Copyright Warning & Restrictions

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

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

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

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

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

Order Number 9401732

D_model and D_algebra: A data model and algebra for office documents

Mhlanga, Fortune Solani, Ph.D.

New Jersey Institute of Technology, 1993

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

$\mathcal{D}$_MODEL AND $\mathcal{D}$_ALGEBRA:
A DATA MODEL AND ALGEBRA FOR OFFICE DOCUMENTS

by

Fortune Solani Mhlanga

A Dissertation

Submitted to the Faculty of

New Jersey Institute of Technology

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Department of Computer and Information Science

May 1993

ABSTRACT

$\mathcal{D}$_Model and $\mathcal{D}$_Algebra: A Data Model and Algebra for Office Documents

by

Fortune Solani Mhlanga

This dissertation presents a data model (called $\mathcal{D}$_model) and an algebra (called $\mathcal{D}$_algebra) for office documents. The data model adopts a very natural view of modeling office documents. Documents are grouped into classes; each class is characterized by a "frame template", which describes the properties (or attributes) for the class of documents. A frame template is instantiated by providing it with values to form a "frame instance" which becomes the *synopsis* of the document of the class associated with the frame template. Different frame instances can be grouped into a folder. Therefore, a folder is a set of frame instances which need not be over the same frame template.

The $\mathcal{D}$_model is a dual model which describes documents using two hierarchies: a document type hierarchy which depicts the structural organization of the documents and a folder organization, which represents the user's real-world document filing system. The document type hierarchy exploits structural commonalities between frame templates. Such a hierarchy helps classify various documents. The folder organization mimics the user's real-world document filing system and provides the user with an intuitively clear view of the filing system. This facilitates document retrieval activities.

The $\mathcal{D}$_algebra includes a family of operators which together comprise the fundamental query language for the $\mathcal{D}$_model. The algebra provides operators that can be applied to folders which contain frame instances of different types. It has more expressive power than the relational algebra. It extends the classical relational algebra by associating attributes with types, and supporting attribute inheritance. Aggregate operators which can be applied to different frame instances in a folder are also provided. The proposed algebra is used as a sound basis to express the semantics of a high level query language for a document processing system, called TEXPROS.

In the model, frame instances can represent incomplete information. Null values of the form *value at present unknown* are used to denote missing information in some fields of the incomplete frame instances. This dissertation provides a proof-theoretic characterization of the data model and defines the semantics of the null values within the proof-theoretic paradigm.

APPROVAL PAGE

$\mathcal{D}$_Model and $\mathcal{D}$_Algebra:
A Data Model and Algebra for Office Documents

Fortune Solani Mhlanga

Dr. Peter A. Ng, Dissertation Co-Advisor (date)
Chairperson and Professor of Computer and Information Science, NJIT

Dr. Jason T. L. Wang, Dissertation Co-Advisor (date)
Assistant Professor of Computer and Information Science, NJIT

Dr. James A. M. McHugh, Committee Member (date)
Associate Chairperson and Professor of Computer and Information Science, NJIT

Dr. Murray Turoff, Committee Member (date)
Professor of Computer and Information Science and Management, NJIT

Dr. Nabil R. Adam, Committee Member (date)
Chairperson and Professor of Computer and Information Science, Rutgers University

BIOGRAPHICAL DATA

Author: Fortune Solani Mhlanga

Degree: Doctor of Philosophy in Computer Science

Date: May 1993

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1993
- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1989
- Bachelor of Science in Computer Science,
  Harding University, Searcy, AR, 1984

Major: Computer Science

Presentations and Publications:

- F. S. Mhlanga, J. T. L. Wang, T. H. Shiau, and P. A. Ng. A Query Algebra for Office Documents. In *Proceedings of the Second International Conference on Systems Integration*, pp. 458–467, Morristown, NJ, June 1992.
- J. T. L. Wang, F. S. Mhlanga, Q. H. Liu, W.-C. Shang, and P. A. Ng. Database Support for Software Documentation: The TEXPROS Project. To appear as a book chapter in *Software Automation and Productivity Improvement*, 1994.
- J. T. L. Wang, F. S. Mhlanga, Q. H. Liu, W.-C. Shang, and P. A. Ng. An Intelligent Documentation Support Environment. To appear in *Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, June 1993.

This dissertation is dedicated to
my brother
Washington Célani Mhlanga

ACKNOWLEDGMENT

The author takes great pleasure in acknowledging his dissertation advisor, Professor Peter A. Ng for his assistance and contribution to this effort. His insights and persistent personal interest in the project from its inception and his constant encouragement were immeasurable. He devoted time and effort reviewing various drafts of the manuscript and provided many helpful comments that influenced the final manuscript. His guidance, friendship, and moral support throughout this research are much appreciated. The author is indebted to his co-advisor, Dr. Jason T. L. Wang who, in his quest for perfection, provided constructive criticism on every chapter. His ability to solicit literature from an eminent array of database professionals contributed significantly to the contents and organization of this dissertation. Unfortunately, Dr. T. H. Shiau had to leave prior to the completion of the research. However, his early contributions to the research and to the drafts of the algebra are still evident. He provided support, encouragement, and crucial feedback during the initial process of the work.

Special thanks to Professors James McHugh, Nabil Adam, and Murray Turoff for serving as members of the committee.

The author is grateful to the University of Zimbabwe, the Ministry of Labor and Manpower of Zimbabwe, the Institute of International Education, and the African-American Institute for their combined effort in the initial funding of his graduate academic endeavors. The research was supported in part by the New Jersey Institute of Technology and by a grant from the AT&T Foundation.

The author wishes to thank colleagues, friends, and fellow PhD students for their moral support. These include Volkan Atalay, Wayne Baker and family, Shy-Shyan Chen, Prof. Christopher Chetsanga, John Chingarire, Kennedy Dzama, Gail Endicott, Michael Halper, Gilford and Sevelyn Hapanyengwi, Dan Harris, Barry and Duster Hatfield, Sharon and David Hicks, Tony Hopper and family, Christine Hubert, Abubakah Ibrahim, Leon Jololian, Rakesh Kushwaha, Ricky and Brenda Lowe, Carl and Debbie McAfee, Pete McCoy and family, Alfred Minango, Joseph Mmbaga, Prof. Erich J. Neuhold, William Ochan, Michail Papamichail, James Owens, Benny Peek, Ajaz Rana, Sunitha Reddy, Temba and Fortune Shoniwa, Rayton and Mary Sianjina, Mehmet Tazebay, Anthoula Trombouki, Devendra Vamathevan, Tamar Zemel, and all the members of the TEXPROS group. Qianhong Liu's contributions to the selection operation of the $\mathcal{D}$-algebra are invaluable and could have otherwise gone unnoticed. The author cannot omit admiration and thanks to Manveen and Apoorva Koticha for all the things they did for him during this strenuous period of research.

The author would like to thank his family and Kundai M. Chagonda for their encouragement. This work took valuable time away from them. Their understanding and patience during this long period of time are much appreciated.

The writing of this dissertation was facilitated by the computing resources and equipment in the CIS department at NJIT.

Finally, the author gratefully acknowledges his debt to the authors of the works that are cited in this dissertation, and claims full responsibility for any bugs that the text may contain.

TABLE OF CONTENTS

Chapter **Page**

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The purpose of Office Information Systems (OISs) is to support office workers in their management of information, and to assist them in their daily work [41]. Often, information circulated in offices is kept in *documents*. Some documents have rigid structures, such as forms [92]; some are text-oriented, such as letters, memos, reports and electronic mails. The documents may also contain graphics, images, audio and video data [93]. Graphics and image data include line graphs and bit maps. Such data can be stored on magnetic disk. Audio and video data are unformatted data and need to be converted into text values [32]. In order to provide greater flexibility, OISs must provide capabilities of editing, formatting, filing, retrieving and processing them [16, 103], and functions of translating audio, image and text data into a common internal representation for processing and storage [32].

1.1 TEXPROS

TEXPROS (TEXt PROcessing System) [103] is a personal, customizable system for processing office documents. The system has functional capabilities of automating (or semi-automating) common office activities such as document classification, extraction, filing and retrieval. To accomplish these goals, the system includes the following components:

- A state-of-the-art data model capable of capturing the behavior of the various office activities.
- Extracting the synopsis or the most significant information from a document (such information is often sufficient to satisfy the user's needs when information retrieval occurs).
- A knowledge-based, customizable document classification handler that exploits both spatial and textual analysis to identify the type of a document.
- An agent-based architecture supporting document filing and file reorganization.
- A retrieval architecture that can handle incomplete and vague queries.
- A hypertext architecture that can provide good information management support for office documents.

