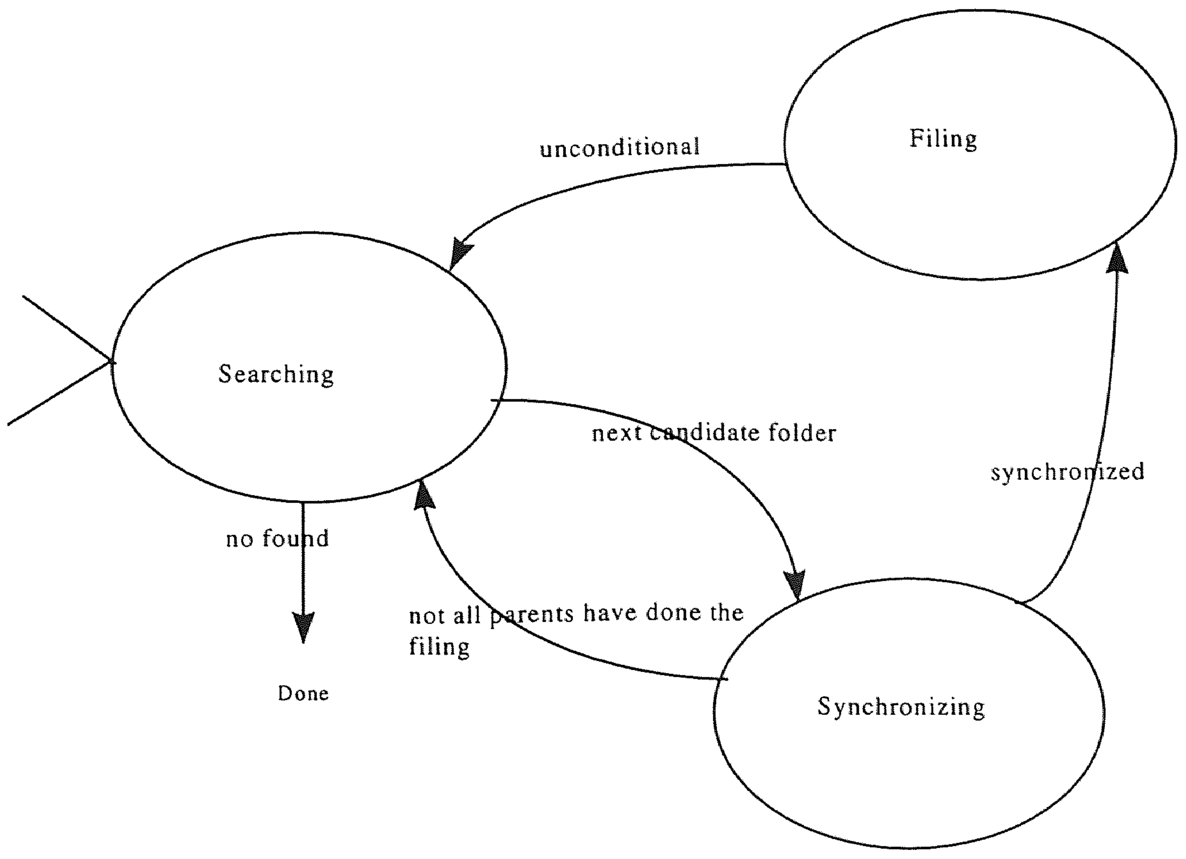
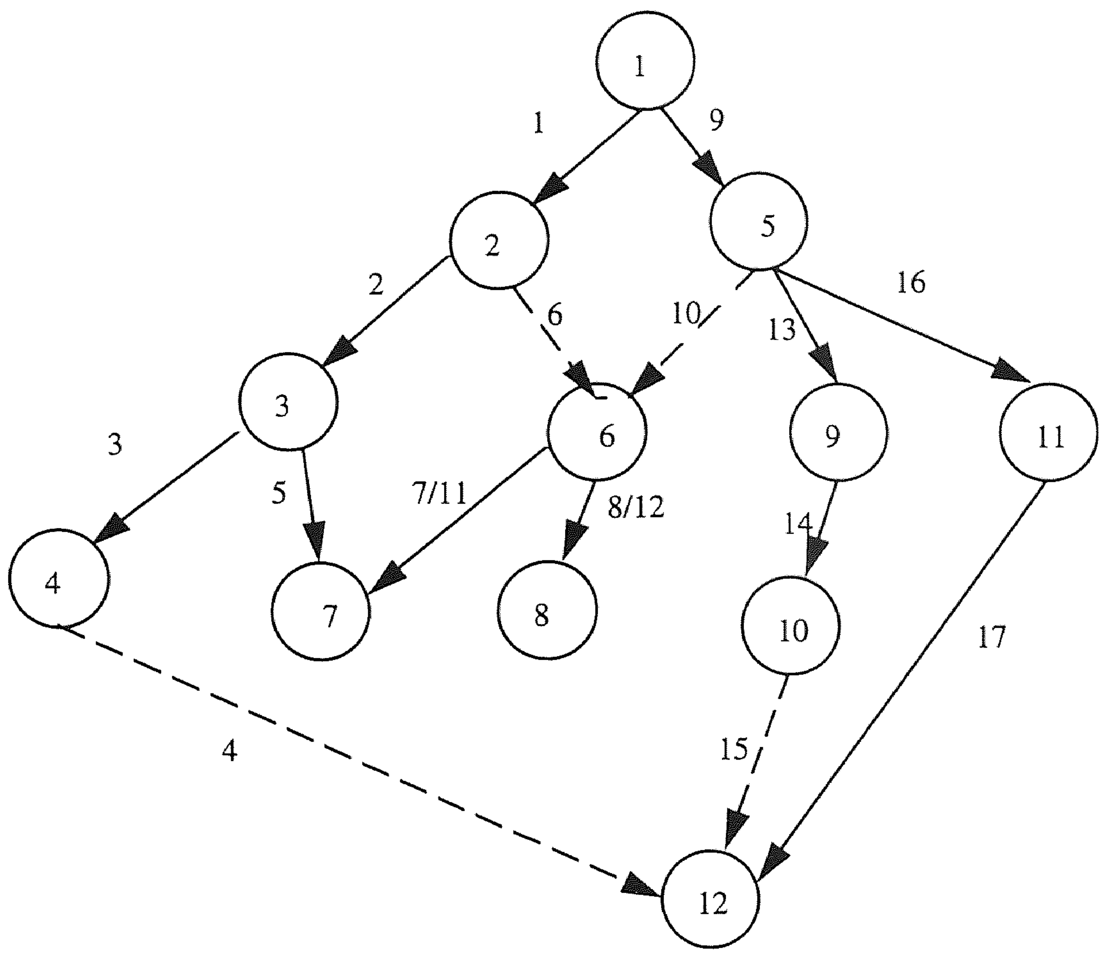


Figure 13 Sequential Filing

Discovery and finishing time indicate different states during the execution of the Depth-First Search algorithm. On a directed graph, each node migrates to the discovery state from its initial state when it is discovered the first time in the search process. This state change is timed as the node's discovery time. Similarly, as a node's adjacency list (contains all nodes which are adjacent to the current node) has been examined completely in the search process, the node moves from the discovery state into the finished state, and this move is timed as the node's finishing time. This technique guarantees that each node ends up in exactly one depth-first tree, so that these trees are disjoint.

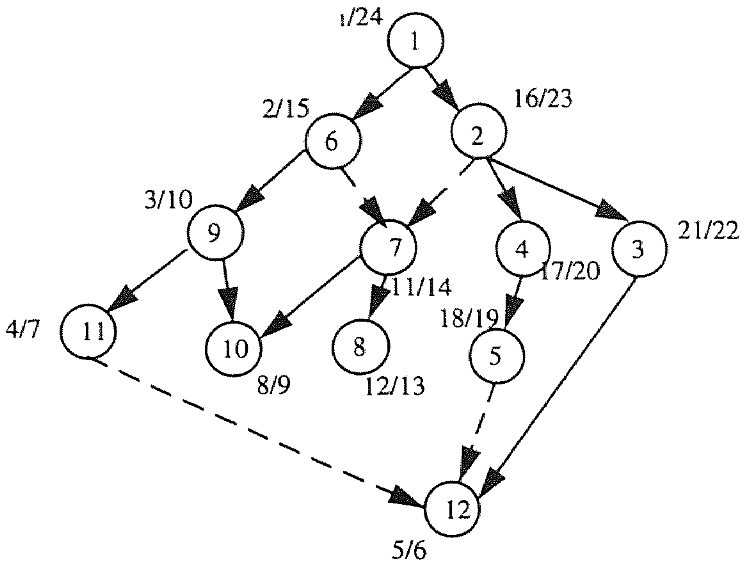


● number in folder: indicates filing order

● number along link: indicates traverse order

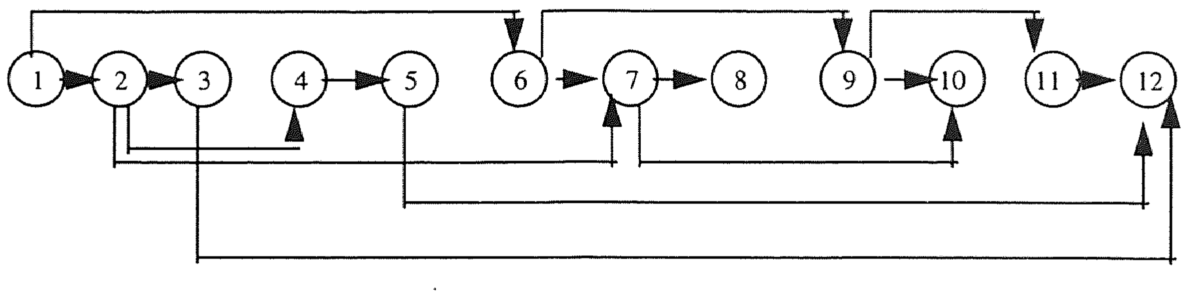
Figure 14 Complete-Parent-First: Example

As soon as a folder is finished, it is inserted into an ordered filing list as shown in Figure 15(b) where each directed link indicates the required filing precedence between the two folders which interconnected by the link. From the process procedure, it is easy to see that the time a folder is finished affects the precedence in which the folder is placed in the list, and, therefore, the earlier the folder is filed. Also note that the system needs not to wait until the preparation (sorting) of such a filing list is completed (topologically sorted) before it starts the actual filing process. The two tasks can be integrated and performed in the proper order.



x/y: discovery time/finishing time (DFS)

(a)



(b)

Figure 15 Topologically Sorted (Filing Order)

The following presents the "Complete-Parent-First" filing algorithm:

Algorithm:

```
// Define filing_flag=1 if the folder finishes the filing; filing_flag=0 otherwise.
// This main routine starts the filing process from the rooted folder and proceeds through
// the root's descendant folders by calling Distribute routine.

Complete_Parent_First (G, fi)
    set filing_flag(folder_ID) = 0 // initialize filing flags for all folders
    Filing(root, fi)
    set filing_flag(root) = 1
    Distribute(root)
end Complete_Parent_First

// This routine traverses the folder organization for filing by calling itself recursively.
// The routine stops and returns at the folder if any of its parent folders has not yet
// finished the filing.

Distribute (F)
    for each child folder Fc of F
        Check_Parent(Fc)
        if ret_val = 1
```

```
            then   Filing(Fc, fi)   // complete filing on all parent folders
                   set filing_flag(Fc) = 1
                   if Fc has outgoing links
                   then   Distribute(Fc)
                   else   return
            else   return
end Distribute
```

// The main body of filing is given as follows. The three conditions must be met before a
// copy of the frame instance is deposited into the folder.

```
Filing (F, fi)   // <F1> = { F' | F' has an outgoing subfolder link pointing to F }
                 // <F2> = { F" | F" has an outgoing virtual-folder link pointing to F }
      if (fi is in F)   // performs object ID checks
      then   return   // duplicate frame instance
      else   if (fi ∈ every F' in <F1>) and (fi ∈ any F" in <F2>)
             then   if (local predicate of F evaluates to true)
                    then   put fi in F        // local predicate is met
                    else   return
             else   return
end Filing
```

// Given a folder, this routine checks if all its parent folders have already done the filing.

```
Check_Parent (F)
        if [filing_flag(F') = 1 for every F' in <F1> and every F' in <F2>]
        then    return 1
        else    return 0
end Check_Parent
```

The above algorithm combines the topological sorting and the subsequent filing action together. An alternative approach is to finish filing order discovery first and then based on that order to complete the filing one folder after the other. Their performance should be the same because they both carry out the same tasks but only via different procedural orders. The following performance model is based on the alternative approach.

Given a folder organization where N represents the number of nodes (folders) and L represents the number of links (folder relations among the nodes). Disregarding the predicate evaluation time, the filing algorithm takes $\theta(N+L)$ time to determine the filing order (contributes to the "discovery" part of the filing performance). This is because topologically sorting a given folder organization relies on the DFS searching technique to determine the order of discovery for each folder of the folder organization. And it takes $\theta(N+L)$ time for DFS to achieve that purpose [18]. Moreover, accompanying each DFS discovery step, the sorting algorithm subsequently takes $O(1)$ time to insert each of the finished (state after discovered) $|N|$ nodes onto the front of the sorted link list, which forms a desirable filing order for the given folder organization.

CHAPTER 5

PERFORMANCE ASSESSMENT FOR SEQUENTIAL FILING

Based on the filing algorithms proposed initially for the U-ORG and I-ORG respectively, a simple model is presented for comparing the performance of filing frame instances using these two organizations. Namely, we are trying to model the computing cost of filing a frame instance into a particular folder of two different representations, the U-ORG representation and the I-ORG representation.

Let fi be a frame instance to be filed into a folder X in a given folder organization. Given the I-ORG representation shown in Figure 16, suppose that this simplified folder organization has n virtual-folder links and m subfolder links interconnecting X with its parent folders V1, V2, ..., Vn (parent folders of virtual-folder links), and S1, S2, ..., Sm (parent folders of subfolder links). In the U-ORG representation, suppose that the folder organization contains only n links from each Vi ($1 \leq i \leq n$) to X. The local predicates of X are different in these two representations. For the sake of simplicity and without loss of generality, assume that the local predicate of X is "TRUE" in the I-ORG representation and is $\wedge[G(Si), 1 \leq i \leq m]$ in the U-ORG representation.

It is assumed that the system has already performed the filing process on all the Vi's and Sj's folders. The distribution of frame instances among these parent folders of X significantly affects the subsequent filing performance in the U-ORG representation, but it has no effects in the I-ORG representation. For instance, in the U-ORG representation, the best case situation is that none of X's parent folders contains a fi before filing fi into X. In this case, the filing system does not need to do anything. On the other hand, the worst case situation is that each of the n parent folders of the folder X has a copy of fi. There are two scenarios of this situation: The first scenario is to present the best case of filing fi into X in the U-ORG, and the second scenario presents the worst case.

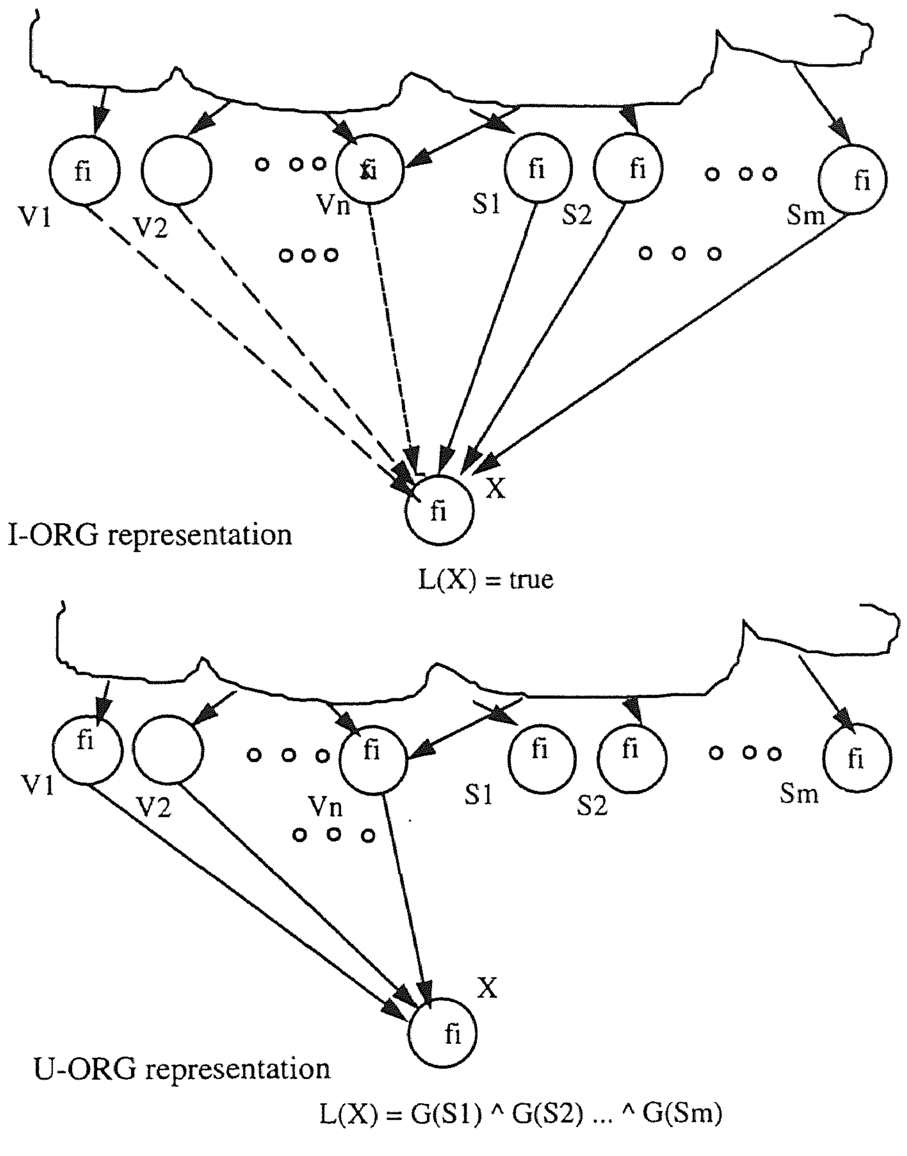


Figure 16 Filing Model

Assume that the global predicate of a folder Si contains an average of i atomic predicates. Let t1 be the average time for evaluating an atomic predicate. Let t2 be the average time for processing setting/checking flags or examining the existence of a frame instance in a folder (This operation is required as part of the relational constraint checkup; it only requires a quick ID lookup). Assume that t1 >> t2. t2 becomes insignificant in comparison with t1 because it may be just some straight operations which could be taken place within the computer registers, rather than creating a lot of I/O's on the other end.

Scenario 1 (the best case situation):

None of the parent folders with virtual-folder relationship, Vi ($1 \leq i \leq n$) contains the frame instance fi. Since the parent folders of X do not contain fi, the U-ORG filing algorithm stops at each Vi without considering further the filing of fi into any of its descendants. Therefore, the filing time for filing fi into the folder X is zero.

Scenario 2 (the worst case situation):

Let fi be a copy found in every Vi ($1 \leq i \leq n$). The filing time for filing fi into the folder X is in the range of [$(t2 * n + i * m * t1), (t2 + i * m * t1) * n$]. After filing fi at each Vi, the algorithm must proceed to filing of fi into X. However, in [99,100], the filing algorithm indicates that the existence of fi in the folder X should be first checked before any predicate evaluation takes place. Therefore, this worst case has two scenarios: (1) fi is deposited in X (during the first run), or (2) fi is not deposited in X. The respective performance of these two scenarios gives the above performance range between (t2 * n + i * m * t1) and (t2 + i * m * t1) * n. Since we assume that the global predicate of Si (G(Si)) contains an average of i atomic predicates and because G(Si) is included as part of L(X), the local predicate of the folder X in the U-ORG, there exists a total of i * m atomic predicates (with possible overlaps) in L(X) to be evaluated.

For the first scenario, the frame instance fi is deposited into the folder X at the very first run, which is also the only run. During the run the X's predicate is evaluated. The remaining n-1 filings only need to check the existence of fi. For the second scenario, although fi is not deposited in the folder X during the first run (i.e., fi from V1 does not qualify the predicate of X), we still need to check the existence of fi at X and evaluate X's predicate for the rest of the runs (fi from V2, V3, etc.) to complete the filing process for X.

The corresponding filing performance with the I-ORG representation is predictable!

No matter how fi is distributed in the parent folders of X, the algorithm requires 2 * t2 to complete the filing of fi into X. Since we assume that the system has already performed the filing process on all the Vi's and Sj's folders, the filing on X is completed in the last visit at X from a parent folder of X. In this visit, the system checks the existence of fi at X which requires t2, and examines whether the relational constraint of X is satisfied which also consumes t2 time. Note that the I-ORG based filing does not contain any t1 term because the model specifies "TRUE" as the local predicate for X. In addition, t2 can be ignored in comparison with the time required for predicate evaluation, and therefore, the

U-ORG's superior performance under the best case scenario is easily offset by the worst case scenario and any other cases.

CHAPTER 6

PARALLEL FILING MODEL

Unlike the sequential filing model where a central process synchronizes and controls all the activities, the parallel filing model requires the agent of each folder in the organization be responsible for its own filing process for the folder. Basically each agent is a self-contained autonomous process which is activated by the filing function. Agents communicate each other and exchange information about (1) knowing the filing status of their parent folders, and (2) letting their child folders know their filing status and/or filing result. The 'filing status' of a folder indicates whether the folder has already done the filing. A folder is considered to be done in filing if its representative filing agent completes the filing process on the folder and changes its associated filing flag. The 'filing result' of a folder indicates whether a copy of a frame instance is deposited into this folder after filing is done for the folder. In other words, the parallel filing model utilizes both structural (static) and real-time (dynamic) information to speed up its filing process. The static information refers to knowledge of the fixed, organizational structure, which comprises all the user-defined predicates and folder relationships. Dynamic information, on the other hand, is the information that may be updated and subsequently affects the down-stream processes during the filing course.

As discussed in the previous section, the sequential filing model has performance overhead since a folder may need to be visited several times before a frame instance can be filed into it. For each visiting at the folder, the same routines (e.g., initial checkup and/or house cleanup tasks) need to be performed repeatedly. The parallel filing model reduces the time spent on these tasks by letting each individual agent determine the readiness of the filing for its representative folder so that the same amount of tasks are being carried out in a parallel sense.

The parallel filing model may inherit the essential synchronization rule from the sequential filing model to ensure the timing of filing any frame instance into a folder. This restricts the parallel filing model to file a given frame instance into the parent folders first before filing it into the child folder. However, the parallel model significantly reduces the number of visits at folders as presented in the sequential filing model.

In order to accomplish the parallelism of processing but not violating the filing precedence restriction, the agent-based concept is employed to handle the frame instance distribution process. Each folder in the organization is represented by a filing agent which is basically a self-contained autonomous process in the system. Filing agents are activated by either of the following request triggers initiated by the user, **"filing"** or **"reorganization"**. The activated filing agents remain active until the frame instance distribution is done for the entire folder organization (filing request), or the system completes the necessary changes and finishes the frame instance redistribution for the entire folder organization (reorganization request). Here we focus our further discussion on the "filing" and include "reorganization" for our future research.

Since each agent performs the filing functions (such as determining filing time, extracting information from frame instance, evaluating predicate, updating filing status, notifying filing result to others, etc.) at its own disposal, it needs to rely on information exchange to update it's current state, and make subsequent processing decision(s). Agent-to-agent communication is accomplished via message passing mechanisms.

In this chapter, we will present two parallel filing models which utilize static and dynamic information to facilitate the filing process. The first model uses a "status broadcast" message passing mechanism while the second model uses a "result relay" mechanism.

6.1 Status Broadcast

For the "status broadcast" filing, each filing agent needs to broadcast to all of its immediate child folders as soon as the agent completes the filing at the folder (a filing is completed as soon as a depository decision of the target frame instance is made and the associated pointers are connected if necessary). Recipients of this broadcast message trigger their internal functions to check and determine if they need to start the filing (for each folder, its representative agent needs to ensure that all of its parent folders have already done the filing) or continue to wait.

We would like to point out here that the filing approach does not prohibit filing multiple frame instances simultaneously. We can assume that the system needs to establish a filing session for serving each frame instance fi when it arrives at the rooted folder, and the filing session terminates when the system completes the filing process for the fi. Since each frame instance to be filed has a unique identity, we can distinguish filing sessions that have been established to serve for different frame instances. With careful design, the only possible filing collision (i.e. filing two separate but identical frame instances simultaneously into the rooted folder which may end up depositing two copies, instead of just one copy, of the frame instance into the rooted folder) can easily be avoided.

Once the filing of a folder is completed, this folder should update its filing status and multicast the message "I have done the filing on <frame instance ID>" to its immediate child folders (disregarding the types of links). Implementation of this process can be as simply as flipping a flag for status change and addressing a message to multiple objects for status broadcast.

Figure 17 depicts the "status broadcast" filing model which describes the process transitions. As shown in Figure 17(a), a folder object (agent) is woken up and transited into the "check parent" state upon the reception of a broadcast signal (indicated by "b") from one of its parent folders. This folder agent will return to the "idle" state if there are some of its parent folders which have not done the filing. Otherwise, it performs the filing and subsequently broadcasts, disregarding the filing result, the "already filed" message to all of its immediate child folders, and returns to the "idle" state.
In the system level, there has to be a dedicated process to monitor the overall filing progress as shown in Figure 17(b). The filing process is completed when it detects that all filing agents are idle.

Given the folder organization in Figure 14, we use the "status broadcast" filing model to perform the filing. The filing procedure and the result are shown in Figure 18.

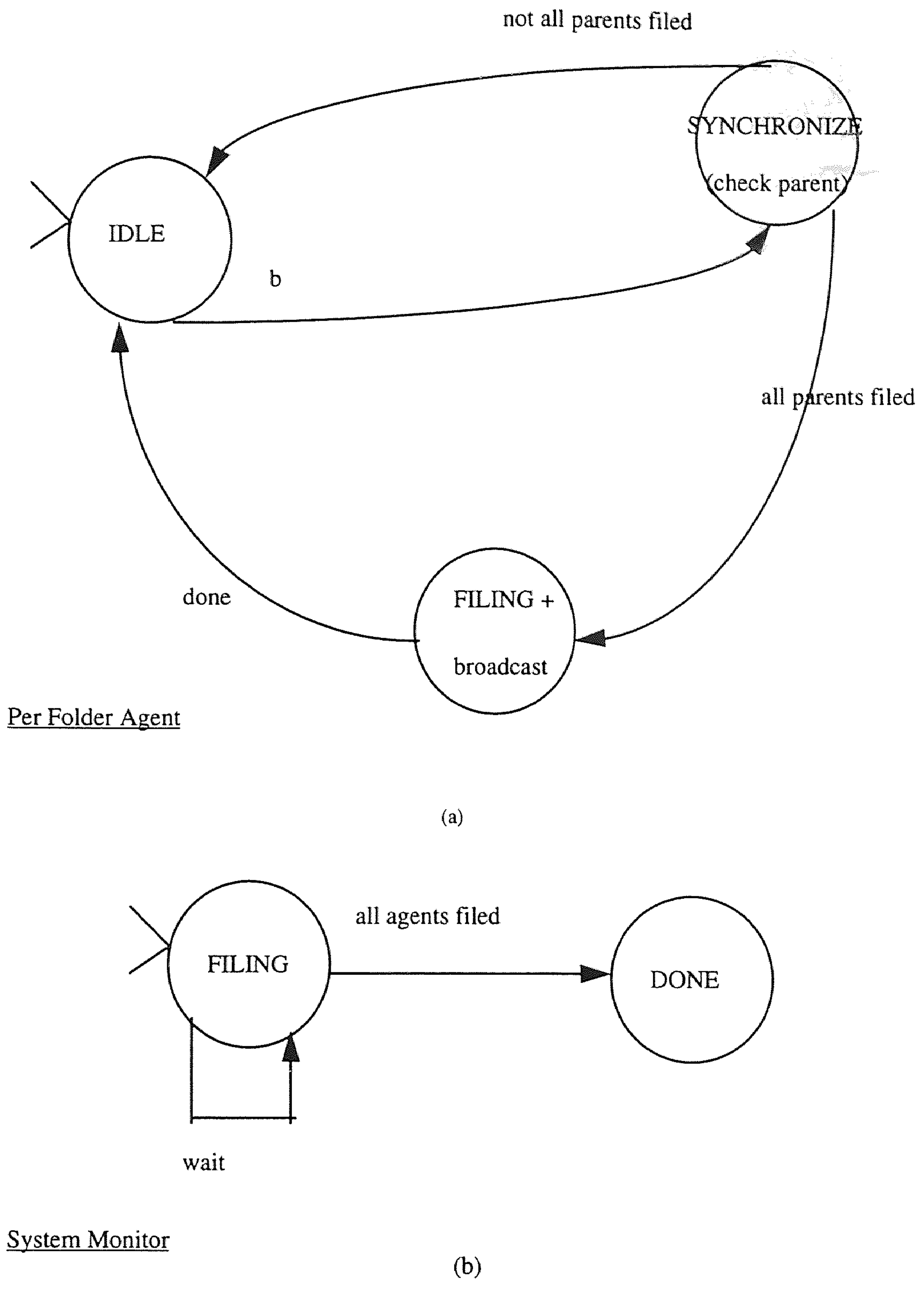
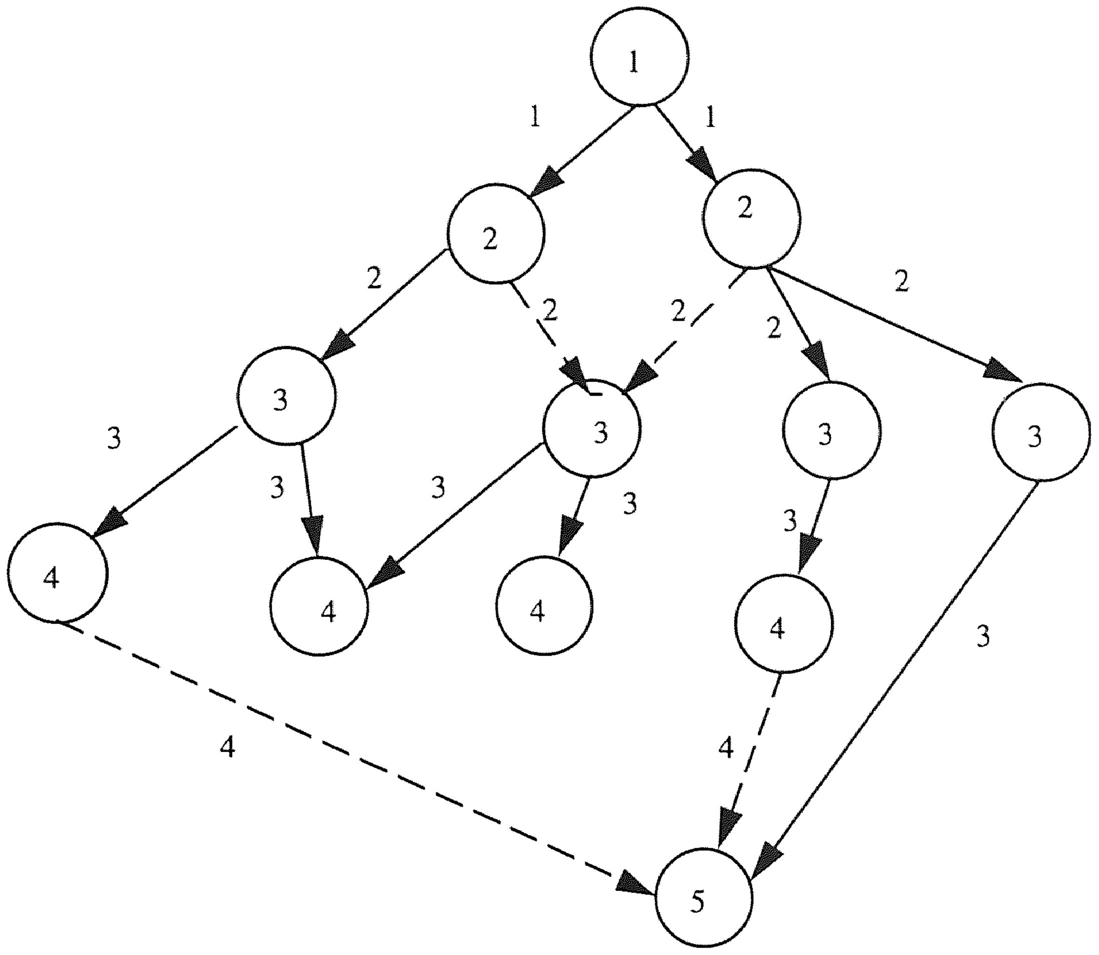


Figure 17 Parallel Filing (Status Broadcast)

The filing order of this model can be figured out since the timing of filing and then broadcasting its "completion" filing status of an individual agent is restricted by the synchronization rule which is presented in the "Complete-Parent-First" algorithm. Again, the numbers in the folder nodes indicate the filing order of each folder in the organization. And, the numbers along with the links indicate the order of broadcasting messages. Therefore, for the parallel filing, the system requires fewer steps to complete the filing for any given frame instance. The figure also shows that a total of four copies of a frame instance which are deposited into four shaded folders.