This dissertation presents a data model for TEXPROS and a practically useful algebraic language for the retrieval and manipulation of TEXPROS objects.

1.2 Organization of the Thesis

The remainder of this dissertation is organized as follows: Chapter 2 presents the motivation of the dissertation and contains a survey of research which is related to

this work. Chapters 3 through 8 present the research work. Chapter 3 describes an informal view of the data model. The chapter first introduces relevant terminology for TEXPROS that is related to this dissertation and provides a comprehensive example that will be elaborated upon throughout the rest of the work. It then proceeds to informally describe the scenario of the fundamental elements that underlie the formal treatment of the data model. Chapter 4 presents the formalism of the data model. This formal framework of the data model will be used throughout the dissertation. Chapter 5 informally introduces the operations of the $\mathcal{D}$_algebra based on some examples. Chapter 6 formally describes the syntax and semantics of various operators supported by the algebraic language for the data model. The question of the expressive power of the algebra relative to its operation on the objects of the data model is addressed in chapter 7. It is demonstrated that a subset of the $\mathcal{D}$_algebra is more expressive than the relational algebra. Chapter 8 defines the semantics of null values within the proof-theoretic paradigm of the $\mathcal{D}$_model. Finally, chapter 9 summarizes the dissertation and gives a brief discussion on some prospective and ongoing research topics that are based on the work described in this dissertation.

CHAPTER 2

MOTIVATION AND RELATED WORK

Motivated by an activity in the area of document modeling, we present a data model and an algebraic language for a Text Processing System (TEXPROS) [101, 102, 103] for processing office documents. The data model presented adopts a very natural view of modeling office documents by closely resembling the real-world document filing system perceived by the user.

2.1 Data Modeling for Office Documents

Data modeling for document management systems has gained quite a bit of attention. Horak [49], Croft and Stemple [23] represent the structures of documents based on the Office Document Architecture (ODA). ODA is part of the standards for document interchange developed by the International Standardization Organization (ISO) and the European Computer Manufacturers Association (ECMA). It distinguishes between the logical and layout structures of a document. The logical and layout structures are made up of hierarchies of *logical objects* and *layout objects* respectively. The logical and layout objects are classified according to their *type* which is the document *class*. The logical structure associates the content of the document with a hierarchy of logical objects. Examples of logical objects are summaries, titles, sections, paragraphs, figures, tables, and so forth. The layout structure associates the same content with a hierarchy of layout objects. Examples of layout objects are pages, columns,

and footnote areas. ODA requires that each document has a logical structure and a layout structure, together with a set of logical-layout, logical-logical and layout-layout relationships. A simplified ODA document structure and a type hierarchy of ODA objects are depicted in Figures 1 and 2 respectively (Figures 1 and 2 are excerpts from [23]). There is a distinction between *composite* and *basic* logical object types. Composite logical objects comprise other composite logical objects or basic logical objects. Basic logical objects are associated with *content portions* which contain the contents of a document. Included in the layout object types are *page set*, *composite page*, *basic page*, *frame*, and *block*.

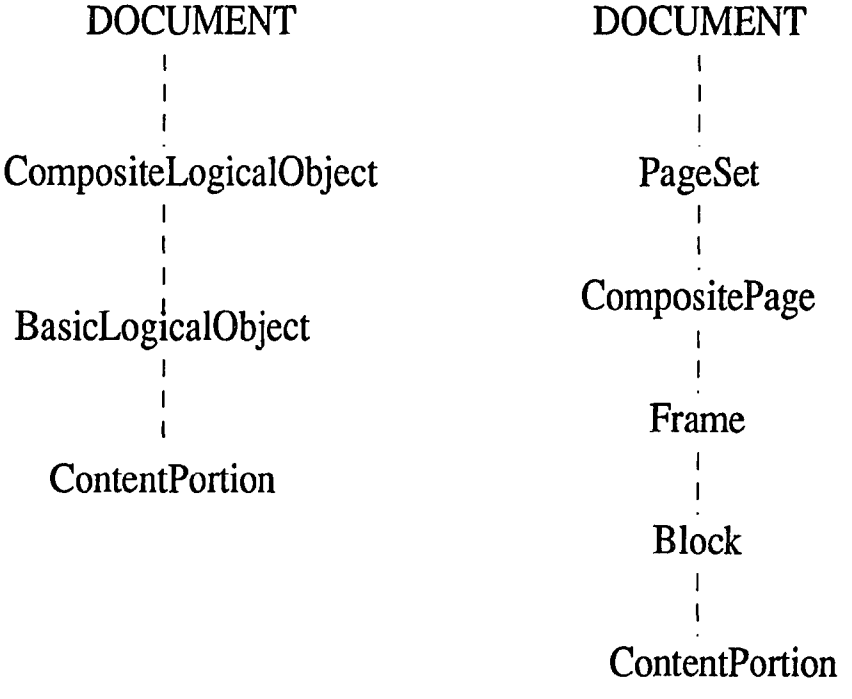


Figure 1: Simplified ODA document structure

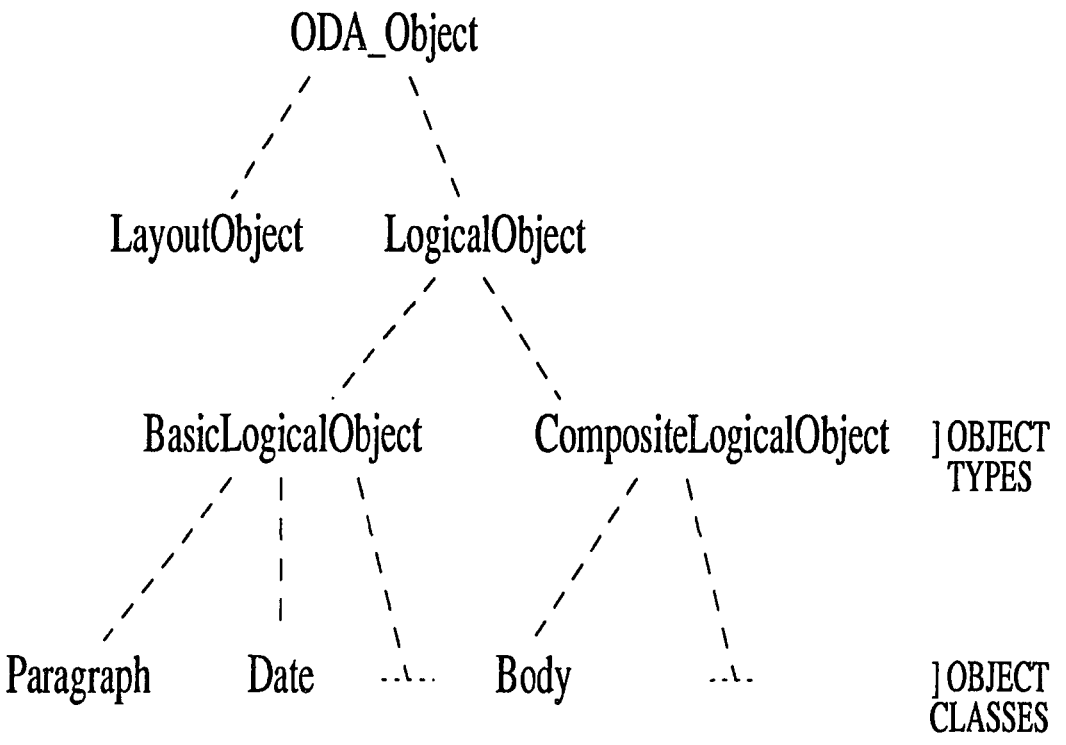


Figure 2: A type hierarchy of ODA objects

Bertino, Rabitti and Gibbs [9] extend[1] the ODA standard by including a conceptual structure, which allows to specify a document in terms of its conceptual component types. A conceptual component type is defined by a set of attributes. It represents a portion of a document used for some specific purpose (e.g., the sender of a memo). Figure 3 shows an example of conceptual structures of document types Generic_Letter and Business_Product_Letter (Figure 3 is an excerpt from [9]). In the figure, the attributes inside the box represent the Generic_Letter document type

[1]This extension is also referred to as the conceptual data model [77].

and those outside the box are included to specify the representation of the Business_Product_Letter document type. The authors argue that component types[2] are more meaningful to the user than the logical and layout components in terms of retrieval where <attribute, value> pairs can be used in specifying queries. This enables the model to support a well-defined query language and techniques for query processing. Bertino et al. describe a distributed office system called MULTOS (MULTimedia Office Server) based on this ODA extension. (MULTOS is also described in [77].) Utilization of conceptual component types allows for the exploitation of the aggregation relationship abstraction [90]. For example, in Figure 3, the component type `Sender` can be considered as an aggregation of conceptual component types `Name` and `Address`. A distinction of a concept of typing [9] is made between a *strong component type* and a *weak component type.* A strong component type completely[3] specifies the structure of its instances (e.g., in the relational model [19, 24, 96], a relation schema completely defines the structure of its instances (or tuples)). Thus, the component types are not divisible any further. MULTOS introduces the concept of a weak type to the conceptual data model. A weak type only partially[4] specifies the structure of its instances; i.e., the instances can have more complicated attributes. We are thus able to define document types at different levels of detail.