6.2 Result Relay

In the "result relay" filing model, the filing is not constrained by any particular order such as the one presented in the "Complete-Parent-First". Therefore, the filing accomplishes the tasks by maximizing the parallel processing. This is because the static information of folder organization and the dynamic information of different folders' filing status and filing results are made available. They can be used collectively to facilitate the decision making during the filing process. The "result relay" filing model is centered with the efficient use and manipulation of this information which, in turn, is a critical performance factor that governs the design of filing algorithms.



numbers in the folders: indicate filing order

numbers along the links: indicate sending order

shaded folders: frame instance deposited

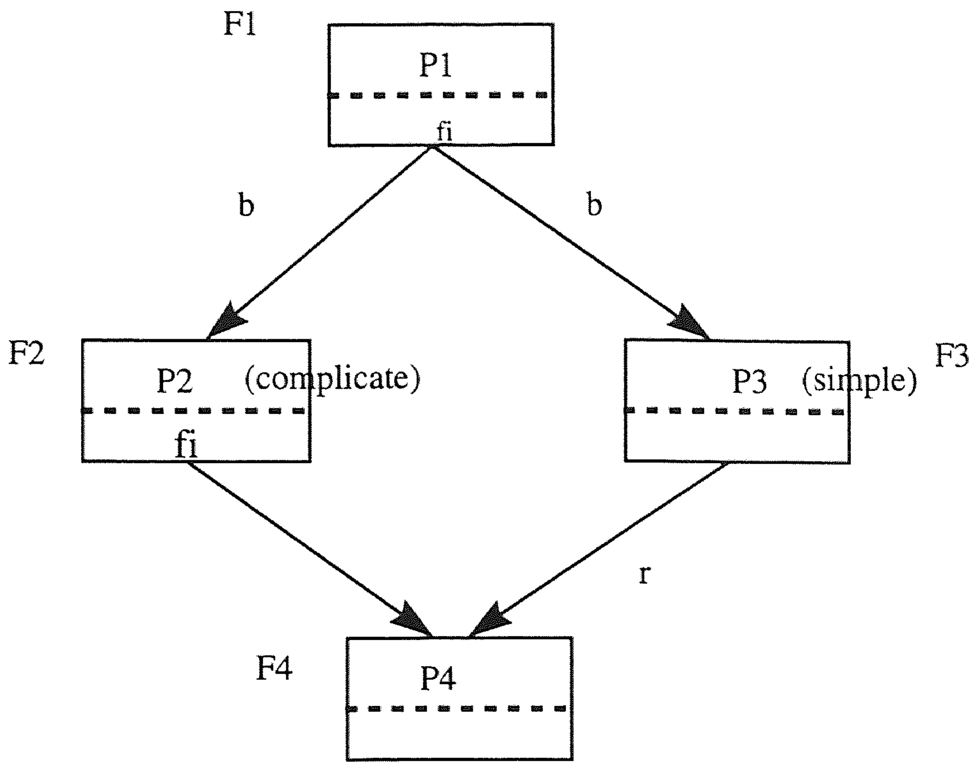
Figure 18 Status Broadcast Filing - Example

The semantics of folder relationships in this model are encapsulated in the filing algorithm. If two folders are related by a subfolder relationship, then a given frame instance cannot be deposited into the child folder if its related parent folder fails to have a copy first. If a child folder has virtual-folder links related to some parent folders, then it must have at least one of these parent folders having a copy of the frame instance first. In this filing model, disregarding the types of links, the filing agent broadcasts a "filing done" message to **all** of its immediate folders only if it has done the filing and a copy of the frame instance is deposited into the folder. The "filing done" message itself only indicates that the filing subsystem has finished the filing tasks on the target folder disregarding the filing result. If, however, there is a reason (such as the frame instance fails to qualify the folder's predicate), the filing process is thus done at this folder without depositing the frame instance. In this case, the filing agent sends a "no deposit" message to each of its immediate child folders which have subfolder links to the folder, and broadcasts the "filing done" message to its immediate child folders which have virtual-folder links to the folders (called controlled broadcast). Each of its child folders, in turn, sends the same "no deposit" messages to all of their immediate subfolders, and broadcasts "filing done" to others, and so forth. Therefore, once a "no deposit" occurs at any ancestral folder, the "no deposit" message will follow the subfolder links from this folder and relay folder by folder, as far as it can reach, through the entire folder organization. By propagating this message, each affected folder (the folder which falls on some relay paths) can instantaneously set its filing flag upon receiving the message and continues to relay a "no deposit" message or to broadcast "filing done" message, to its child folders. Those folders waiting for a folder to finish its filing may now be able to start its own filing process. Since each relay of the "no deposit" message may cause some chain-reactions of activating the downstream filing processes which offer opportunities for improving further the parallel filing performance. The filing performance of the relay algorithm is expected to be superior over the "status broadcast" filing algorithm. It is because in the "status broadcast" filing model, every folder of the folder organization is constrained by the "synchronization" rule. But the "result relay" filing model allows us to further reduce the number of folders that are still constrained by the rule. The synchronization rule cannot be removed entirely, but it can only be relaxed in the "result relay" model. We will use the next example to describe how a child folder in the "result relay" filing model may finish the filing before its parent folders do. As shown in Figure 19, given a folder organization containing four folders F1, F2, F3, and F4 with four subfolder relations (all of the links are defined as subfolder links). Assume that a frame instance fi qualifies the local predicates P1 and P2 of the folders F1 and F2 respectively. The frame instance fi will be eventually filed into the folders F1 and F2. During the filing process, it is possible that F4 finishes its filing before F2, even though F2 is a parent of F4. This occurs when the folder F2 is busy in performing its filing process (assume that the local predicate P2 of the folder F2 is a complicated predicate, and the evaluation of whether the frame instance qualifies P2 takes a lot of time), the folder relays quickly a "no deposit" message to F4 through the subfolder link (F3,F4) (assume that the local predicate P3 of the folder F3 is a simple predicate) after the folder F3 completes its filing process and the frame instance is not deposited in it. When the folder F4 receives the message from F3, it immediately updates its filing status to "completion". Therefore, a possible filing order in this example could be F1->F3->F4->F2.

Figure 20 depicts the "result relay filing model". It is possible to relax the "synchronization" constraint in this model. A folder may continuously receive "status broadcast" and/or "result relay" messages from other folders even if it has already completed the filing. The messages should be ignored under this circumstance[3].

Figure 21 again employs the "result relay" filing model to perform the filing task on the same folder organization as shown in Figure 14. Assume that the right child of the rooted folder detects that the filing process does not deposit a copy of a frame instance into itself. It propagates a "no deposit" message through its subfolder linked children (the right child folder of the rooted folder has two subfolder-linked child folders). The "no deposit" message is continuously relayed downward to all of the subfolder-linked descendant folders. On the other hand, a copy of the frame instance is deposited into the left child of the rooted folder. The "status broadcast" filing process is followed. Because the "synchronization" constraint is relaxed on the right hand side of the filing process, the bottom-most folder in the organization may complete its filing process before some of its parent folders.

[3] The folder should check its own filing status first upon receiving a message from others.



example filing order: F1 -> F3 -> F4 -> F2

fi: frame instance

b: “filing done” broadcast

r: “no deposit” relay

Fi: folder

Pi: predicate

Figure 19 Relax the Filing Precedence Constraint

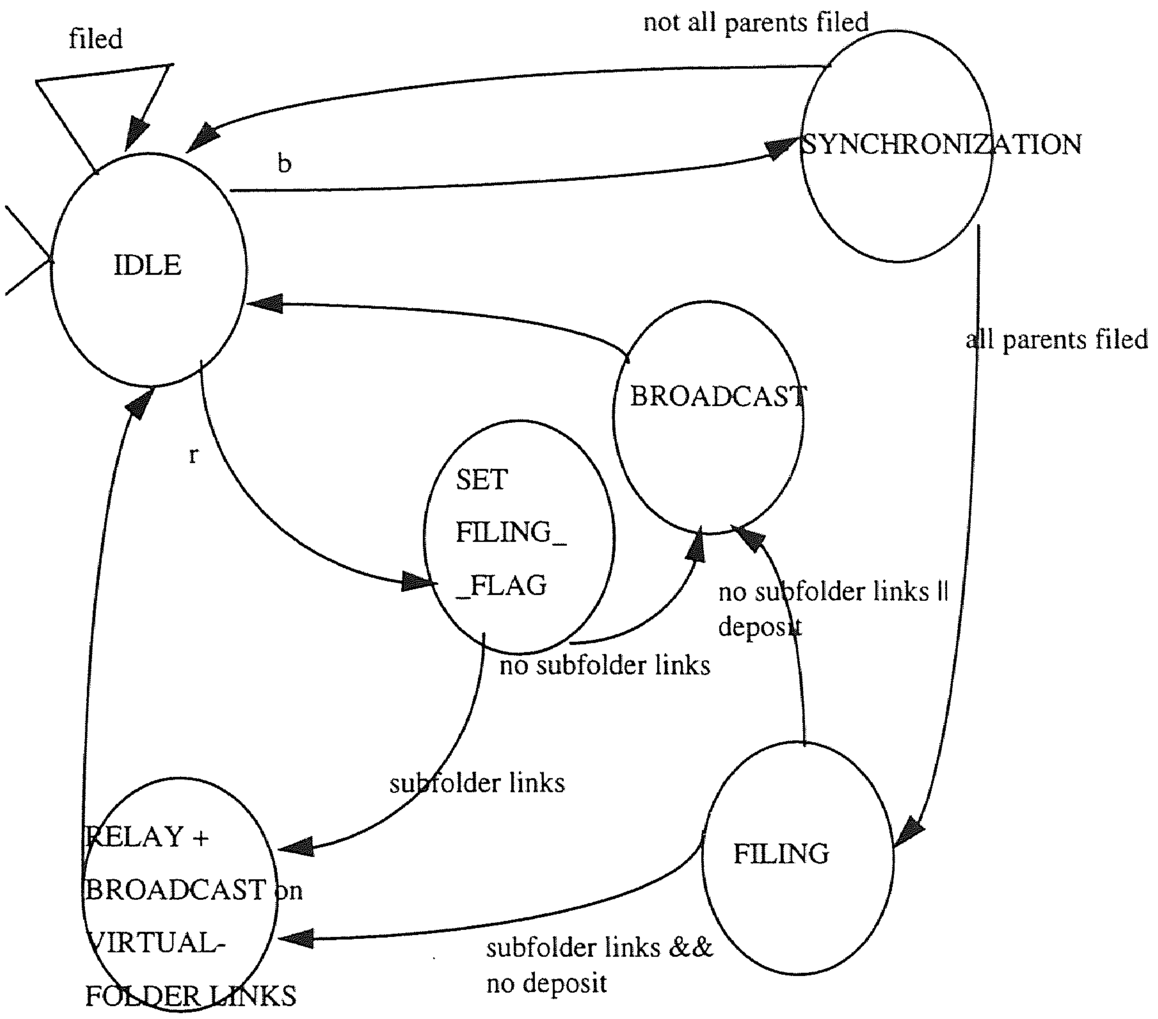
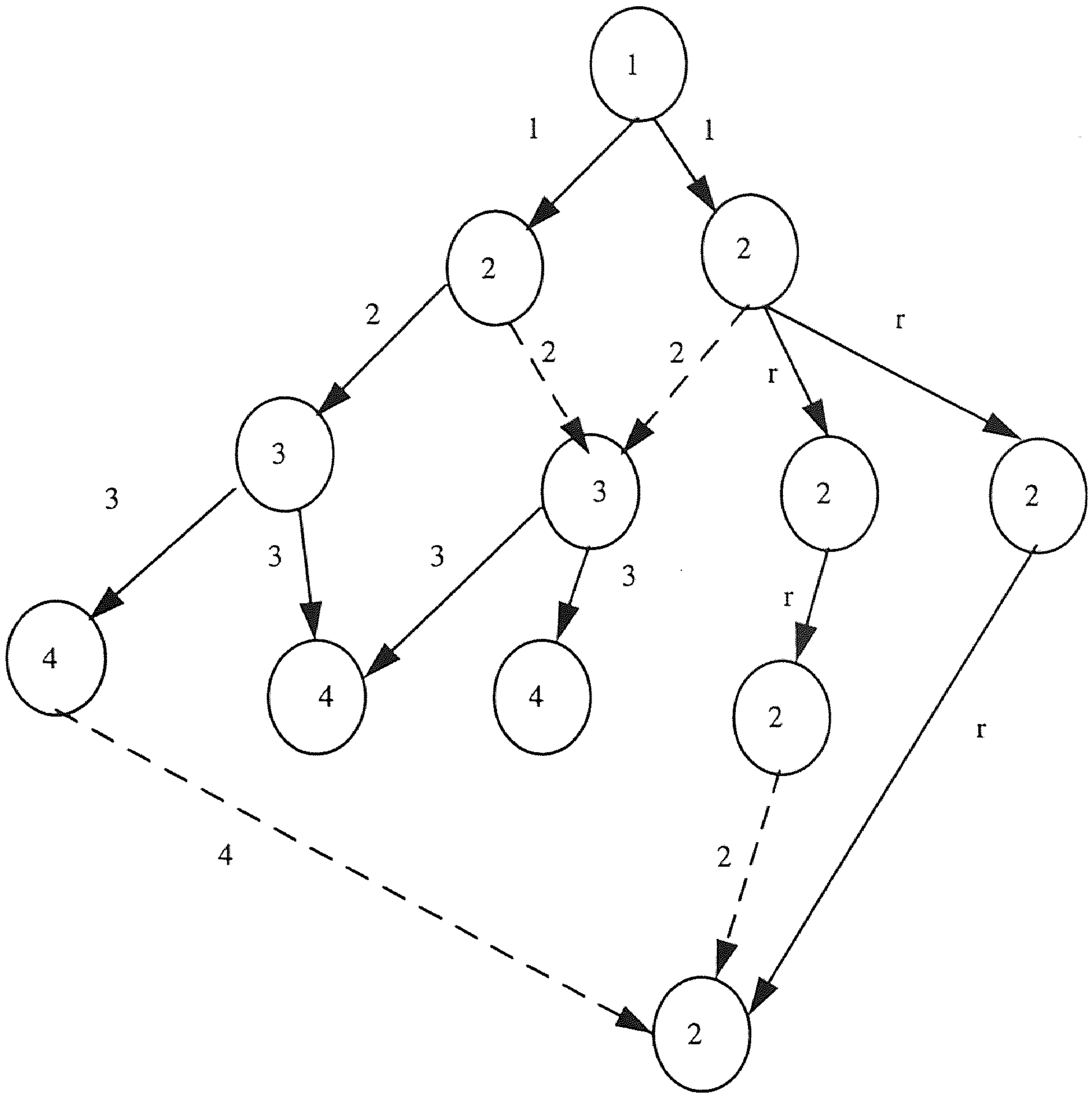