[2]The terms conceptual component type and component type are used interchangeably here.

[3]Completely in the sense that all component types are not considered as aggregations of other component types.

[4]Partially in the sense that component types can be aggregations of other component types.

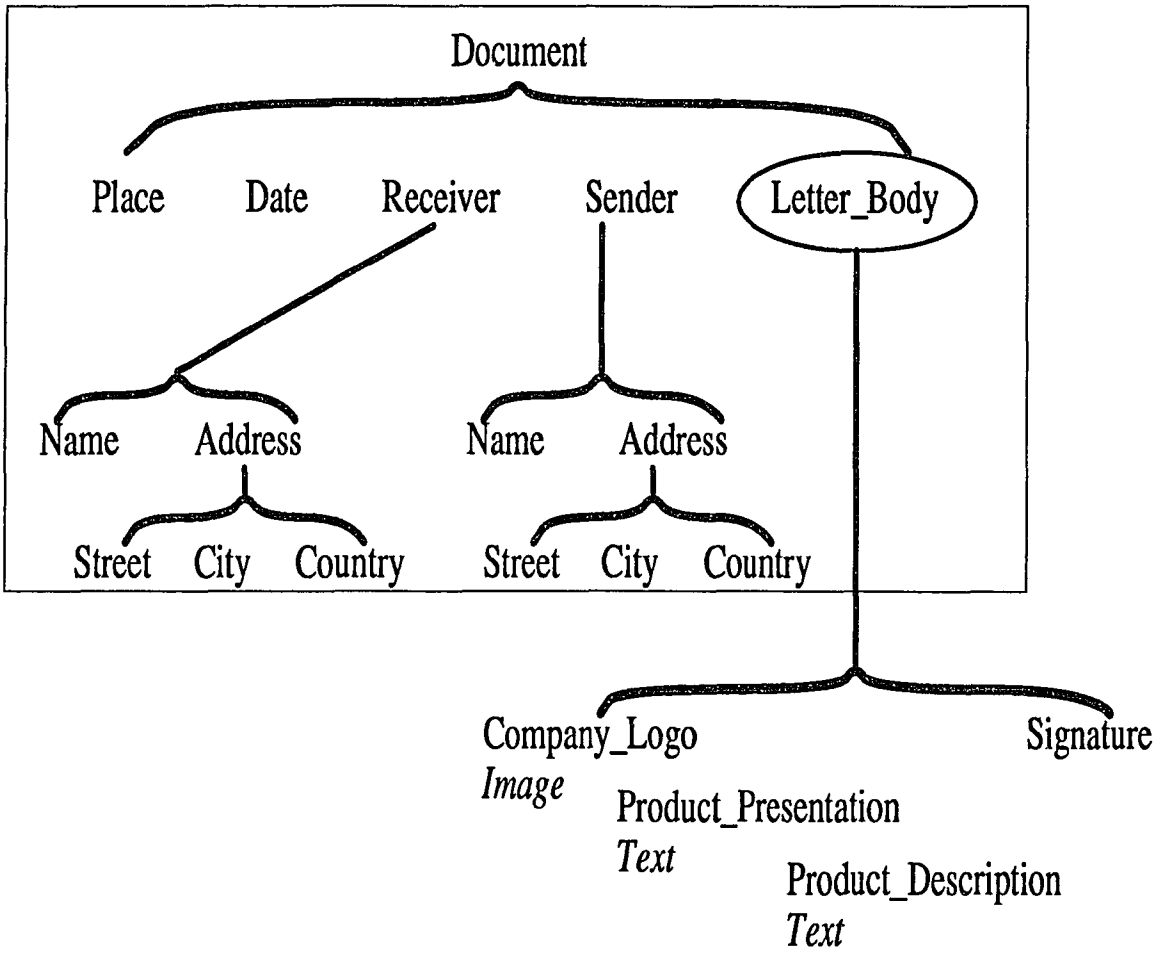


Figure 3: Example of conceptual structures of document types Generic_Letter and Business_Product_Letter

The document types shown in Figure 3 are defined at different levels (see the attribute `Sender` for example). This allows the use of *path notation* [82] in referencing a conceptual component type in a document. For example, to reference only the `City` component type of a sender (cf. Figure 3), the path name would be of the form `Sender.Address.City`.

Lutz et al. [65] develop a document classification system, called MAFIA (MAil-FIlter-Agent), based on MULTOS. The MAFIA provides an automatic document classification system which utilizes the conceptual data model. The basic modeling principles discussed are those of *aggregation* [90], *typing* [9], and *generalization* [90]. The representation of documents is described through the aggregation of conceptual component types.[5] Documents are defined at different levels of detail using the concept of typing called the weak type [9]. (Figure 3 illustrates the concept of the weak type. Note that the two document types Generic_letter and Business_Product_Letter are defined at different levels of detail.) MAFIA, however, is a system only sanctioned for electronic mail.

Hoepner [48] extends ODA to support multimedia documents by integrating synchronization properties and temporal relationships into it. The presentation of multimedia documents is considered to be a set of actions temporarily related to each other, and which are executed in a special intended sequence defined by the user. This scheduling is called synchronization of actions.

Woelk, Kim and Luther [105] present an object-oriented approach to describing multimedia documents. The basic object-oriented aspects that are required in the standard object-oriented paradigm are the notions of instantiation and generalization. Woelk et al. extend these two notions by augmenting the notions of aggregation and relationships to capture the data modeling requirements of multimedia applications.

[5] Recall that a conceptual component type is defined in terms of attributes in MULTOS.

Information in a document is considered, in the first place, to form an aggregation (*part-of*) hierarchy of component node types. A component node, in addition to its place in the aggregation hierarchy, is also considered to be a part of a generalization hierarchy. A generalization hierarchy, in terms of subtyping, defines a component node N as a subtype of a component node M such that M can reuse the attributes defined for N; M becomes a specialization of N. In addition, each of M and N can be an aggregation of component node types. The component node types can result into a *dag* structure since any node can have a relationship with any other node in the aggregation hierarchy and generalization hierarchy. The paper [105] elaborates on augmenting these basic data modeling requirements by utilizing the concept of a *token object* which provides a single mechanism for representing diverse types of data and relationships among these diverse types of data. However, augmenting the notions of instantiation, generalization, and aggregation into one same concept of a token object increases the complexity of property inheritance and constraints management [105]. Property inheritance and constraints management is more complex in this system than in conventional object-oriented systems since the data model discussed here supports the notions of instantiation, generalization, and aggregation.

Christodoulakis et al. [17] represent multimedia documents using two structures: a logical structure representing the logical components of the documents such as titles, sections, paragraphs and so forth, and a physical structure specifying the components of the layout presentation of the documents on an output device such

as the screen of a workstation. A mapping from the logical to the physical structure of a document is provided to specify which components of the logical structure are mapped onto which components of the physical structure. The argument given for separating the logical structure from the physical structure is that the same logical structure shared by two different documents can be presented through different mappings. The authors implement this technique of describing multimedia documents into the MINOS multimedia information system.

Our work differs from the above approaches in several ways. First, we do not model a document using logical, physical, layout or conceptual structures. Instead, we combine these structures and incorporate them into a frame template. The idea of combining the logical and layout structures into a frame template allows the user to store the synopsis, as opposed to the original document, into the template.[6] We call the synopsis of a document a frame instance. Each frame instance is composed of a set of attribute-value pairs. (The frame instance results from instantiating the document's frame template.) The information contained in the frame instance represents the most significant information (i.e., the synopsis) of the document pertinent to the user. Various frame instances can be grouped into a folder based on the nature of their contents. One motivation for considering a frame instance rather than the original document is that the frame instance describes the document in a succinct manner. Also, a user may not be concerned with all the information contained in

[6]In other words, we do not distinguish between logical, physical, layout or conceptual structures of a document. Rather, we concentrate on the information that the user considers to be significant from the document.

a document. When retrieval occurs, the information contained in frame instances suffices to satisfy the user's needs.

Our document model is a dual one – it provides a separate treatment of the structural organization of documents from the real-world folder organization perceived by the user. The structural organization of documents is depicted by a document type hierarchy, which is used for classifying various documents based on the generalization abstraction among the frame templates. The folder organization, on the other hand, mimics the user's document filing system.

The differences stem from the different design philosophies: TEXPROS is for personal use, whereas the systems mentioned above are mainly designed for a multi-user or distributed environment (as a consequence, they need a standard for document interchange). As we have discussed earlier, when using TEXPROS in an information sharing environment, one needs to specify protocols for governing the definitions of frame templates. Note that Gibbs and Tsichritzis [32] also mention frame templates, though their template is used for the layout presentation only. Malone et al. [68] and Clifton et al. [18] propose similar ideas of organizing documents into semi-structured messages. Malone et al. define semi-structured messages as messages of identifiable types, with each type containing a known set of fields, but with some of the fields containing unstructured text or other information. However, these authors do not consider using the documents' synopses or the folder organization.

From a data modeling point of view, our data model does not adhere to the

object-oriented paradigm. In object-oriented communities [7, 33, 56, 104] all the objects that share the same properties (attributes and methods) are collected together into a class. In TEXPROS, a folder is a set of frame instances which need not be over the same frame template. TEXPROS employs an agent-based architecture to automate document categorization and to cope with file reorganization.

Each folder is monitored by an agent. Each agent has a set of criteria and data structures for holding the frame instances. The criteria are used to categorize frame instances (i.e., to place them in appropriate folders). The agents are implemented as objects using an object-oriented approach. The approach encapsulates the internal representations of folders with the operations that manipulate them, thereby enhancing information hiding. The agents communicate with each other through message passing [101, 102].

2.2 Algebraic Languages

As for the data model's algebraic language, there are two groups of work that are closely related to this work. The algebra described by Guting et al. [41] deals with documents. Each document is described in terms of schemas, instances and layouts. A schema is represented by ordered labeled trees, which describe the logical structure and data values contained in a class of documents. A document instance results from instantiating the schema with data values. A layout is a mapping that converts a document instance into a printable or displayable document by merging the data

values of the instance with some fixed text, graphics, etc., and placing the result on document pages. In our work, a frame template does not distinguish between the logical and layout structures of documents. Moreover, we store the synopsis of a document, rather than the original contents, in the frame instance. The order of attributes is significant in Guting's algebra since the schemas are represented as ordered labeled trees. Since the information contained in a frame instance does not reflect any particular (logical or layout) structure, the order of the attributes is insignificant.

The second group deals with non-first-normal-form (NF^2) data models. Despite its rich mathematical foundation, the relational data model introduced by Codd [19] in 1970 requires enhancements[7] for applications such as retrieval of textual data. Considerable research has been discussed in the literature of NF^2 relations to extend the relational data model by dropping the first-normal-form (1NF) assumption. This assumption restricts relation schemas to have indivisible atomic attributes only and the value of any attribute in a tuple is a single value from the domain of that attribute [19, 24, 66, 96]. The NF^2 data model was first advocated by Makinouchi [67] who suggested that the 1NF assumption of the relational data model be relaxed since it was too restrictive. Although his treatment was fairly informal, he showed that relaxing the 1NF assumption could, without loss of generality, model some database applications. Furtado and Kerschberg [29] and Kambayashi et al. [55] also published

[7]The enhancements also include dealing with more complex data objects than flat relations, and specifically those data structures that occur in application areas such as CAD/CAM and office information systems. In these applications, it is necessary to deal with more complex objects than tuples of the classical relational model.

other work related to the early development of this topic.

Jaeschke and Schek [54] proposed a model to generalize the relational model by allowing relations to have set-valued attributes. They proposed operators NEST and UNNEST which convert 1NF relations into non-1NF relations and vice versa. These operators were applied only over single attributes which were defined over atomic domains. The model of Jaeschke and Schek was generalized by Thomas and Fischer [91] by allowing relations to have relation-valued[8] attributes. Since then, several researchers [1, 4, 27, 28, 42, 74, 76, 83] have extended relational database theory to nested relations.

In application areas such as CAD/CAM and office information systems, attributes can be associated with more complicated value sets such as hierarchies and repeating values, and these do not satisfy the 1NF assumption. The algebras of NF^2 data models handle relations with relation-valued attributes. These data models have, as their basis, the theory of relational databases and, hence, topics such as functional dependencies among attributes are relevant here.

In this dissertation, an algebraic language for retrieving and filing various office documents is provided. The algebra supports operations for manipulating both frame instances of different types and folders.

[8] An attribute is *relation-valued* if it is not atomic and its value in a tuple is a set.

2.3 Formal Treatment of Incomplete Information in Database Theory

Considerable research has been discussed in the literature on the treatment of incomplete information in relational database theory. There are different kinds of incomplete information that have been studied. These include null values [11, 12, 21, 35, 39, 52, 81, 99, 106], disjunctive information [30, 53, 62, 80], maybe information [58, 62, 71], and partial values [36, 37]. The concept of null values has been used in database theory to denote missing information for some fields of instances in the database. Null values have been used as placeholders of missing information [39, 99]. In the context of disjunctive information, a fact may be represented by a set of tuples $\{t_1, \ldots, t_n\}$ which correspond to inclusive disjunctions, where each tuple represents a particular case and yet it is not known which of these tuples actually is the definite fact. Here the fact would be the disjunctive fact $t_1 \vee \ldots \vee t_n$. Assuming that some tuple t_i from this set of tuples is later known to be the definite fact, then t_i would subsume the disjunctive fact $t_1 \vee \ldots \vee t_n$. This will remove all the tuples $t_j, 1 \leq j \leq n, j \neq i$, from the database. Liu and Sunderraman [62] consider such tuples that are removed as *maybe tuples* or maybe information. They also discuss how users may want to add maybe information of their own (e.g., for some memorandum whose `Subject` component is unknown, but it is possibly regarding a conference to be held). Grant [36] discusses how partial values may be introduced within the framework of relational databases. He considers the representation of partial numerical values only. Here, partial values are represented by interval entries that denote the lower and upper

bounds or endpoints of the unknown numerical values. For example, if some numerical value between 10 and 20 is all that is known, this fact would be represented by the entry (10,20).

An extension of the relational model for relations that may contain null values of the meaning value at present unknown was posited by Codd in [21]. Imielinski [51] points out that while it is easy to use null values as placeholders in databases to denote missing information, it is more difficult to process queries in the presence of nulls without clear and well defined *semantics.* This dissertation focuses attention on the null values of the form *value at present unknown* for the $\mathcal{D}$_model. It provides a proof-theoretic characterization of the data model and defines the semantics of the null values within the proof-theoretic paradigm.

CHAPTER 3

INFORMAL DESCRIPTION OF THE DATA MODEL

In this chapter we present an informal view of the data model. We introduce relevant terminologies for TEXPROS and provide examples that we will elaborate upon throughout the rest of the dissertation. The emphasis of the discussion in this chapter will be on the intuition and motivations behind the model, rather than on the formal definitions, which will be presented in chapter 4.

3.1 Preliminaries

In offices, information is kept in *documents* [41]. Office documents such as memos, letters, brochures, reports, fliers, legal documents, and so forth can be grouped into *folders* based on the nature of their contents.

In TEXPROS, documents can be grouped into *document classes*; each document class is characterized by a *frame template*, which describes the properties (or attributes) for the class of documents [103] (reminiscent of the *schema* used by Guting et al. [41]). For example, Figure 4 shows the frame template for a class of memorandums that are concerned with meetings. Each memo has attributes `Sender`, `Receiver`, `Subject`, etc. The attributes `Sender`, `Receiver`, `Subject`, and `MemoDate` are the inherited attributes (i.e., common attributes) for the class of generic memos. All attributes of generic memos appear in the frame templates for the meeting memos and

memos of other purposes. In other words, the frame template for the meeting memos inherits [32, 105] all the attributes of generic memos.

<table>
<tr><td>Sender</td><td colspan="3"></td></tr>
<tr><td>Receiver</td><td colspan="3"></td></tr>
<tr><td>Subject</td><td colspan="3"></td></tr>
<tr><td>MemoDate</td><td colspan="3"></td></tr>
<tr><td rowspan="4">MtgDescription</td><td rowspan="2">MtgDay</td><td>MtgDate</td><td></td></tr>
<tr><td>MtgTime</td><td></td></tr>
<tr><td>MtgPlace</td><td colspan="2"></td></tr>
<tr><td>Synopsis</td><td colspan="2"></td></tr>
<tr><td>Remark</td><td colspan="3"></td></tr>
</table>

Figure 4: A frame template for a class of memorandums

A frame template is instantiated by providing it with values extracted from the document to form a *frame instance* which becomes the *synopsis* of the document of the class associated with the frame template. The frame instance contains only the most relevant information of a document pertinent to the user in a precise and succinct manner. A meeting memo and its corresponding frame instance are shown in Figures 5 and 6, respectively. A folder is a set of frame instances which may or may not be *over* the same frame template.