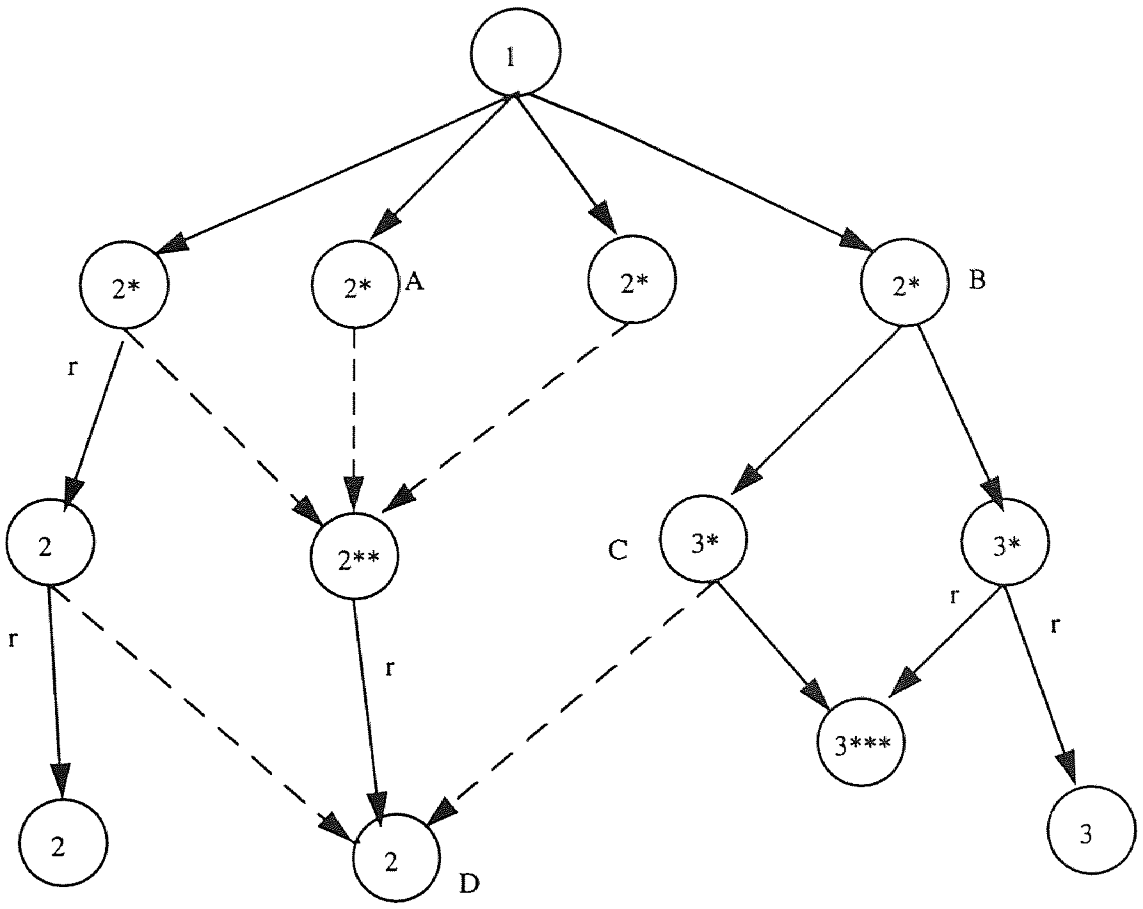
Figure 20 Parallel Filing (Result Relay)

Figure 22 gives another example of a "result relay" filing model. It is easy to see that the filing order is difficult to predict because it relies on the performance of real-time dynamic frame instance distribution, which, in turn, depends on the static organization structure and predicate specifications. For instance, the filing process proceeds to file a given frame instance in the folder D before the folder C; and the process proceeds to file the frame instance in the folder D before the folder B if the predicate of the folder A is much simpler than those specified for the folder B and the folder C. On the other hand, if the predicate of the folder A is somehow more complicated than those predicates of the folders B and C, then the filing process is expected to proceed to file a given frame instance into the folders B and C before the folder D.



- number in folder: indicates filing order
- number along with link: indicates traverse order
- shaded folder: frame instance deposited
- r: result relay

Figure 21 Result Relay Filing - Example



Number in the node: indicates a possible filing order

*   check relational constraint & evaluate predicate (**t1**)

**   check relational constraint only (**t2**)

***   may or may not need to check relational constraint (**t3**)

t1 >> t2 >= t3

Figure 22 Another Example of Result Relay Filing

CHAPTER 7

PERFORMANCE ANALYSIS FOR PARALLEL FILING

There are three different approaches to accomplishing the filing task, the sequential approach, the status broadcast approach and the result relay approach. The sequential approach represented by the "Complete-Parent-First" algorithm, as shown in Figure 14, requires a total of twelve nodal processing time units to complete the filing for the given folder organization. A nodal processing time unit is defined as the average time required to finish the filing task for a folder, assuming that each folder's predicate in the organization has the same complexity. In contrast to the nodal processing time, the message propagation time becomes insignificant because the node only needs to send a signal out to its adjacent node.

For the "Status Broadcast" filing model, the initial parallel filing approach, broadcasts a folder's filing status to its immediate child folders when it completed the filing of the folder, and subsequently, its child folders can re-examine each respective readiness for filing (Are all parent folders completed the filing?). As shown in Figure 18, the required nodal processing time for completing the filing on the same folder organization as in Figure 14 is now reduced to five units. This gives a significant saving in the overall system processing time.

The "Status Broadcast" filing model can be further modified to be a "Result Relay" filing model, which takes full advantage of the parallel filing approach to accomplishing the filing tasks. The primary restriction imposed on the other two models is that the filing on a given folder would not be proceeded unless all parent folders of this given folder completed their filings. This restriction can be relaxed in this result relay filing model. The algorithm designed for this filing approach utilizes both static and dynamic system information to accomplish the filing task. As shown in Figure 21, the nodal processing time of the "Result Relay" filing is further reduced to four. The additional saving is obtained from being aware that the given frame instance is not deposited into the very first right child of the rooted folder, and, therefore, a specially formulated message can be quickly relayed (propagated) downwards to all the subfolder link-connected descendant folders. The example also shows that a no-deposit message has already been relayed to the very bottom folder (initiated by the right child of the rooted folder) even though some of this folder's parent folders are still performing their filings.

CHAPTER 8

DOCUMENT MANAGEMENT BASED ON I-ORG

In this chapter, the management performance of an I-ORG based folder reorganization will first be investigated, which is superior to that of an equivalent U-ORG based architecture, by presenting the following simple folder reorganization activity: reset a folder to another parent folders.

As shown in Figure 23, suppose that the folder X in the organization which currently has two parent folders (folder A and folder B), contains frame instances found in both A and B which satisfy the local predicate of the folder X. In the U-ORG representation, the local predicate of X is $L(X) \wedge G(A) \wedge G(B)$, where $G(A)$ and $G(B)$ are the global predicates of the folder A and the folder B, respectively. If the user requests to change the connection of folder X from B to C (called rehoming X from B to C), then the folder X should contain only the frame instances which satisfy the revised local predicate of X, $L(X) \wedge G(A) \wedge G(C)$. With U-ORG representation, the user modifies first the local predicate of X, changing from $L(X) \wedge G(A) \wedge G(B)$ to $L(X) \wedge G(A) \wedge G(C)$, wipes out the content of the folder X, and then redistributes frame instances from the folders A and C which satisfy the new local predicate of X. For this case, the system needs to redistribute all frame instances found in $A \vee C$ (the union of the two parent folders). If $A \wedge B = \varnothing$, then it gives the worst case where the system needs to redistribute all frame instances found in the two parent folders.

With I-ORG representation, the user simply removes the subfolder link from folder B to folder X, and creates a subfolder link to connect folder C and folder X without changing the associated local predicate for X, which is L(X). For frame instance redistribution, the system only needs to redistribute the common frame instances from A $\wedge$ C (the intersection of the two parent folders).

It is easy to see the required management tasks in response to the above proposed reorganization under different model representations. The performance becomes notable as the sizes of the two new parent folders (A and C) grow very large, but only a few instances are in common. For example, as shown in Figure 23, assume that the folder A contains 1000 frame instances; the folder B contains 100 frame instances, and the folder C contains 2 frame instances. After the reorganization, we need to redistribute at least 1000 frame instances from (A $\vee$ C) in the U-ORG representation while we only need to redistribute 2 frame instances from (A $\wedge$ C) in the I-ORG representation.

For the document retrieval, since there exists certain similarities between folders' predicate specification and the way a user specifies criteria to retrieve his documents, the enhanced folder organization might prove helpful in simplifying user query specification and improve response time in the document retrieval process. If the similarities between predicate and query specification exist, then every user's query on the folder organization could be translated to or represented as a folder predicate. Associated with this predicate there is a view folder (a folder which may or may not be shown to the user). The system could create and attach this folder to the existing folder organization; and it should contain the retrieved documents in response to the query. In this way, the response time of some complicated or frequent queries could be significantly improved because the system only needs to identify the view folder and return its contents to the user, without going through the entire time-consuming query processing and document retrieval procedures.

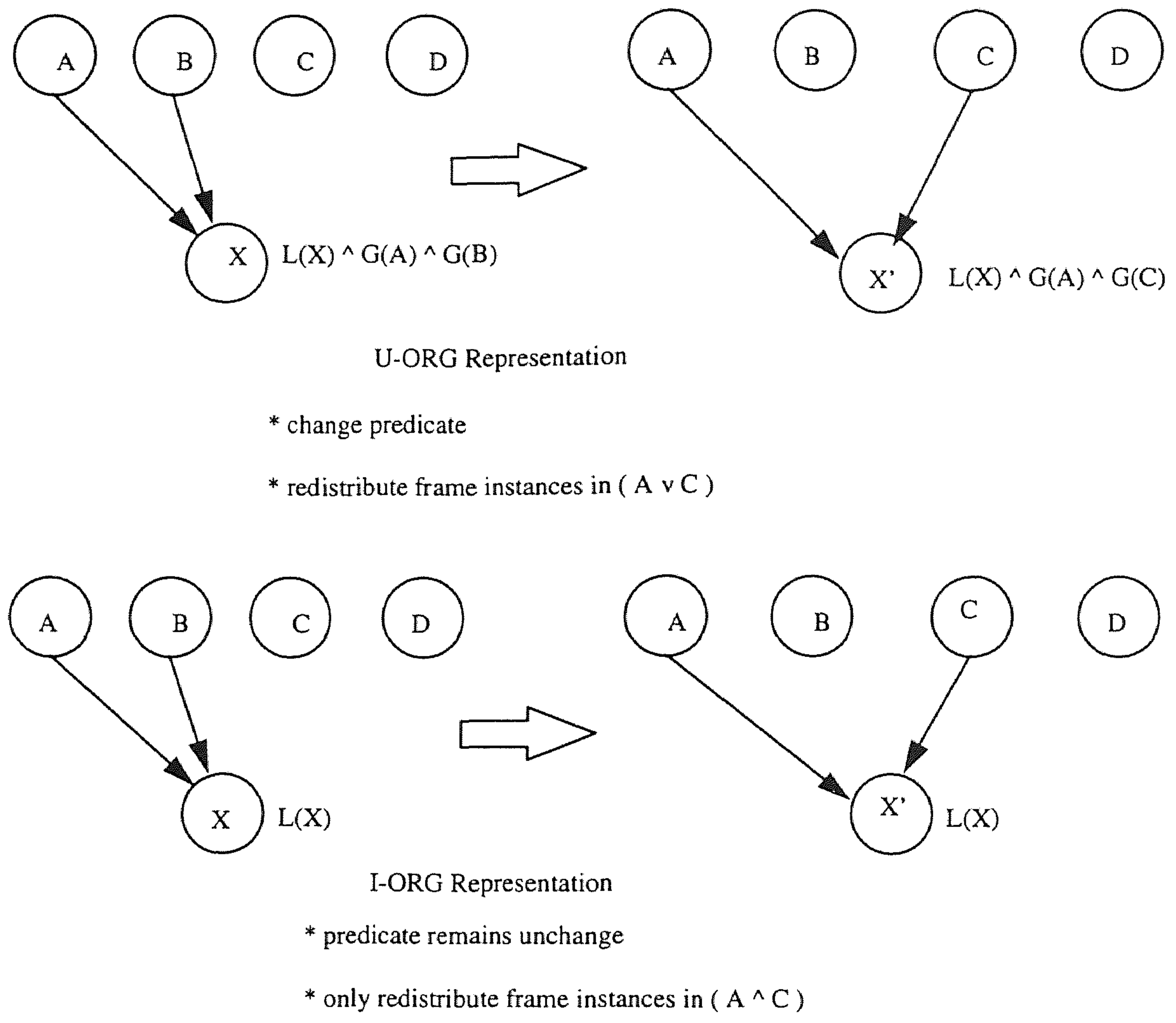

A
B
C
D
X
L(X) ^ G(A) ^ G(B)
X'
L(X) ^ G(A) ^ G(C)
U-ORG Representation
* change predicate
* redistribute frame instances in (A v C)
X
L(X)
X'
L(X)
I-ORG Representation
* predicate remains unchange
* only redistribute frame instances in (A ^ C)

Figure 23 Reorganization (Rehome)

As an example, suppose that a user enters the following selection criteria against the folder organization as shown in Figure 9, requests to view records of people who are "older than 30 and enrolled in either a Ph.D or MS program". Without taking into consideration the way of specifying the corresponding predicate, a view folder can be created and attached to the three parent folders of the folder organization: Emp_of_Age_30+, PHD, and MS folders. This view folder should contain all of the records pertinent to the user's interest, upon which the user can retrieve and record easily.

Having such a view folder created as part of the folder organization, all of the subsequent document filing will deposit the documents, which qualify its criteria, into this view folder. Retrieving the content of the view folder is even simpler, it needs to return to the user all the frame instances contained in the folder. Without this, the retrieval subsystem needs to understand the overall information structure of the folder organization. Before the query result can be determined, the system needs to process and analyze the query, to identify candidate folders from the organization, and then to discover, filter and evaluate the relevant documents from the candidate folders. If the user wants to retrieve the same kind of (frequently accessed) documents from his filing system at any time, the system needs to execute repeatedly the same process of formulating the query, interpreting it and retrieving the intended documents.

The new construction of the folder organization model is introduced in such a way that the related objects (documents) with similar properties in nature, which would be possibly selected in response to a specific and frequent query, are collected into a single repository (regular or view folder). The desired "selection criteria," as specified in the original query statement is now represented in the construction of this single repository in the folder organization. It consists of a part describing the well-defined folder relationships and a part describing the local predicate of that folder. For instance, the selection criteria in the above example can be represented by a view folder in the folder organization, where the local predicate of this view folder is "True" and the folder is connected to the three parent folders with virtual-folder links.

CHAPTER 9

IMPLEMENTATION GUIDELINES (OBJECT MODELING)

The filing and retrieval subsystem deals with the following data elements: folders, links, predicates, frame instances and/or the original documents. Each of these data elements can be modeled as an object. Relationships among these objects may become complex, and the degree of inter-connectivity among different entities may become very high [88]. Moreover, the filing and retrieval subsystem may need to support other non-standard data types such as an application-specific document or a multimedia document [55,72,85]. The document may be synthesized by various components of different data type such as voice message, text, picture and video. To support this document, the system performance becomes poor as the folder organization grows in size so that eventually it becomes too large and complicated to manage. In order to prevent this from happening, we should consider implementing the system by employing object-oriented technologies. Another good reason for considering the object approach for our implementation is the nature of the problems that we are modeling. It is a real world problem - seeking ways for automating the document processing in today's office environment. The object technology has demonstrated its superior expressive power in closing the semantic gap between problem domain and solution environment. It represents consistently the information and knowledge gathering from different dimensions of the real world and demonstrates more resilient to future changes (i.e., an object based system is more stable when handling changes or enhancements). In addition, the modeling of these problems can be directly translated into internal repository schema if the designed model and document bases are both supported by an object database system.

We begin our model design with five classes: folder, link (includes subfolder-link and virtual-folder link), frame instance, local predicate and global predicate. Basically, we employ the three relations defined and used by all major object modeling methodologies: inheritance (is_a), aggregation (has_a), and other relationship, where the other relationship must be associated with a role[4].

In our current design, both global and local predicate classes are defined as subclasses of the predicate class, and subfolder link and virtual-folder link classes are subclasses of the link class. This model may be further refined as we move forward when some of the current uncertainties are resolved.

Figure 24 gives a object reference model which shows some desired relationships among those classes.

[4] For example, an other relationship can be defined between a lecturer class and a student class where a lecturer plays a teacher's role in a particular classroom.