New Jersey Institute of Technology

Office of the Provost
Ext. 3220

MEMORANDUM

TO: Members of Committee on Student Appeals
FROM: Dr. Gary Thomas, Provost
DATE: May 8, 1992
RE: Student Appeals Meeting

****

There will be a meeting of the Committee on Student Appeals on Wednesday, June 10, 1992 at 10:00 a.m. to 1:00 p.m. in Room 504 - Cullimore.

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

Figure 5: A meeting memorandum

Sender	Gary Thomas		
Receiver	Student Appeals Committee		
Subject	Student Appeals Meeting		
MemoDate	05/08/92		
MtgDescription	MtgDay	MtgDate	06/10/92
		MtgTime	10:00
	MtgPlace	Cullimore 504	
	Synopsis		
Remark	contact Mary Armour, Ext. 3275 if can't attend		

Figure 6: A frame instance for the corresponding meeting memorandum

3.2 Data Modeling Approach

3.2.1 Document Classes and Document Types

Here we describe a way of supporting the exploitation of structural commonalities between documents. Documents of similar nature of content are grouped to form a document class. We refer to the structure of a document class as the document type (the frame template) of the document class. A document type defines the common structure of the documents that belong to the same document class. This document type is described by a collection of attributes each having a name and a data type. Thus the document type depicts the representation of these documents.

Consider the attribute `MtgDescription` in Figure 4. This attribute is decomposed into the more detailed attributes `MtgDay`, `MtgPlace` and `Synopsis`. In the model, attributes such as `MtgDescription`, which can be further refined into more detailed attributes, are called composite attributes. *Aggregation* [90] (an object contains other objects) gives the designer the ability to either gradually decompose objects into their detailed components or to aggregate them into higher-level objects. Aggregation enables the designer to define document types at different levels of detail. The different levels of detail form an aggregation hierarchy. The case of refining the type of an attribute applies to composite attributes [9]. The data model is thus based on a *weak component type* typing system [9] since attributes can be refined into different levels of detail. Aggregation allows users to use *path notation* [9, 82, 86] in query expressions. A path notation specifies a path name that is used to reference components

of composite attributes. For example, to reference only the `MtgPlace` component of `MtgDescription`, the path name would be of the form `MtgDescription.MtgPlace`. It may become quite tedious, however, to rely on the path-notation when referring to attributes of composite objects where the aggregation hierarchy becomes very deep. We propose a new operator, called *highlight* (γ) [70] as an alternative to navigate down the hierarchy and take the user to a desired level of aggregation from where the data values can be accessed directly. For example, to access the value of the attribute `MtgDate` of Figure 6, an alternative would be to use the form $\gamma_{\texttt{MtgDescription}_{\{\texttt{MtgDate}\}}}$ instead of the conventional path notation `MtgDescription.MtgDay.MtgDate`.

3.2.2 Document Type Hierarchy

Classification is a form of abstraction which is used widely in semantic data models (e.g., [23, 34, 43, 57]) which are used to describe semantics of data in complex applications and have gained tremendous interest over the past decade. Documents are classified to form document classes. A document class (S) is said to be a subtype of another document class (C) if the documents in the class S are also classified as class C. C is called the supertype of S. This supertype/subtype relationship is also called an IS-A relationship [105]. Document types are related via *specialization* and *generalization* [90]. In this case S *is a specialization* of C (they are connected via the *is-a* relationship) and C *is a generalization* of S. The frame template of S inherits all attributes of the frame template of document class C. In other words, frame instances of S,

which also belong to C, have all the attributes for C (inheritance property [32, 105]). We use *subtyping* [105], where the component types are *weak component types* [9], to generate a (DAG-structured) document type hierarchy. Figure 7 shows a hierarchy for five templates: **Publications, Journals, Proceedings, Book_Chapters** and **Technical_Reports**. Here, **Proceedings**, for example, is a specialization of **Publications** (they are connected through the is-a relationship). Frame instances of **Proceedings** also belong to **Publications** and inherit all attributes for **Publications.**

3.2.3 Folders

In practice, office documents such as memos, letters, brochures, reports, fliers and so forth can be grouped into folders with appropriate labels based on the nature of their contents. We define a *folder* to be a set of frame instances which may or may not be over the same frame template. Thus, a folder may be associated with a collection of frame templates.[9] Figure 8 shows an example of Bill Blake's folder containing five frame instances: publication (fi_1), application for the faculty position (fi_2), employment visa (fi_3), university transcript (fi_4) and a meeting memorandum (fi_5).

[9]This is a deviation from the *relation* of the classical relational data model, in which a relation is associated with precisely one schema [96].

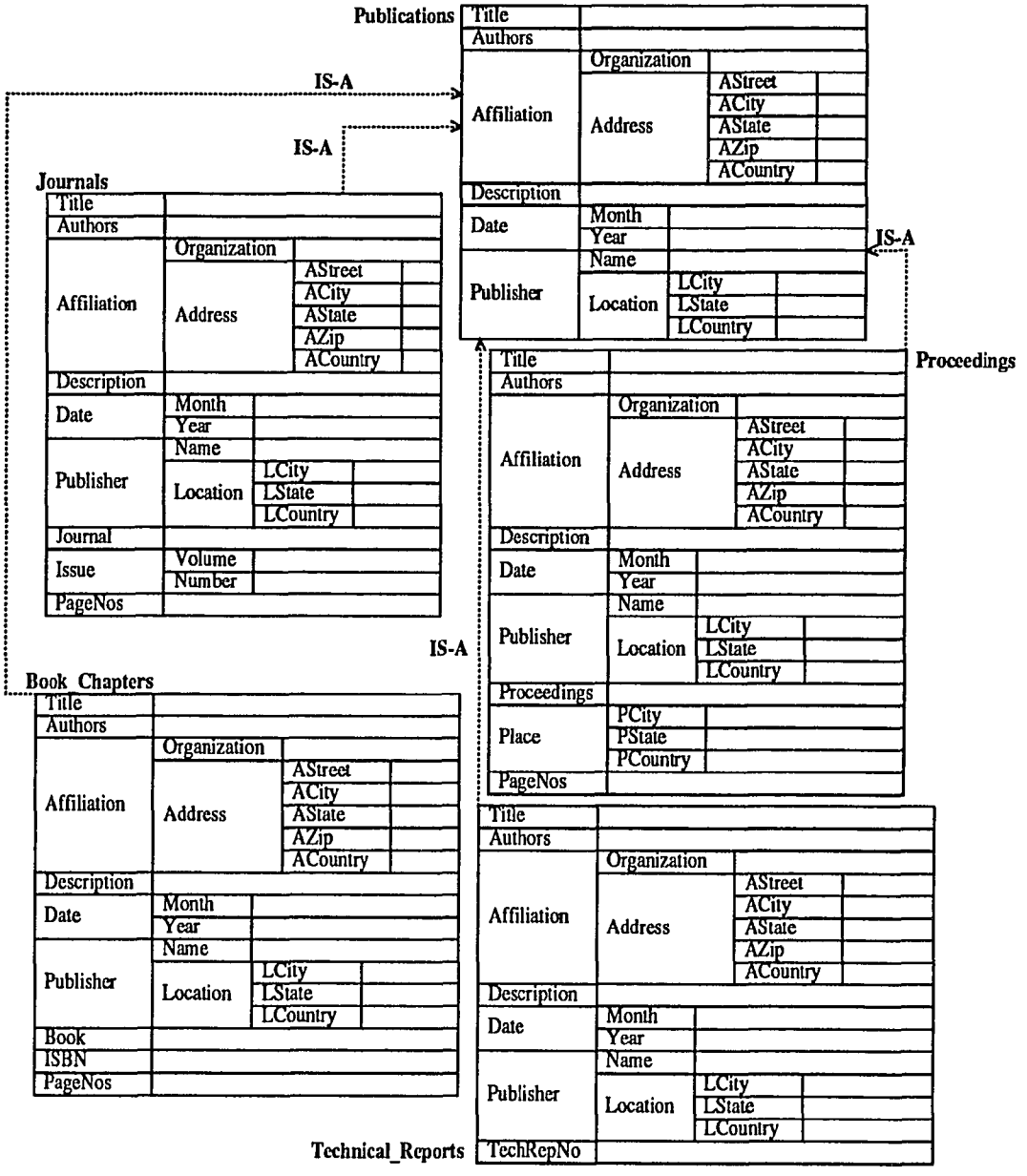


Figure 7: A document type hierarchy

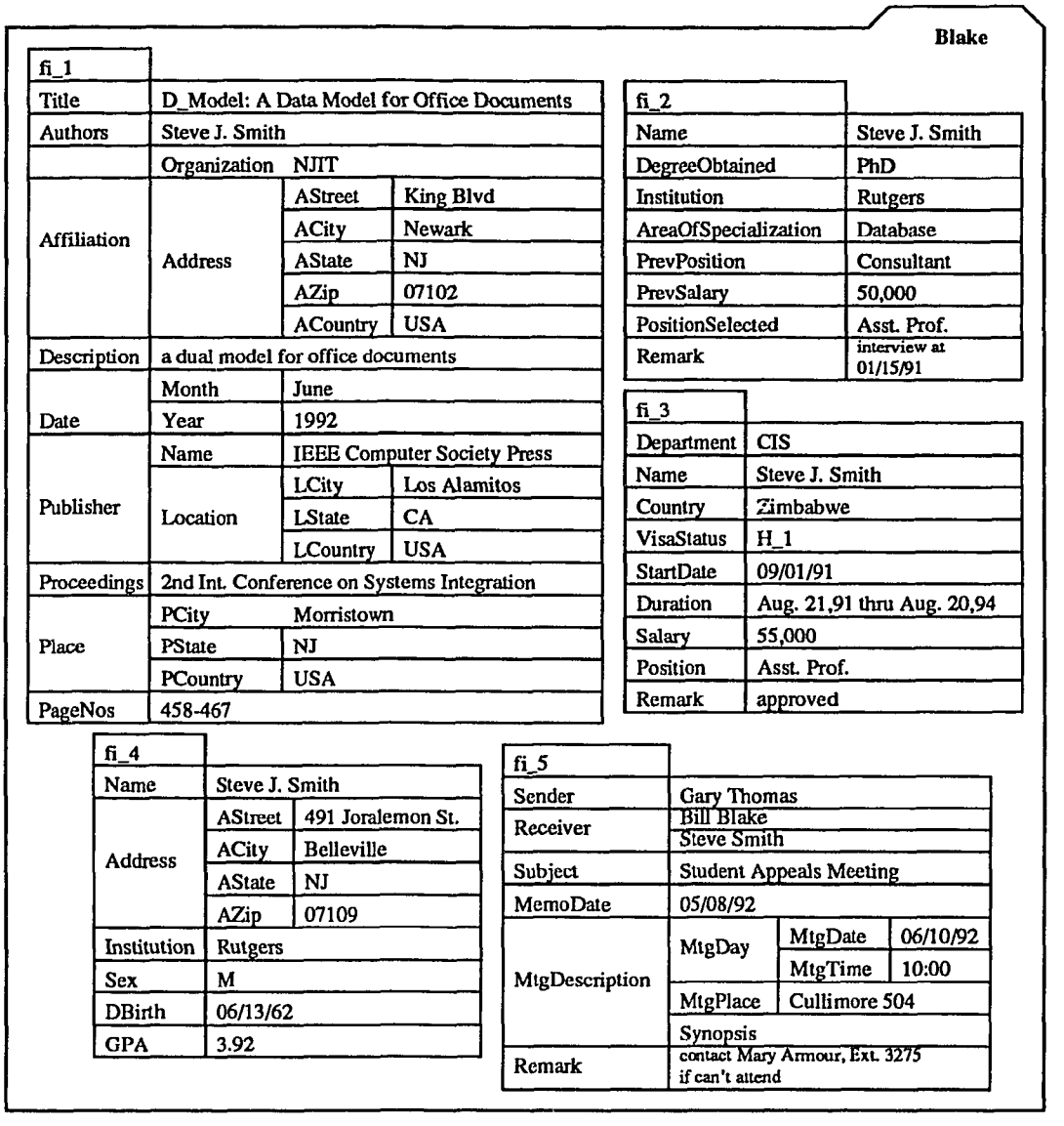


Figure 8: A folder for the faculty member Bill Blake

3.2.4 Folder Organization

Folders in our model are connected to one another forming a folder organization. Such an organization mimics the user's real-world document filing system. Figure 9 shows a partial folder organization that a departmental chairperson of a university may use in keeping track of the status of his/her faculty members and PhD students. The string of text on each folder represents the label of that folder. The arrow from folder f_1 to folder f_2 indicates that f_1 depends on f_2 (i.e., f_1 is a subfolder of f_2). Therefore, folders are connected to one another via the depends-on relationship. In the figure, the CIS Dept folder contains three subfolders: Faculty, Publications and PhD Program. Each of these folders is further broken down into subfolders containing more specific areas of information.

Keeping the folder organization in the system has several advantages. The folder organization provides the user with an intuitively clear view, showing his/her current filing system. Such a view facilitates document retrieval and filing activities. Moreover, since the filing system is composed of folders, the user can query, create, remove an entire folder, rather than perform many separate retrieve/insert/delete operations on frame instances. Separate treatment of the document type hierarchy from the folder organization allows a user to have documents of different structures in the same folder, while at the same time frame instances of these documents of different frame templates can be retrieved.

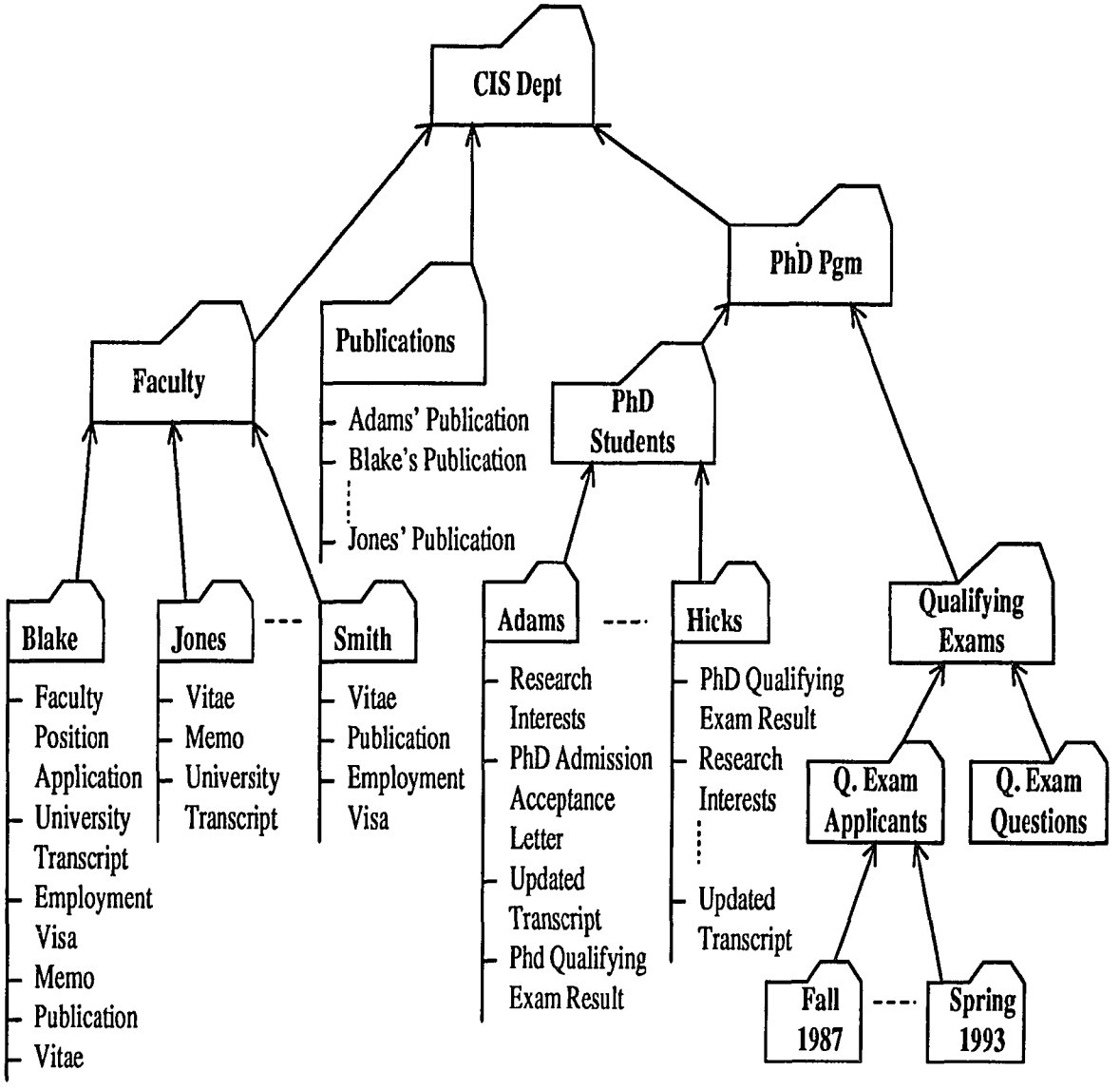


Figure 9: A folder organization

CHAPTER 4

FORMAL FRAMEWORK OF THE DATA MODEL

This chapter presents the formalization of the $\mathcal{D}$_model for the document processing system, called TEXPROS. This formal framework of the data model will be used throughout the dissertation.