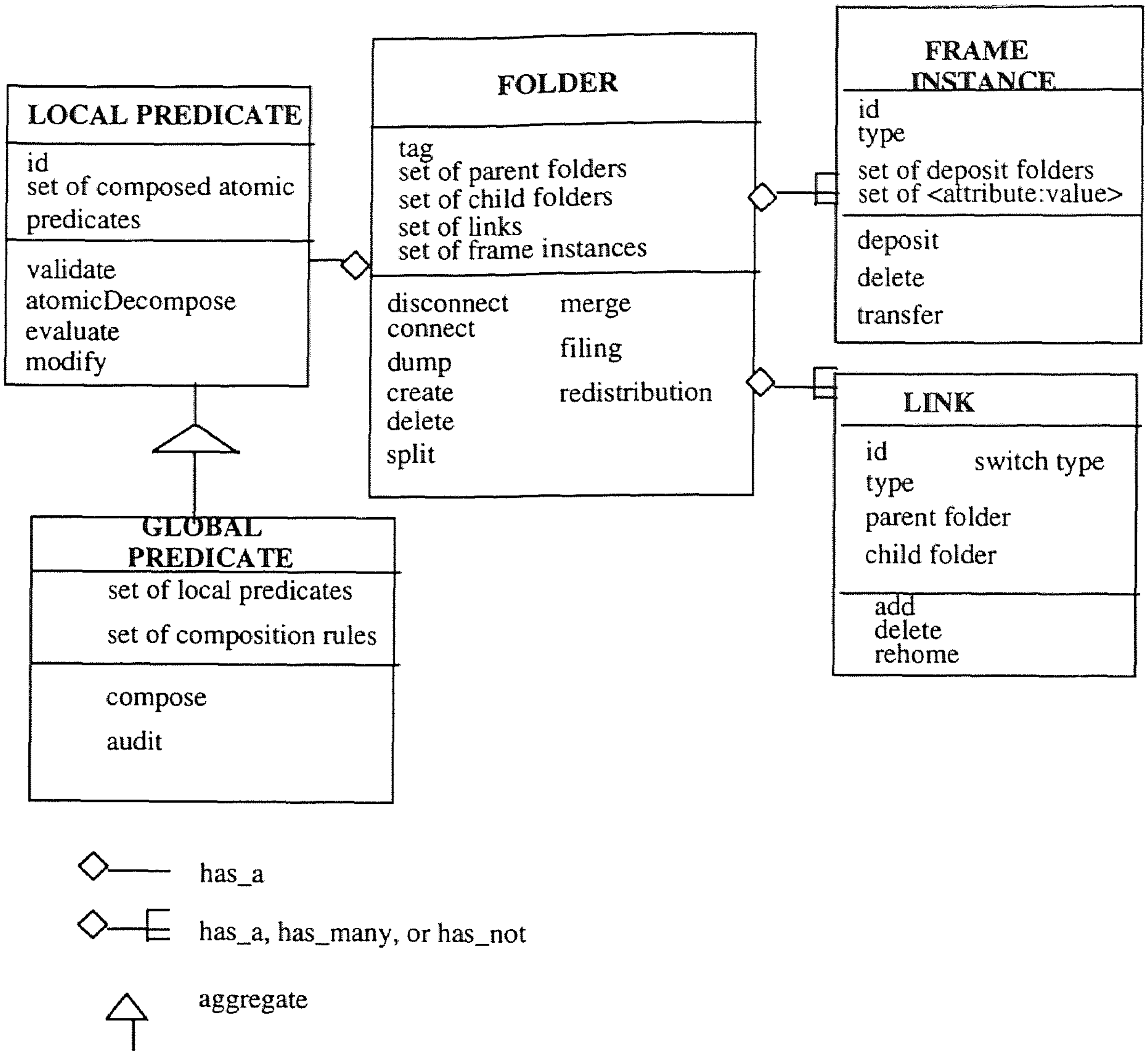


Figure 24 An Object Model of Folder Organization

CHAPTER 10

U-ORG TO I-ORG TRANSFORMATION

In comparison with the I-ORG, which is operational more efficient, the U-ORG has its simplicity because it maintains only a single type of link. Therefore, the implementation of the system can have two models which represent the folder organization at two different levels: the user interface level (or the external representation using U-ORG), and the system execution level (or the internal representation using I-ORG). Interoperabilities between the two models needs to be well-coordinated and kept transparent to the user while the system optimizes its performance by utilizing the architectural strength from both models.

In this Chapter, we shall explore the co-existence issues between the U-ORG and the I-ORG folder organization and present a preliminary transformation process which transforms the given U-ORG into an I-ORG representation when the user folder organization is constructed.

10.1 Initial Transformation Between U-ORG and I-ORG

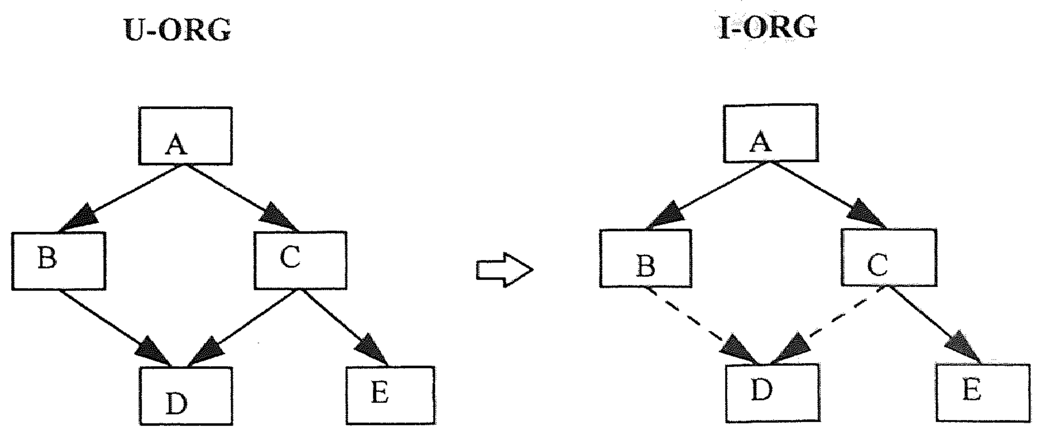
Given an U-ORG folder organization, it can be transformed into an I-ORG representation by replacing all of the U-ORG links with either subfolder links or virtual-folder links in the I-ORG representation. The transformation rules are presented in the following algorithm:

Algorithm

```
For each folder except the rooted folder
    If number of incoming links = 1
    Then    replace the link with subfolder link
    Else    replace all incoming links with virtual-folder links
    End If
End For
```

All of the predicates remain unchanged and the resultant I-ORG folder organization is equivalent to the given U-ORG folder organization because the link definition of the U-ORG is exactly the same as the virtual-folder link definition of the I-ORG (i.e., the links defined in the two respective folder organizations are interchangeable).

Figure 25 gives an example of transforming an U-ORG into an I-ORG representation. Conversely, an I-ORG folder organization can be transformed into an U-ORG representation accordingly.



all local predicates remain intact

Figure 25 Initial Transformation

Algorithm

```
For each folder C except the rooted folder
        If at least one incoming links is a subfolder link
        // Assume that (S1,C), (S2,C),.., (Sm,C) are subfolder links and (V1,C), (V2,C),..,
        // (Vn,C) are virtual-folder links where Si's ($1 \le i \le m$) and Vj's ($1 \le j \le n$) are parent
        // folders of C.
        Then    modify the local predicate of C, L(C), to
                $L'(C) = L(C) \wedge [\wedge G(Si)] \wedge [\vee G(Vj)]$
        // G(Si) and G(Vj) are global predicates of folder Si and Vj respectively.
        End If
        Replace all of the links (both subfolder links and virtual-folder links) by the only
        type of links as defined for the U-ORG.
End For
```

The change of the local predicate of every folder which has incoming subfolder link(s), requires the modification of the local predicate. For instance, let a folder X have one parent folder S1 with subfolder relationship and $n \geq 2$ number of parent folders V1, V2,...Vn with virtual-folder relationships. Let G(S1), G(V1), G(V2),..,G(Vn), be the global predicates of the folders S1, V1, V2,.., Vn respectively. Let L(X) be the local predicate of the folder X. The new local predicate of the folder X after transforming into the U-ORG becomes $L'(X) = L(X) \wedge G(S1) \wedge [\vee G(Vi), 1 \leq i \leq n]$. This new local predicate L'(X), characterizes the semantics of the original links which include one subfolder link (S1,X), and n virtual-folder links (V1,X), (V2,X),..;(Vn,X). After the change of the predicate, we proceed to replace all of the links (both subfolder links and virtual-folder links) by the only type of links defined for the U-ORG. Now, we should ensure the content of the folder organization remains intact for transforming from U-ORG into I-ORG and vice versa.

10.2 Content Equivalence

The content equivalence between two folders of two different folder organizations can be defined as follows: Given a folder F_A in ORG-A and a folder F_B in ORG-B, the folder F_A is content-equivalent to the folder F_B if their contents are identical. We then prove the equivalence between the two organizations by showing that the content of the folder X in an I-ORG has not changed after transforming it into the U-ORG representation. Any frame instance fi found in the folder X of the I-ORG representation has the following three facts: (a) it satisfies L(X), the local predicate of X; (b) fi must appear in the folder S1 which has a subfolder link connecting to X; and (c) fi must appear in at least one of the folders V1, V2,.., and Vn which have virtual-folder links connecting to X.

Now, in the U-ORG representation, the new local predicate of the folder X becomes $L'(X) = L(X) \wedge G(S1) \wedge [\vee\, G(Vi),\ 1 \leq i \leq n]$. A frame instance fi may flow into folder X via any of the possible paths: (S1,X), (V1,X), (V2,X),..., or (Vn,X), since we have only one type of link defined for the U-ORG. But regardless of which path fi might take for filing into the folder X, the implication of the above three facts (a), (b) and (c) supports the following conclusions: (a') fi satisfies L(X), the local predicate of X in the original I-ORG (from (a)); (b') fi satisfies G(S1) because fi is found in the S1 (from (b)); and (c') fi also satisfies $G(V1) \vee G(V2) \vee \ldots \vee G(Vn)$ because fi is found in either V1, V2,...or Vn in the original I-ORG representation (from (c)). Therefore, we conclude that fi satisfies $L(X) \wedge G(S1) \wedge [\, G(V1) \vee G(V2) \vee \ldots \vee G(Vn)]$, or L'(X), the local predicate of the folder X in the U-ORG representation.

Similarly, it can be shown that any incoming frame instance which is deposited into folder X in the U-ORG will be deposited into the folder X in the I-ORG representation as well.

10.3 Definitions

The direct transformation of a given U-ORG into its content-equivalent I-ORG as described above is just the first step in the conversion process. We need fine-tune this initial representation to become an optimal I-ORG representation. Before the remainder of the transformation procedure is presented, we need to introduce the following terminology definitions:

(a) Supnet: The supnet of a given folder F is the part of the folder organization that does not contain any descendant folder of F.

(b) Common Parent Folder (CPF): A CPF of a given folder set S, is a folder where all paths from the rooted folder to any folder in S intersect (since rooted folder is a CPF of any S, so at least one CPF can be found for any given folder set of a folder organization).

(c) Common Parent Folder Chain (CPFC): All CPFs of a given folder set S form a direct chain. This direct chain is called CPF chain (the rooted folder is contained in the CPFC as default).

(d) Least Common Parent Folder (LCPF): The leaf CPF of the CPFC is called the LCPF of the given folder set F (again, if the rooted folder is the only node in the CPFC, then it is also the LCPF).

The examples shown in Figure 26 and Figure 27 illustrate the above definitions.

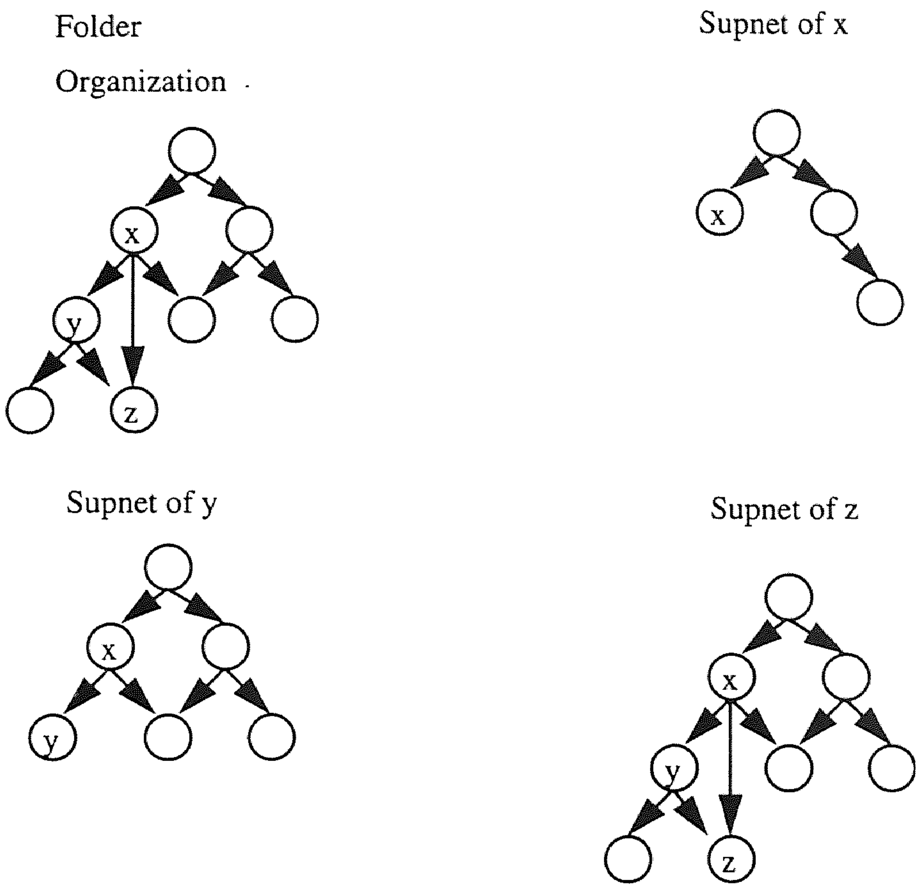
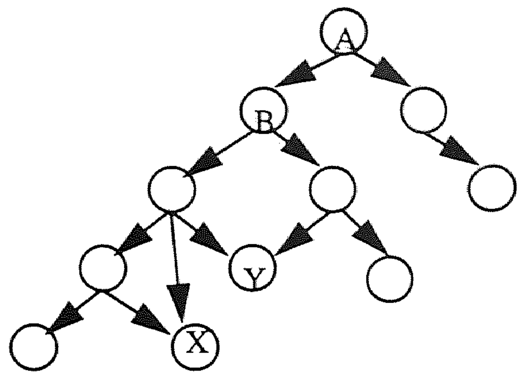


Figure 26 Supnet Examples

10.4 Transformation - Overview

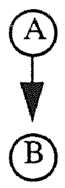
This section continues the transformation process to further refine the I-ORG generated from the initial direct transformation described above.



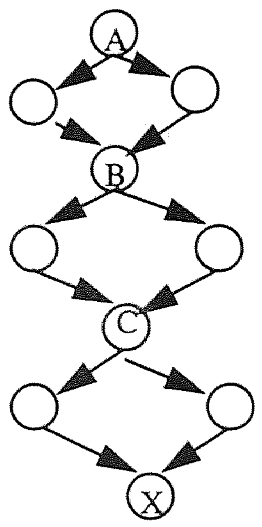
S = {X, Y}

CPF of S are: A and B

CPF Chain:



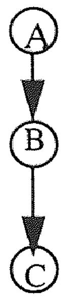
LCPF: B



S = {X}

CPF of S are: A, B and C

CPF Chain:



LCPF: C

Figure 27 CPF, CPFC and LCPF Examples

The transformation process starts from the rooted folder, follows the topological sorting order and visits every folder in the folder organization. For each stop at a folder, the same procedure repeat. This procedure is performed by running a step-locked reduction process on the two structures - the supnet of the target folder and the logical predicate representation of the folder. The two structures will be eventually reduced to their simplest forms and merged to reflect how a complicated predicate is simplified at the end of the reduction process. At each step of the reduction, the two structures need to reference each other before changing their states (or called step-locked procedure).

The local predicate of a folder to be transformed is first decomposed into a set of atomic predicates and represented by a logical structure. The logical structure is simplified by removing the irrelevant atomic predicates and identifying the portion of the structure which can be deleted from the structure and transformed into proper folder relationships without losing the semantics of the original predicate. The removed predicates will be kept in a set for reconstructing the local predicate of the folder at the end of reduction process. As each reduction of logical structure is done, the remaining structure is locked for reference and followed by the removal of non-interested folders from its structural counterpartner, the supnet of the target folder. When the predicate of certain folder in the supnet is not referenced in the locked predicate structure, the folder and its descendant folders together, are removed from the supnet. So the size of the supnet organization may be also reduced. Performing step by step, the transformation process continues until both the supnet and the predicate logical structure reach their simplest forms and cannot go any further. The two reduced structures are then merged and the local predicate is reconstructed to reflect the transformation result: the simplified predicate structure and the modified folder organization. If the simplified predicate is a composite predicate, it will be transformed into an equivalent Partial Folder Organization (PFO) which comprises folders, dummy folders and folder relationships. A dummy folder is a folder whose local predicate is assigned to be "true". One purpose to create dummy folders for instance, is for the sake of maintaining the integrity of folder organization.