4.1 Attribute Types

The data model makes extensive use of the concept of types. An attribute type describes a (possibly infinite) set of values. An example of an attribute type is Name which describes a set of names of people. This attribute type can be broken down into attribute types FirstName, LastName and MiddleName, each of which describes a set of character strings. Let $\mathcal{T} = \{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_n\}$ be a finite set of *attribute types* [86]. Each attribute type is either atomic or composite [86]. (A composite attribute type contains a collection of attribute types in $\mathcal{T}$.) The degree of an attribute type T, denoted $deg(\mathrm{T})$, is the number of component attribute types T comprises. (Thus, $deg(\mathrm{T})$ is 1 if T is atomic and k ($k > 1$) otherwise.) Let $\mathcal{D}$ be a finite set of domains. For example, the set of integers is a domain. Let $dom : \{\mathrm{T} \in \mathcal{T} | deg(\mathrm{T}) = 1\} \longrightarrow \mathcal{D}$ be a total function which associates each atomic attribute type $\mathrm{T} \in \mathcal{T}$ with a domain, $dom(\mathrm{T})$ in $\mathcal{D}$. We generalize the function dom to a composite attribute type recursively as follows. Let $\mathrm{T} = \{\mathrm{B}_1, \mathrm{B}_2, \ldots, \mathrm{B}_m\}$ be a composite attribute type, where

B_j is either atomic or composite. Then the domain of T is the cartesian product of the domains of the B_i's, i.e., $dom(T) = dom(B_1) \times dom(B_2) \times \ldots \times dom(B_m)$.

4.2 Frame Template

Let $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$ be the finite set of documents in the system. The system assigns a unique identifier to each document. The identifier is not visible to the user (reminiscent to the very powerful technique that has been suggested for supporting identity through *surrogates* [21]). For each document $o \in \mathcal{O}$, $\mathbf{F} = \mathbf{F}(o) = \{A_1(T_1), A_2(T_2), \ldots, A_l(T_l)\}$, specifies the frame template associated with o, where A_j is an attribute of type $T_j \in \mathcal{T}$, $1 \leq j \leq l$. An attribute A is composite if its type is composite; otherwise the attribute is atomic. Therefore, the degree of an attribute A_i of type T_i is the degree of the attribute type T_i. The degree of $\mathbf{F}$, denoted $deg(\mathbf{F})$, is the number of attributes in $\mathbf{F}$. If $\mathcal{S}$ is a subset of the attributes of $\mathbf{F}$, the notation $\mathbf{F}.\mathcal{S}$ is used to refer to $\mathcal{S}$. $\mathbf{F}$, also denoted by $\mathbf{F}(o)$, is the underlying or basic data structure[10] *over* which documents in the real-world are defined.[11] We also refer to this data structure as a *document type.*[12] The terms frame template and document type will be used interchangeably in this dissertation.

[10]A data structure is a collection of variables, possibly of several different data types, connected in various ways [2]. (A frame template is a collection of attributes which are possibly of different attribute types.)

[11]This is reminiscent of the *intensional level* which corresponds to the time invariant description of relations (*schemas*) in the relational model [5]. The attributes in a frame template, however, can be composite.

[12]From section 3.2.1 of the previous chapter, a document type is described by a collection of attributes, each having a name and a data type. The data type is precisely the attribute type.

4.3 Frame Instance

Let $\mathbf{F} = \{\mathtt{A}_1(\mathrm{T}_1), \mathtt{A}_2(\mathrm{T}_2), \ldots, \mathtt{A}_m(\mathrm{T}_m)\}$ be a frame template. A frame instance *fi over* $\mathbf{F}$ is a set of attribute-value pairs $\{< \mathtt{A}_1, V_1 >, < \mathtt{A}_2, V_2 >, \ldots, < \mathtt{A}_m, V_m >\}$, where $\mathtt{A}_j$ is an attribute of $\mathbf{F}$, and $V_j \subseteq dom(\mathrm{T}_j), 1 \leq j \leq m$. Let $fi = \{< \mathtt{A}_1, V_1 >, < \mathtt{A}_2, V_2 > , \ldots, < \mathtt{A}_m, V_m >\}$ be a frame instance. If V_j contains a single element v, $\mathtt{A}_j$ is *flat* with respect to fi and $< \mathtt{A}_j, V_j >$ can be represented as $< \mathtt{A}_j, v >$; otherwise $\mathtt{A}_j$ is *nested* (i.e., $\mathtt{A}_j$ has a set of elements V_j). Let $\mathcal{S}$ be any subset of $\{\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_m\}$. We define the $\mathcal{S}$-value of fi, denoted by $fi(\mathcal{S})$, to be the frame instance obtained by deleting from fi those components $< \mathtt{A}_j, V_j >$ where $\mathtt{A}_j \notin \mathcal{S}$. For example, let fi be the frame instance shown in Figure 6 and let $\mathcal{S}$ be the subset $\{\mathtt{MemoDate}, \mathtt{MtgDay}, \mathtt{MtgPlace}\}$. Then $fi(\mathcal{S})$ is the frame instance

MemoDate	05/08/92	
MtgDay	MtgDate	06/10/92
	MtgTime	10:00
MtgPlace	Cullimore 504	

If $\mathcal{S}$ consists of a single attribute, say A, then $fi(\mathcal{S})$ is simply written as $fi(\mathtt{A})$. (In this case, we use $fi[\mathtt{A}]$ to represent the value V in the pair $< \mathtt{A}, V >$ of fi.) Figure 10 shows

a frame instance for a student's transcript.[13] Here, the attributes `Term`, `CourseName`, and `CourseGrade` are nested attributes.

NameOfInstitution	StdId	StdName	Term		CourseName	CourseGrade
			Semester	Year		
NJIT	000901234	John Smith	Spring	1991	CIS630	A
					CIS631	A
					CIS635	B+
			Fall	1991	CIS636	A
					CIS785	B

Figure 10: A frame instance for a student's transcript

Let $\mathtt{X}=\{\mathtt{Y}_1, \mathtt{Y}_2, \ldots, \mathtt{Y}_m\}$ be a composite attribute, where $\mathtt{Y}_j$ is either atomic or composite, $1 \leq j \leq m$. Each of the $\mathtt{Y}_j$'s is called a component attribute of X. Let A and B be attributes. B is said to be a descendant of A if B is a component attribute of G where G is a descendant of A or G is A. Let $fi = \{< \mathtt{A}_1, V_1 >, \ldots, < \mathtt{A}_i, V_i >, \ldots, < \mathtt{A}_l, V_l >\}$ be a frame instance. Suppose $\mathtt{A}_i$ is a composite attribute. Let $\alpha(\mathtt{A}_i)$ be the set of all descendant attributes of $\mathtt{A}_i$. Define the leaves of $\alpha(\mathtt{A}_i)$, denoted by $\alpha_{leaf}(\mathtt{A}_i)$, to be the subset of $\alpha(\mathtt{A}_i)$ comprising the set $\{\mathtt{A} \in \alpha(\mathtt{A}_i) | deg(\mathtt{A}) = 1\}$.

[13]This is a convenient way of showing frame instances with composite attributes that are nested. Note that the attribute `Term` is both composite and nested.

4.4 Document Class

Let $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$ be the finite set of documents in the system. For each document $o \in \mathcal{O}$, the system maintains a frame instance, $\delta(o)$, as the synopsis of the document, consisting of its identifier and values for each attribute of the associated frame template. The set of all these frame instances is denoted by $\delta(\mathcal{O})$. Let $\mathcal{F} = \{\mathbf{F}(o) | o \in \mathcal{O}\}$ be the finite set of all the frame templates currently stored in the system: $\mathcal{F} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n\}$ for some integer n. The system maintains a set of *document classes* $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$, where each c_j is a subset of $\delta(\mathcal{O})$. All the frame instances in a document class c_j are *over* the same frame template $\mathbf{F}(c_j)$, (or denoted by $\mathbf{F}$ if c_j is understood).