Each folder in the PFO is associated with an atomic predicate and logical operators appearing in the original composite predicate are represented by dummy folders, and proper subfolder or virtual-folder relationships. The PFO is then attached to the folder organization to complete the transformation cycle for the given folder and its associated local predicate. Following the topological sorting order, the same procedure is applied to every folder in the folder organization.

10.5 Transformation - Details

Facilitating with examples, the remainder of this section describes in details the step-locked transformation process. As mentioned above, the same transformation procedure is applied to every folder the process visits. We shall organize the procedure into the following eleven steps. If folder F in a given folder organization is the target folder to be transformed, then (1) all folders except F appearing in the supnet of F have already completed their transformations and each of them is associated with an atomic predicate, (2) the portion of the local predicate of F, L(F), to be transformed into folder relationships can only be linked to the ancestral folders of F in order to maintain a valid RDAG graph, therefore, we are only interested in the supnet of F rather than the subnet of F which comprises all descendant folders of F in the folder organization, and (3) as resulting from the initial transformation, if the number of incoming links toward folder F is greater than one, the links are virtual-folder links. The rest of the links in the supnet of F could be either subfolder links or virtual-folder links.

With these three implications in mind, we start listing the eleven transformation steps as follows:

Step 1: Generate the global predicate P(F) for the folder F.

Step 2: Identify the supnet of folder F, Sup(F).

Step 3: Examine the global predicate of F and remove the irrelevant atomic predicates. The removed predicates will be kept in a set R for reconstructing the local predicate of F at the end of the process. Irrelevant atomic predicates are those predicates which are not associated with any folders in the supnet of F. Let P'(F) be the resulting global predicate of F after this step.

Step 4: Identify the CPF, CPFC and LCPF from the supnet of F where S={F} in this context.

Step 5: Use a logical structure Log[P'(F)] to represent P'(F). Refer to Figure 8 for an example. Simplify the Log[P'(F)] by applying rules for logical operations such as:

(1) $(A \wedge B) \wedge (C \wedge D) = A \wedge B \wedge C \wedge D$

(2) $A \vee (A \wedge B) = A$ and $A \wedge (A \vee B) = A$

$$(3)\ (A \wedge B) \vee (C \wedge B) = B \wedge (A \vee C)$$

Step 6: Now the step-locked reduction starts with the two structures: Sup(F), the supnet of F and Log[P'(F)], the logical representation of P'(F).

Step 7: Log[P'(F)] - Remove all atomic predicates associated with any ancestral folders of LCPF. Suppose Log'[P'(F)] is the resulting structure after this step.

Sup(F) - Remove all ancestral folders of LCPF and disconnect F from the sup(F) since it will be reconfigured afterward. Suppose Sup'(F) is the resulting organization after this step and notice that the LCPF is now the rooted folder of Sup'(F).

Step 8: For each child folder C of LCPF in the Sup'(F), do the following:

Step 8': If its atomic predicate L(C) is not found in any part of Log'[P'(F)] or $L(C) \notin$ Log'[P'(F)], identify the subnet of C, sub(C), where sub(C) contains all descendant folders of C and the incoming links to C initially but needs to be reconfigured as follows: If any folder C' in sub(C) whose local predicate can be found in the predicate structure, or $L(C') \in$ Log'[P'(F)] and, (1) has incoming links from folders in the [sup'(F) - sub(C)] or $\neg$ sub(C), and (2) has an incoming subfolder link from another folder in the sub(C), then make sub(C) to also include those incoming links from [sup'(F) - sub(C)]. On the other hand, if the condition (1) remains unchanged, but the condition (2) changes to have an incoming virtual-folder link from another folder in the sub(C), then the sub(C) should be reconfigured to remove folder C' and its descendant folders from it, but still retain the incoming virtual-folder link to the C'. For those C' where L(C') ∈ Log'[P'(F)] but do not meet either of the above conditions, their predicates should be moved into R.

Figure 28 gives one example for each respective cases.

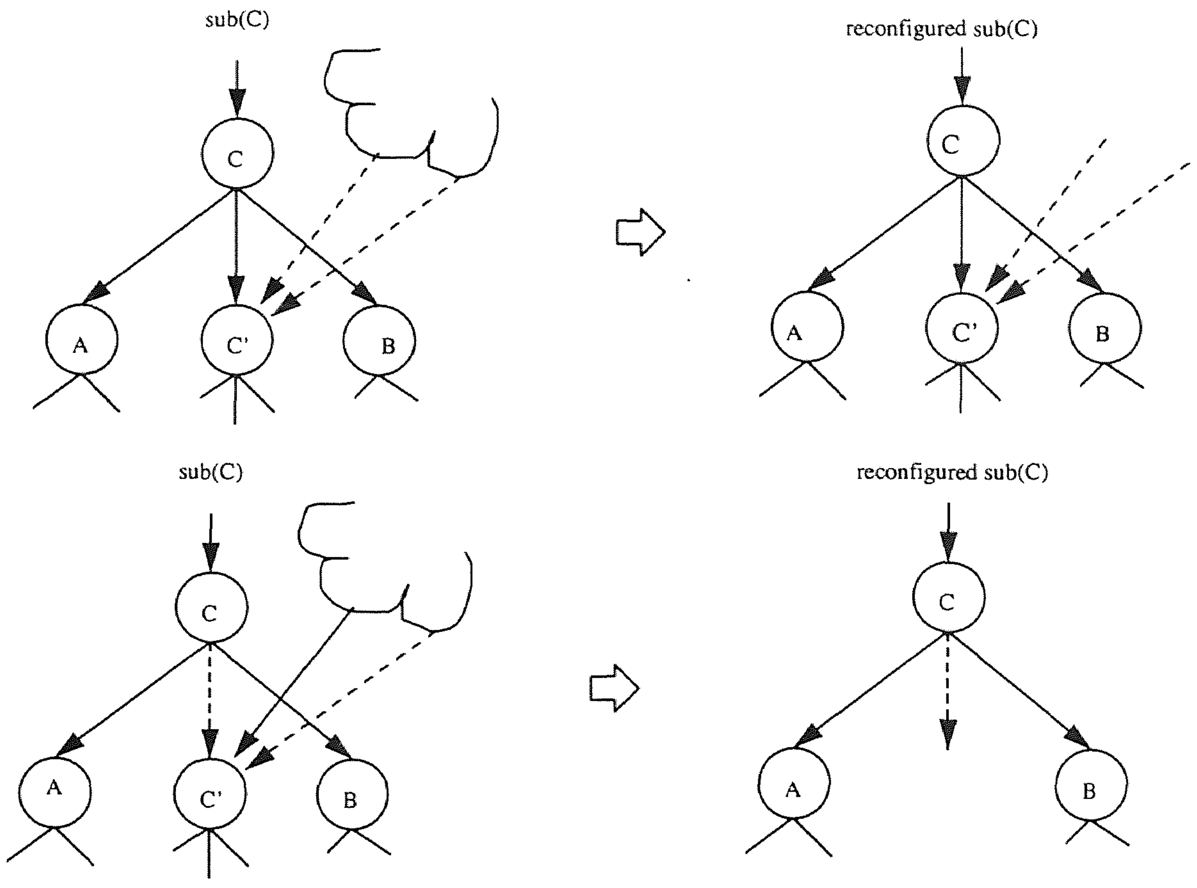


Figure 28 Reconfigure Sub(C)

Log'[P'(F)] - Remove and put into R, all atomic predicates associated with folders in sub(C). Suppose Log"[P'(F)] is the resulting structure after this step.

Sup'(F) - Remove the sub(C). Suppose Sup"(F) is the resulting

organization after this step.

Step 8": If however, L(C) $\in$ Log'[P'(F)], the following reductions take place:

Log'[P'(F)] - Remove the atomic predicate associated with LCPF and let

Log"[P'(F)] be the resulting structure after this step.

Sup'(F) - Remove LCPF and its associated links. Let Sup"(F) be the resulting

organization after this step.

Repeat the same procedure (Step 8) on the descendant folders of C (C is now considered the LCPF or the rooted folder of the split branch from the sup"(F)). The procedure repeats with the two reduced structures Log"[P'(F)] and Sup"(F) until all folders under C are visited or the process cannot go any further.

Step 9: Repeat (Step 8) the same procedure on the next child folder of LCPF until all of its child folders are visited or the process cannot go any further.

Step 10: Merge the two remaining structures and restore the local predicate, L'(F), from R to reflect how the local predicate of F, L(F), is reduced and how the folder F is re-attached to the right folders remained in the Sup"(F).

Step 11: If L'(F) is a composite predicate, transformed it into an equivalent Partial Folder Organization (PFO) which comprises of folders, dummy folders and folder relationships.

Each folder in the PFO is associated with an atomic predicate and logical operators appear in the composite predicate are represented by dummy folders, and proper subfolder or virtual-folder relationships. After replacing the folder F with PFO in the original folder organization, the transformation process for the given folder F and its associated local predicate L(F) is completed.

We will use the following examples to illustrate the above transformation procedure. As shown in Figure 29, given a folder organization and a target folder F in it, assuming that the number appearing in a folder represents the identification of this folder's atomic predicate. So "1" is the atomic predicate assigned to the rooted folder, "2" is the atomic predicate assigned to the left child of the rooted folder, and so on. Suppose $L(F)=[7\wedge(8\vee9)]\wedge2\wedge1\wedge(3\vee4)$ is the user supplied local predicate for the folder F. The global predicate of F, P(F), is first generated (Step 1), that is $[7\wedge(8\vee9)]\wedge2\wedge1\wedge(3\vee4)\wedge1\wedge(2\vee3)$. The transformation domain of P(F) is the folder organization itself since the Sup(F) equals to the original folder organization in this example (Step 2) and it is shown in the figure. In the Step 3, we remove the irrelevant predicates 7, 8 and 9 since none of them is associated with any folders in the Sup(F). So the remaining global predicate, P'(F), becomes $2\wedge1\wedge(3\vee4)\wedge1\wedge(2\vee3)$. Step 4 identifies that the rooted folder of the Sup(F) is the LCPF of the original folder organization. Some logical operational rules are applied to the P'(F) in Step 5 to make P'(F) the simplest logical expression. For instance, there are two "1"s in the P'(F), so we remove one of them. In addition, since "2" is a must condition in the logical tree, so we remove the entire branch $(2\vee3)$ from the predicate structure (see Step 5 in Figure 29). Step 6 starts the step-locked transformation procedure with structures as presented in the left most displays in Figure 29 (before Step 7 is applied). After disconnect folder F from the Sup(F), Step 7 can be skipped since the LCPF is the rooted folder itself. The two structures iterate three times within the Step 8 to reduce their structures, and represent their final versions with Sup"(F) and Log"[P'(F)] respectively. Step 9 is skipped since we assume F is the only folder in the given folder organization remains untransformed. In Step 10, the two reduced structure are merged to show that the simplified predicate of folder F, L'(F), is reduced to $7\wedge(8\vee9)$, and the folder F should be connected to the folder (predicate identity "2") with subfolder link, to the folder (predicate identity "3") with virtual-folder link and to the folder (predicate identity "4") with virtual-folder link as well. Since L'(F) is a composite predicate, we need to further transform it into a PFO. As shown in Figure 29, the PFO of F comprises five new folders including two dummy folders (labeled with "T") so each of them is associated with an atomic predicate (Step 11). The transformation process for the folder F is completed at this point.

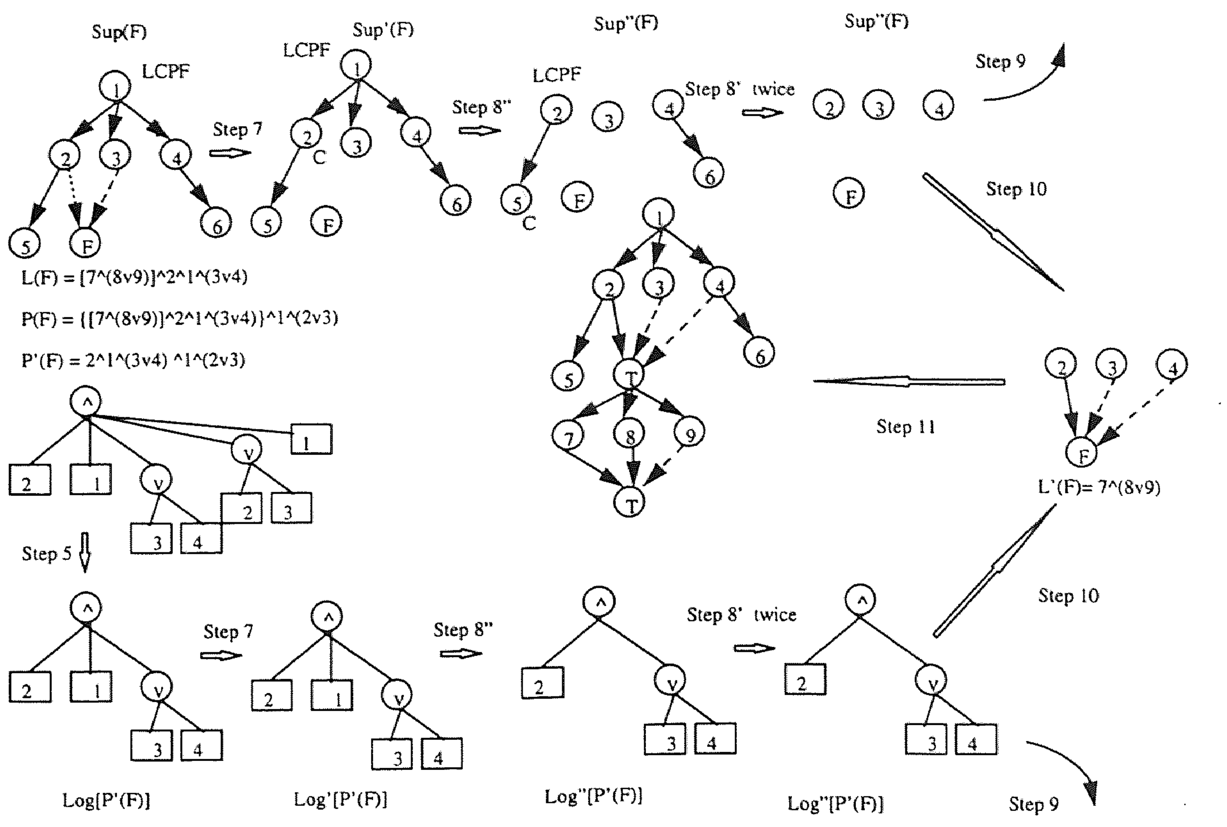


Figure 29 Step-Locked Transformation Process

Figure 30 presents another example where the local predicate of folder F, L(F), is given by 7∧4∧6∧1. Going through the transformation process, the local predicate of folder F can be simplified to L'(F)=7 and F is connected to, with appropriate types of links, to the three folders whose predicate identities are 2, 3 and 6, respectively. Since L'(F) is an atomic predicate, meaning that the output Sup"(F) structure is the PFO of F. So we can direct attach it to the original folder organization without going through the last transformation step.

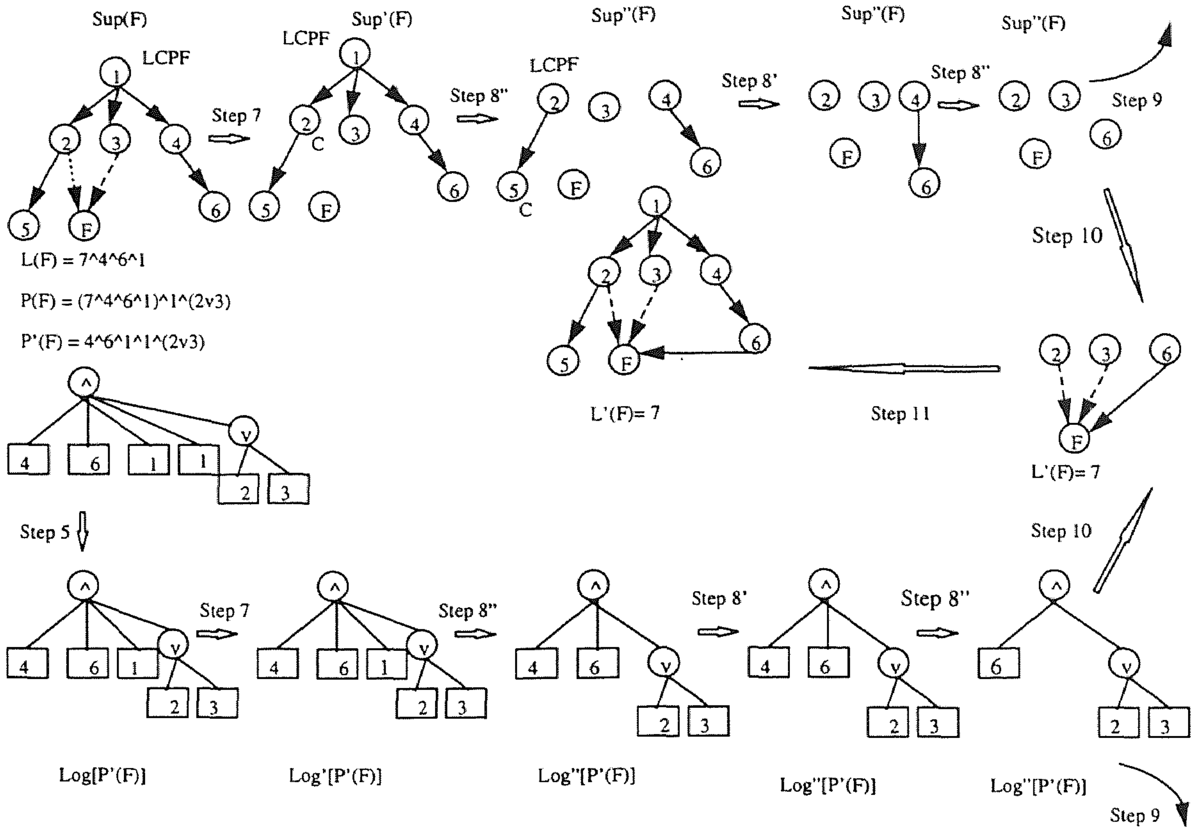


Figure 30 Step-Locked Transformation Process - Another Example

CHAPTER 11

CONCLUSION

In the modern office environment, an intelligent personal filing system is designed to electronically mimic the folder organization and document filing and retrieving methodology a person might use to manage daily documentation. This dissertation presents an architectural enhancement to the existing folder organization model [99,100,103,104], and three different filing approaches based on the enhanced model.

In [99,100,103,104], an agent based filing organization is used, in which each folder associates with a local predicate which governs the document filing for the folder.

The existing folder organization model was originally proposed by the TEXt PROcessing System (TEXPROS) research project [48,91]. It is represented by a Rooted Direct Acyclic Graph (RDAG) [61,99,101], in which each node represents a folder and each link represents a parent-child folder relationship. Each folder is created with a predicate specification designed by the user which governs the content filing from all of its parent folders into this folder. In other words, if the information extracted from a document in a parent folder qualifies the child folder's predicate, the filing system will deposit a copy of this document into the child folder. Otherwise, the document stops at the parent folder and will not go any further beyond that folder. Therefore, a link in the existing folder organization model describes only a possible filing path relationship between the two connecting folders. This relationship is simple and direct. A folder organization constructed with this type of link is perceptible by the user and therefore is referred to as User ORGanization (U-ORG) for simplicity. However, this model also presents some architectural constraints such as it is unable to model a more sophisticated folder organization (with special user requirements) without introducing predicate complexity.

For example, if a user wants to add a new folder to a given folder organization as a depository of common documents from some selected parent folders, this requires the user to specify carefully the filing predicate for this new folder so that the subsequent filing and retrieving can be performed correctly. More precisely, modeling such a simple folder relationship within the existing framework requires the user to provide a condition-equivalent filing predicate for the folder, because the predicate specification in this case needs to take into account all of the related ancestral folders' predicates. This creates the design complexity and has a significant impact on system performance in both reorganization and document filing.

Our proposal supplements the only filing path relationship in the U-ORG model with a subfolder relation link which captures the "and" relationship among links connecting different parent folders to the same child folder. This new organizational model supplements the architectural deficiency of the U-ORG model because:

(1) it is able to model the folder relationship in the previous example without creating unnecessary complexity,

(2) it helps the user to simplify predicate specification inputs, and

(3) it fosters the creation of different filing approaches in two innovative models - sequential and parallel, to further improve the document filing performance based on this new model foundation.

The new folder organization model expresses its superior architectural power by hiding or simplifying the complexity of predicate specification and also helps improve document filing and retrieving performance.

In contrast to the existing folder organization which is simple but limited in modeling capabilities, the proposed new model can be implemented as an Internal representation of a folder ORGanization (I-ORG) for taking its architectural advantages to improve user input specifications and system output performance. In doing so, a two-layer representational hierarchy is proposed. This representational hierarchy, tightly coupling a U-ORG and its underlying support I-ORG, constructs a common system foundation for processing, filing, managing, and retrieving a user's documentation in the electronic office environment.

The design of the filing under the new organization model should capture the architectural attributes of the I-ORG model in order to gain some performance advantages. As an example, the relational constraint, is an architectural attribute presented in the I-ORG model. A constraint can be that certain parent folders must contain the document before this document can be filed against their common child folder. It is checked prior to the time-consuming predicate evaluation task is taken place.

In doing so, filing performance is improved in comparison with the performance in the U-ORG environment.

Another example is as follows. Since the I-ORG representation greatly reduces the complexity of predicate specification by representing a good portion of the complicated predicate with proper folder relationships, it facilitates the system to reduce the number of atomic predicates to be evaluated during the filing, and therefore, it improves the filing performance.

To further support the architectural expressive power of the new organizational model, we present three filing models in this dissertation. First of all, we propose a "complete-parent-first" filing algorithm which employs the well-known topological sorting and its underlying support Depth-First Search (DFS) technique to accomplish the filing tasks. This filing algorithm files the frame instances in an orderly sequential manner. The processing characteristics and filing performance of this filing model are analyzed and evaluated. The other two filing models, the "status broadcast model", and the "result relay model", are proposed to complete the filing in a parallel fashion. The significance of these two filing models are twofold: firstly, the document distribution is more dynamic relying on the message (filing status and filing result) exchanges among folders (the filing order is therefore less predictable); secondly, it fits perfectly into our overall object-oriented design architecture and implementation which paves the road toward our eventual goal in complying with the main development stream as well as gaining notable performance savings in the future distributed processing environment.

Before giving our conclusion that the new folder organization model possesses superior architectural expressive power over the existing model, a generalized filing scenario and a reorganization activity were discussed for each organization model to explore the potential savings gained in the predicate evaluation, the predicate change, and the document redistribution processes.

With the proposed architectural enhancement and the associated filing models, we establish a concrete infrastructure for other components in the TEXPROS system. When an U-ORG is proposed by the user, it is transformed into a content-equivalent folder organization represented by an I-ORG. And furthermore, the system fine-tunes this initial I-ORG into another I-ORG representation, which not only corrects certain mistakes the user might introduce, but also optimizes the overall predicate structure to help improve the filing performance. In such a final representation, each folder in the organization only associates with a single atomic predicate.

CHAPTER 12

FUTURE RESEARCH

In attempt to establish a solid organizational infrastructure for the construction, filing, reorganization, and retrieval subsystem of TEXPROS, we have observed some interesting issues, which are worth further investigation. A brief description of each of the observations is given as follows:

(1) The three filing models as presented in this dissertation have their own approach to traversing through the folder organization and determining the filing orders. But none of them has done enough exploration in the area of predicate evaluation. The predicate evaluation process, as we discussed in this dissertation, sounds simple and straightforward, but it is based on the current defined predicate structure. In either cases, if we decide to rely on this structure or if a new predicate structure proposal is well-accepted in the future, there is always a need to develop a sound evaluation algorithm to furnish the completeness of those three filing models.

(2) An interesting question arises: once a personal folder organization is constructed, can a user deposit a given frame instance into any of his desired folders by simply pointing and clicking at the targeted folder (through a well-defined user interface) instead of always filing from the top (rooted) folder? One reason for doing this is due to the user's perception of his own folder organization. For an incoming frame instance, a user may request to deposit directly a copy of this frame instance into a selected folder. If the deposition can only succeed if the filing information carried in this frame instance

satisfies the target folder's global predicate, then the remaining question is how the system brings the folder organization back to a consistent state. We may argue that, internally, the system can still perform the filing from the rooted folder, since this operation is totally hidden from the user. The system can have many ways to redistribute frame instances. But the real issue here is what happens if the filing information carried in the frame instance does not qualify the global predicate of the desired folder?

(3) If the document classes of a folder organization are disjointed at every level (predicates of folders from the same parent folder are mutually exclusive - for example, male student folder and female student folder), is this kind of folder organization architecturally simpler than a non-disjoint one and possesses with some performance advantages?

(4) If a frame instance contains insufficient information for filing (or some key information is missing), should we provide some kind of facility, that allows us to add or modify the content of a given frame instance, or to attach some hints for going through the filing process and remove them afterward?

(5) The validation process must be enforced before a user's folder organization is actually created. The validation capabilities are twofold: firstly, the system should examine the folder relationships which physically construct a valid organization; secondly, the system should check the predicates whether they are free of redundancy and contradiction.

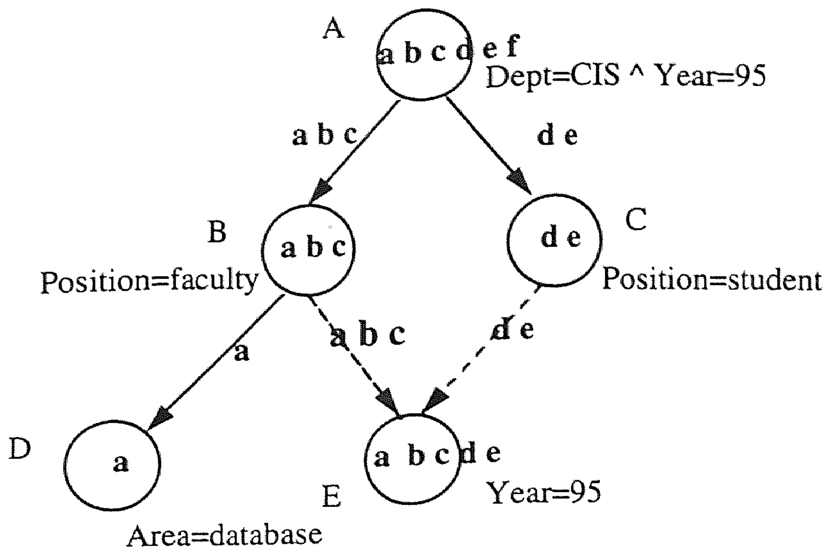
It is simple to validate the physical construction because we only need to check whether a user-defined folder organization has a single rooted folder and is acyclic. This is due to the definition of RDAG. In addition, we need to ensure that the imposed constraints in defining folder relationships are also met (e.g., a folder should not have a single incoming virtual-folder link defined for it).

For validating predicates whether they are free of redundancy and contradiction, it is necessary to validate thoroughly all of the user-defined local predicates. We use the following three examples to explain predicate definition problems. In example 1, assume that the folder B is a subfolder of the folder A. The A folder's local predicate is defined as dept=CIS $\wedge$ position=chairman, while the B folder's predicate is defined as dept=CIS. Apparently, the predicate of folder B is redundant with respect to the predicate of folder A. In example 2, let the predicate of A folder remain the same, but the predicate of B folder be position=student. Now it is easy to see that the two predicates are contradicted each other unless double-positioning frame instances are allowed to entering to the system. In example 3, let the predicate of A folder be salary > 40K and the predicate of B folder be salary > 20K. Again, the predicate of B folder is a redundant predicate with respect to the predicate of A folder. If any of the above cases is created by the user during the construction phase of a folder organization, the system needs to notify the user that the created predicates may introduce some problems to the system and offers an opportunity to do modifications before the actual construction of this folder organization is committed.

Note that only predicates defined with the same or synonymous attributes can be used to detect redundancy or contradictions. Similar attributes defined with different data types are not allowed to be compared unless the required criteria is supported by the knowledge base. For example, if the predicate of folder A is defined as Salary > 40K, while the predicate for folder B is defined as salary = HIGH, how does the system perform a comparison between the two predicates and determine if they are free of the redundancy and/or contradiction? This is a difficult problem.

Next, we formally define rules for detecting redundant and contradictory predicates. Let P and Q be two related local predicates. Let Q's folder be a descendent folder of P's. If P implies Q, then we say Q is redundant with respect to the predicate P. For example, in Figure 31, the folder E is redundant since the local predicate of A implies the local predicate of E and the content of the folder B (E's immediate parent) is duplicated in the folder E. Similarly, the content of the folder C (another E's immediate parent) is also duplicated in the folder E. We should remove the redundant folders from the folder organization to save the unnecessary operation and management cost.

On the other hand, let Q's folder be a descendent folder of P's. Two related local predicates P and Q are said to be conflicting predicates if P AND Q is always false. For example, in Figure 32, folder A and folder E have contradictory predicates since (L(A) AND L(E)) = False, and folder E is always empty. We should also remove the contradictory folders from the folder organization to save the unnecessary operation and management cost.



L(X): local predicate of folder X

G(X): global predicate of folder X

L(A)/P(A): (Dept=CIS ^ Year=95)

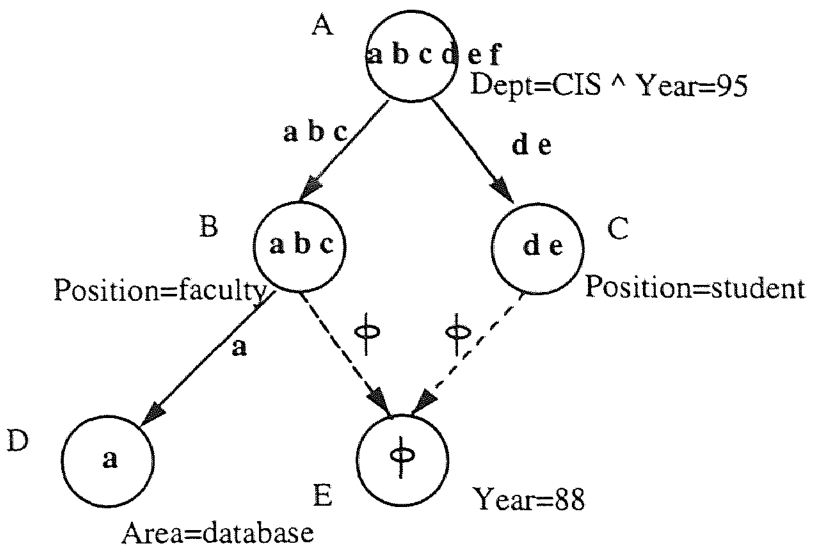
L(E): Year=95

G(E): (Year=95) ^ (Dept=CIS) ^ ((Position=faculty) v (Position=student))

$$L(A) \xrightarrow{\text{implies}} L(E)$$

Figure 31 Redundant Folders

Note that both redundant and contradictory problems can be tolerated by the system because their existence only wastes system resources (such as memory and CPU), and does not cause any operational problems. But we can carefully guide the user during the design phase (constructing or modifying a folder organization) to avoid creating these kinds of problems. However, before a definition is presented for clarifying the nature and the characteristics of these kinds of design problems [79] and before a proven technique can be applied, these problems exist commonly in the normal design but may be hidden from the user as his folder organization is created.



L(X): local predicate of folder X
G(X): global predicate of folder X

$L(A) \rightarrow\!\times\!\leftarrow L(E)$

L(A)/P(A): (Dept=CIS ^ Year=95)
L(E): Year=88
G(E): (Year=88) ^ (Dept=CIS) ^ (Year=95) ^ ((Position=faculty) v (Position=student))

Figure 32 Conflict Folders

Furthermore, the validation process, as described above, is considered a passive approach which helps the user correct problems after predicates are specified. Alternatively, a proactive approach may be considered, which guides the user throughout the predicate specification process and immediately corrects or revises any inefficient factors the user introduced, like those problems presented above.

Note that the capability of validating input predicates and folder relationships is also required during the folder reorganization process. The only difference is that, instead of getting all of the input information from the user, the validation process is now based on the existing and already validated structure and a restructure proposal that requests an reorganization change. The request for reorganization cannot be committed before the proposal is granted by the system (passing similar validation procedures).

(6) Our current research is focused on creating a personal folder organization, a centralized, predicate governed, and self-contained personal filing system that is represented by a RDAG. To cover larger operating environments such as a department, an electronic library [84,96], a corporation, etc., an extended personal model may be needed to model a folder organization in such an environment. Each personal folder organization is defined by its owner as a private folder system that can only be accessed by its owner. On the other hand, sharable folders can be designed which are either open to the public or restricted to certain user group(s). As an example, we can create a departmental, an enterprise or a global information system which combines and links many private folder organizations with some public folder organizations together. A folder organization defined and operated in such an environment is partially shared by multiple users, and can be physically distributed to different locations. Such a folder organization can be represented by a multiple rooted DAG. The major differences between a personal model and an extended personal model, are in their filing and folder reorganization. In filing, a personal model always has one single source (the rooted folder) to begin with, where the extended model can have multiple sources to do the filing. The extended personal model also allows a filing system of person A's to receive frame instances that are deposited from the filing system owned by person B. Moreover, the performance advantage of the parallel filing model may only be realized when the model is implemented in an open distributed environment. In such a computing environment, everything is networked and

object-based. As an example, we will define roles and interactions among objects like **folder**, **link**, **document**, **component** and **agent** in our distributed folder organization model. The **folder** objects do not necessary reside in a central location or on a single platform; they can be distributed anywhere in the network and configured as public-accessible or private-owned folders. **Link** objects for defining relationships between folders are maintained on a bilateral basis. Each link in the organization provides only a connection interface between two related folders. These two folders are the only objects in the network that know about the link. **Document** objects presented with different forms may be inserted into the filing system from different rooted folders as entry points. They could be text-only documents, or multimedia documents consisting of different components such as text, voice, picture or video. Each of these components is treated as an individual **component** object. How a document is decomposed into components and stored in the networked repositories, and how different components are retrieved, filtered and synthesized to meet a user's inquiry interests are on-going research topics. **Agent** objects of folders in our folder organization model can be enhanced to become active agents. Active agents may be empowered with mobile capabilities that enable them to move around and interact with each other or with functional entities of the system to proactively searching for ways to accomplish their assignments (e.g., filing tasks) more efficiently and effectively. The implementation of such a multi-user support system, is expected to operate on a resource-shared distributed processing environment. We believe the parallel filing model as proposed in this paper, perfectly fits in an object-based, open and distributed architecture, which is consistent with the evolving trend of data processing and provides opportunities to meet the performance needs.

REFERENCES

1. Adeli, *Knowledge Engineering*, McGraw-Hill, NY, 1990.

2. Aho, J. Hopcroft and J. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Menlo Park, CA, 1987.

3. 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.

4. G. Booch, *Object Oriented Design with Applications*, the Benjiamin/Cummings Publishing Company, Inc., Redwood City, CA, 1991.

5. G. Bracchi, B. Pernici, "The Design Requirements of Office Systems", *ACM Transactions on Office Information Systems*, Vol. 2, No.2, April 1984, pp151-170.

6. A. Celentano, M.G. Fugini and S. Pozzi, "Knowledge-Base Document Retrieval in Office Environments: The Kabiria System", *in Proc. of the 14th Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval, Chicago, IL*, pp. 183-189, October 1991.

7. A. Celentano, M.G. Fugini and S. Pozzi, "Querying Office Systems about Document Roles", *ACM Transactions on Office Information Systems*, Vol. 13, No. 3, pp. 237-268, July 1995.

8. L. Chen, J.T.L. Wang and P.A. Ng, "A Knowledge Based System for Intelligent Document Processing", draft, Dec. 1993.

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

10. S. Chen, F. Shih and P. Ng, "A Fuzzy Model for Unsupervised Character Classification", *Information Sciences, An International Journal*, Vol. 2, No. 2, 1994.

11. S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa and A. Pathria, "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and System", *ACM Transactions on Office Information Systems*, Vol. 4, No. 4, pp. 345-383, October 1986.

12. H. Clifton, H. Garcia-Molina and R. Hagmann, "The Design of a Document Database", *in Proceeding of the ACM Conference on Document Processing Systems*, pp. 125-134, December, 1988.

13. W.F. Clocksin and C.S. Mellish, *Programming in Prolog*, Spring-Verlag, NY, 1981.

14. P.R. Cohen and R. Kjeldsen, "Information Retrieval by Constrained Spreading Activation in Semantic Networks", *Information Processing and Management*, Vol. 23, No. 4, pp. 255-268, 1987.

15. W.B. Croft, NSF Center for Intelligent Information Retrieval, *Communications of the ACM*, Vol. 38, No. 4, pp. 42-43, April 1995.

16. W.B. Croft and R. Krovetz, "Interactive Retrieval of Office Documents", *in proc. ACM-IEEE Conference on Office Information Systems*, pp. 228-235, NY.

17. W. Croft and D. Stemple, "Supporting Office Document Architectures with Constrained Types", *in Proc. Of ACM SIGMOD International Conference on Management of Data*, pp. 504-509, 1987.

18. T.H. Cormen, C.E. Leiserson and R.L. Rivest, "Introduction to Algorithms", *The MIT Press*, (McGraw-Hill, NYC, 1991).

19. P. Dadam and V. Linnemann, "Advanced Information Management (AIM): Advanced database technology for integrated applications", *IBM Systems Journal*, Vol. 28, No. 4, pp. 661-681, 1989.

20. P.J. Denning, Electronic Junk, *Communications of the ACM*, Vol. 25, No. 3, pp. 163-165, March 1982.

21. S. Doong, el., "A Major Enhancement to the Folder Organization Model for Office Information Systems", draft paper, May, 1996.

22. P. Fischer and S. Thomas, "Operators for Non-First Normal Form Relations", *in Proceeding of the 7th International Computer Software Applications Conference*, pp. 464-475, 1983.

23. E.A. Fox, R.M. Akscyn, R.K. Furuta and J.J. Leggett, "Digital Libraries - Introduction", *Communications of the ACM*, Vol. 18, No. 4, pp. 22-29, April 1995.

24. 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 1986.

25. A. Ginsberg, "A Unified Approach to Automatic Indexing and Information Retrieval", *IEEE Expert*, pp. 46-56, October 1993.

26. U. Gupta, "Validating and Verifying Knowledge-Based Systems",*IEEE Computer Society*, Los Alamitos, CA, 1991.

27. R. Guting, R. Zicari and D. Choy, "An Algebra for Structured Office Documents", *ACM Transactions on Office Information Systems*, Vol. 7, No. 4, pp. 123-157, April 1989.

28. 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.

29. X. Hao, J.T.L. Wang and P.A. Ng, "Nested Segmentation: An approach for Layout Analysis in Document Classification", *in proc. Of the Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, pp. 319-322, October 1993.

30. X. Hao, J.T.L. Wang, M.P. Bieber, P.A. Ng, "A Tool for Classifying Office Documents", *Proc. of the 5th International Conference on Tools with Artificial Intelligence*, Boston, MA, Nov. 1993.

31. X. Hao, J.T.L. Wang, M.P. Bieber, P.A. Ng, "Heuristic Classification of Office Document Classifications", *International Journal on Artificial Intelligence Tools*, Vol. 3, No. 2, pp. 233-265, 1994.

32. X. Hao, J.T.L. Wang, P.A. Ng, "Information Extraction from Structured Part of Office Documents", draft. 1995.

33. P. Hoepner, "Synchronizing the Presentation of Multimedia Objects - ODA Extensions", *ACM SIGOIS Bulletin*, Vol. 12, No. 1, pp. 19-32, July 1991.

34. W. Horak, "Office Document Architecture and Office Document Interchange Format - A Current Status of International Standardization", *IEEE Computer*, Vol. 18, No. 10, pp. 50-60, October 1985.

35. J. Hughes, *Object Oriented Database*, Prentice Hall, NY, 1990.

36. R. Hunter, P. Kaijser and F. Nielsen, "ODA: A Document Architecture for Open Systems", *Computer Communication*, Vol. 12, No. 2, pp. 69-79, April 1989.

37. C. Huser, K. Reichenberger, L. Rostek and N. Streitz, "Knowledge-based Editing and Visualization for Hypermedia Encyclopedias", *Communications of the ACM*, Vol. 18, No. 4, pp. 49-51, April 1995.

38. E. Ide and G. Salton, "Interactive Search Strategies and Dynamic File Organization in Information Retrieval", *in G. Salton, editor, The Smart Retrieval System - Experiments in Automatic Document Processing*, pp. 373-393, Prentice-Hall Inc., 1971.

39. G. Jaeschke and H. Schek, "Remarks on the Algebra of Non-First Normal Form Relations", *in Proc. Of the ACM SIGACT-SIGMOD Symposium on PODS*, pp.124-138, 1982.

40. N. Jardine and C.J. van Rijsbergen, "The Use of Hierarchic Clustering in Information Retrieval", *Information Storage and Retrieval*, Vol. 7, No. 5, pp. 217-240, December 1971.

41. W. Kim and F. Lochousky, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publishing Company, NY, 1989.

42. D.E. Knuth, J.H. Morris and V.R. Pratt, "Fast Pattern Matching in Strings", *SIAM Journal of Computing*, Vol. 6, No. 2, pp. 323-350, June 1977.

43. D.L. Lee, Efficient Signature File Methods for Text retrieval,*IEEE Transactions on Knowledge and Data Engineering*,, Vol. 7, No. 3, pp. 436, 1995.

44. Q. Liu, *An Office Document Retrieval 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, NJ, 1994.

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

46. Q. Liu and P. 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.

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

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

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

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

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

52. E. Lutz, H.V. Kleist-Retzow and K. Hoernig, "Mafia - An Active Mial-filter-agent for an Intelligent Document Processing Support", *Multi-User Interfaces and Applications*, pp.16-32, 1990.

53. D. Maier, *The Theory of Relational Database*, Computer Science Press, Potomac, MD, 1983.

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

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

56. B.P. McCune and et al. Rubric: A System for Rule-based Information Retrieval, *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 9, pp. 939-945, September 1985.

57. J. McHugh, *Algorithmic Graph Theory*, Prentice Hall, New Jersey, 1990.

58. C.T Meadow, *Test Information Retrieval Systems*, Academic Press, San Diego, CA, 1992.

59. 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.

60. F. Mhlanga, J. Wang, T. Shiau and P. Ng, "A Query Algebra for Office Documents", *in Proc. Of the 2nd International Conference on Systems Integration*, Morristown, NJ, pp. 458-467, June 1992.

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

62. N. Naffah and A. Karmouch, "Agora - An Experiment in Multimedia Message Systems", *Computer*, Vol. 19, No. 5, pp. 56-66, May 1986.

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

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

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

66. R. Rao, J.O. Pedersen, M.A. Hearst, J.D. Mackinlay, S.K. Card, L. Masinter, P-K. Halvorsen and G.G. Robertson, "rich Interaction in the Digital Library", *Communications of the ACM*, Vol. 18, No. 4, pp. 25-39, April 1995.

67. Jr. J.J. Rocchio, "Relevance Feedback in Information Retrieval", *in G. Salton, editor, The Smart Retrieval System - Experiments in Automatic Document Processing*, pp. 313-323, Prentice-Hall Inc., 1971.

68. S. Sakata and T. Ueda, "A Distributed Office Mail System", *Computer*, Vol. 18, No. 10, pp. 106-116, October 1985.

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

70. 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.

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

72. S. Saxin, "Computer-based Real-time Conferencing System", *Computer*, Vol. 17, No. 10, pp. 33-35, October 1985.

73. D. Shasha and J. Wang, "Optimizing Equijoin Queries in Distributed Databases Where Relations are Hash Partitioned", *ACM Transactions on Database Systems*, Vol. 16, No. 2, pp. 279-308, June 1991.

74. G. Shaw and S. Zdonik, "A Query Algebra for Object-Oriented Databases", *in Proceedings of the 6th International Conference on Data Engineering*, Los Angeles, CA, pp. 154-162, February 1990.

75. F.Y. Shih, S. Chen, D.C.D. Hung and P.A. Ng, "A Document Segmentation, Classification and Recognition System", *in Proceedings of 2nd International Conference on Systems Integration*, Morristown, NJ, pp. 258-267, June 1992.

76. M. Snoeck and G. Dedene, "Generalization/Specification and Role in Object Oriented Conceptual Modeling", *Data and Knowledge Engineering*, Vol. 19, No. 2, pp. 171-195, June 1996.

77. S. Su, M. Gou and H. Lam, "Association Algebra: A Mathematical Foundation for Object-Oriented Databases", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 5, pp. 775-798, October 1993.

78. Y.Y. Tang, C.D. Yan and C.Y. Suen, "Document Processing for Automatic Knowledge Acquisition", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 1, pp. 3-21, 1994.

79. C. Thanos, *Multimedia Office Filing: The MULTOS Approach*, Elsevier Science Publishers B. V., 1993, Chapter 3: The MULTOS Document Model.

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

81. D. Tsichritzis, *Office Automation*, Springer-Verlag, Berlin, 1985.

82. D. Tsichritzis, S. Christodoulakis, P. Econopoulos, C. Faloutsos, A. Lee, D. Lee, K. Vanderbroek and C. Woo, "A Multimedia Office Filing System", *in Proc. Of 9th International Conference On Very Large Databases*, 1983.

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

84. 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.

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

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

87. E.M. Voorhees, "The Cluster Hypothesis Revised", *in Proceedings of the 8th Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pp. 188-196, NY, June 1985.

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

89. J.T.L. Wang, F.S. Mhlanga, Q. Liu, W.C. Shang, P.A. Ng, "An Intelligent Documentation Support Environment", *Proc. of 5th International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, pp429-436, June 1993.

90. J. Wang, F. Mhlanga and P. Ng, "A New Approach to Modeling Office Documents", *ACM SIGOIS Bulletin*, Vol. 14, No. 2, pp. 46-55, December 1993.

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

92. C. Wei, *Knowledge Discovery 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.

93. C. Wei, Q. Liu, J. Wang and P. Ng, "Knowledge Discovery for Document Classification Using Tree Matching in TEXPROS", *Submitted to Information* Sciences, an International Journal, March 1996.

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

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

96. C. Winkler, "Desktop Publishing", *Datamation*, Vol. 32, No. 23, pp. 92-96, December 1986.

97. R.J. Wirfs-Brock and R.E. Johnson, "Surveying Current Research in Object-Oriented Design", *Communications of the ACM*, Vol. 33, No. 9, pp. 104-124, September 1990.

98. D. Woelk, W. Kim and W. Luther, "An Object-Oriented Approach to Multimedia Databases", *in Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington D.C., pp. 311-325, May 1986.

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

100. Z. Zhu, Q. Liu, J. Mchugh, and P. Ng, "A Predicate Driven Document Filing System", *Journal of Systems Integration*, Vol. 6, No. 3, pp. 373-403, September 1996.

101. Z. Zhu, J.A. McHugh, J.T.L. Wang, P.A. Ng, "A Formal Approach to Modeling Office Information Systems", *Journal of Systems Integration*, Vol. 4, No. 4, pp. 373-403, December 1994.

102. J. Zobel, J.A. Thom and R. Sacks-Davis, "Efficiency of Nested Relational Document Database Systems", *in Proc. Of the 17th International Conference on Very Large Databases, Barcelona*, Spain, pp. 91-102, September 1991.

103. Fan, X., Q.H. Liu and P.A. Ng, "A Multimedia Document Filing System", *Proceedings on IEEE International Conference on Multimedia Computing and Systems*, Ottawa, Canada, pp. 492-499, June 3-6, 1997.

104. Fan, X., Q.H. Liu and P.A. Ng, "Knowledge-Based Document Filing: TEXPROS Approach", *Proceeding on the 13th International Conference on Advanced Science and Technology*, in conjunction with the 2nd International Conference on Multimedia Information Systems and IEEE/CS YUFORIC on Multimedia Information Systems - Chicago, (ICAST97/ICMIS97), Motorola University, Schaumburg, IL, pp. 58-67, April 3-5, 1997.

105. Simon Doong, C.S. Wei, Xien Fan, D.C. Hung and Peter A. Ng, "A Folder Organization Model in the Office Environment", *submitted to the 4th International Conference on Information Systems Analysis and Synthesis*, Orlando, Florida, July 12-16, 1998.

106. Simon Doong, Xien Fan, C.S. Wei and Peter A. Ng, "A Process for Constructing a Personal Folder Organization", *submitted to the 4th International Workshop on Multimedia Database Management Systems*, Dayton, Ohio, June, 1998.

107. Simon Doong, Ron Curtis and Peter A. Ng, "Document Filing Approaches Using an I-ORG Based Folder Organization", *submitted to the 17th International Conference on Conceptual Modeling*, Singapore, November 16-20, 1998.

108. C.Y. Wang, Q. Liu and P.A. Ng, "Intelligent Browser for TEXPROS", *in IASTED proceedings of the International Conference on Intelligent Information Systems*, Grand, Bahama Island, Bahamas, December 8-10, 1997, pp. 388-398, IEEE Computer Society.