4.5 Document Type Hierarchy

Document types are related via *specialization* and *generalization* [90]. As discussed in chapter 3, document types can be ordered in terms of is-a relationships. This kind of ordering is very natural and allows us to say, for example, that a call-for-meeting memo is a memo, or that a memo is a document. We now formally define an ordering on document types called *is-a*.

Definition 4.1: Let $\mathbf{F}$ and $\mathbf{F}'$ be two frame templates (document types) with $deg(\mathbf{F}) > deg(\mathbf{F}')$. $\mathbf{F}$ *is-a* $\mathbf{F}'$ if for every attribute $\mathtt{A}'_j$ of type T'_j in $\mathbf{F}'$, there is an attribute $\mathtt{A}_i$ of type T_i in $\mathbf{F}$ such that $\mathtt{A}_i = \mathtt{A}'_j$, $\mathrm{T}_i = T'_j$. In this case, $\mathbf{F}$ is a specialization of $\mathbf{F}'$; $\mathbf{F}'$ is a generalization of $\mathbf{F}$.

Now, we formally define the document type hierarchy $\mathcal{D}_t\mathcal{H}$, recursively as follows:

1. A distinguished document type **F** designated as the generic document type is a document type hierarchy $\mathcal{D}_t\mathcal{H}$.

2. Let **F** be a document type. Let $\mathcal{D}_t\mathcal{H}_1$, $\mathcal{D}_t\mathcal{H}_2, \ldots, \mathcal{D}_t\mathcal{H}_k$ be document type hierarchies with generic document types $\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_k$, respectively such that $\mathbf{F}_i$ *is-a* $\mathbf{F}, 1 \leq i \leq k$. Then a new document type hierarchy can be constructed by designating **F** as the generic document type, and $\mathbf{F}_i$ *is-a* $\mathbf{F}, 1 \leq i \leq k$.

4.6 Folder

A *folder* f is a finite set of frame instances which may or may not be *over* the same frame template. A folder is user-defined and may be associated with a collection of frame templates. We use $\mathsf{f}(\mathbf{F})$ to denote the subset of frame instances in f that are *over* the same frame template **F**. That is, $\mathsf{f}(\mathbf{F}) = \{fi \in \mathsf{f} | fi$ is a frame instance *over* $\mathbf{F}\}$. A folder can also be defined by a first-order predicate clause P [88].

Definition 4.2: Let *comp* be a comparison operator in $\{<, \leq, >, \geq, =, \neq, \subset, \subseteq, \supset, \supseteq, \in, \notin\}$. A predicate atom ($PA$) has one of the following two forms:

- Type PA-1: attribute *comp* constant
- Type PA-2: attribute1 *comp* attribute2.

Definition 4.3: A predicate clause (PC) is a logical combination of predicate atoms:

1. each PA is a PC
2. if P is a PC, then $(\neg P)$ is a PC
3. if P_1, P_2 are PCs, then $(P_1 \wedge P_2)$, $(P_1 \vee P_2)$ are PCs
4. nothing else is a PC.

Formally, $\mathsf{f} = \{fi | P(fi) \text{ is true}\}$ is a folder. For example, let $\mathbf{F}_1, \mathbf{F}_2$, and $\mathbf{F}_3$ be the frame templates **PhDAcceptLetter, UpdatedTranscript,** and **PhDQEResult** respectively. Let $\mathsf{f}(\mathbf{F}_1) = \{fi \in \mathsf{f} | fi \text{ is a frame instance } over\ \mathbf{F}_1\}$. Let $\mathsf{f}(\mathbf{F}_2) = \{fi \in \mathsf{f} | fi \text{ is a frame instance } over\ \mathbf{F}_2\}$. Let $\mathsf{f}(\mathbf{F}_3) = \{fi \in \mathsf{f} | fi \text{ is a frame instance } over\ \mathbf{F}_3\}$. Let $P(fi)$ be the predicate clause,

$$fi \in \bigcup_{i=1}^{3} \mathsf{f}(\mathbf{F}_i) \ \wedge \ (fi[\texttt{Receiver}] = Fortune\ S.\ Mhlanga \ \vee$$
$$fi[\texttt{StdName}] = Fortune\ S.\ Mhlanga),$$

where either `Receiver` or `StdName` is an attribute in each $\mathbf{F}_i, 1 \leq i \leq 3$. Then $\mathsf{f} = \{fi | P(fi) \text{ is true}\}$ is a folder, which possibly contains frame instances over $\mathbf{F}_1, \mathbf{F}_2$ or $\mathbf{F}_3$ with *Fortune S. Mhlanga* as the Receiver or the StdName.

Folders are related via the *depends-on* relationship.

Definition 4.4: Let f_i and f_j be two folders. Then f_i *depends-on* f_j if and only if $\mathsf{f}_i \subseteq \mathsf{f}_j$.

4.7 Folder Organization

A *folder organization* $\mathcal{FO}$ is a finite set of folders $\mathcal{FO} = \{\mathsf{f}_1, \mathsf{f}_2, \ldots, \mathsf{f}_q | \mathsf{f}_i \subseteq \delta(\mathcal{O}), 1 \leq i \leq q\}$ such that there is one distinguished folder called Document along with a *depends-on* relationship defined on the folders. We formally define the folder organization recursively as follows:

1. A distinguished folder f designated as the Document folder is a folder organization.

2. Let f be a folder. Let $\mathcal{FO}_1, \mathcal{FO}_2, \ldots, \mathcal{FO}_k$ be folder organizations with Document folders $\mathsf{f}_1, \mathsf{f}_2, \ldots, \mathsf{f}_k$, respectively such that $\mathsf{f}_i \subseteq \mathsf{f}, 1 \leq i \leq k$. Then a new folder organization can be constructed by designating f as the Document folder, and f_i *depends-on* f, $1 \leq i \leq k$.

In the model, the user defines a folder organization $\mathcal{FO}$, which contains all the folders of interest. We refer to folders that have dependent folders in $\mathcal{FO}$ as *abstract* folders and to those that do not have dependent folders as *concrete* folders.

A *filing organization* $\mathcal{FLO}$ is the tuple $< \mathcal{D}_t\mathcal{H}, \mathcal{FO} >$, which is an organization of the document type hierarchy and the folder organization. The $\mathcal{FLO}$ (or loosely

speaking, the data model) is the formal tool for describing the filing system of interest perceived by the user. The $\mathcal{D}_t\mathcal{H}$ describes the *intensional level* of the data model, which corresponds to the time invariant description of frame templates and the is-a relationships among them. The $\mathcal{FO}$ describes the *extensional level* which corresponds to the actual contents of folders at any instant and also the depends-on relationships between the folders.

We now conclude this chapter by formally defining the $\mathcal{D}$_model.

Definition 4.5: The $\mathcal{D}$_model *over* $\mathcal{O}, \mathcal{T}, \mathcal{D}, \mathcal{F}$, $\delta, \mathcal{D}_t\mathcal{H}$, and $\mathcal{FO}$ is a 7-tuple $< \mathcal{O}, \mathcal{T}, \mathcal{D}, \mathcal{F}, \delta, \mathcal{D}_t\mathcal{H}, \mathcal{FO} >$, where

- $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$ is a finite set of documents;
- $\mathcal{T} = \{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_n\}$ is a finite set of attribute types;
- $\mathcal{D} = \{\mathsf{D}_1, \mathsf{D}_2, \ldots, \mathsf{D}_l\}$ is a finite set of domains;
- $\mathcal{F} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_p\}$ is a finite set of frame templates;
- δ is a function which maps a document $o \in \mathcal{O}$ to its frame instance $\delta(o)$. The frame instance consists of the document's identifier and values for each attribute of the associated frame template $\mathbf{F}(o) \in \mathcal{F}$;
- $\mathcal{D}_t\mathcal{H}$ is the document type hierarchy;

- $\mathcal{FO}$ is the folder organization consisting of all the folders defined by the user where each folder is a finite set of frame instances (which need not be *over* the same frame template).

CHAPTER 5

INFORMAL VIEW OF THE $\mathcal{D}$_ALGEBRA

In this chapter we informally introduce the operations of the $\mathcal{D}$_algebra by examples based on a part of the folder organization shown in Figure 9. The folders of interest in the system are:

1. PhDStds which is the name of the PhD Students folder. The frame templates associated with PhDStds include:

   - the frame template for PhD Admission Acceptance Letter:

     PhDAcceptLetter{Sender(Name), Receiver(Name), LetterDate(Date), SemAccepted(Semester), Remark(Contents), CC(Names)}.

   - the frame template for PhD Qualifying Exam Result:

     PhDQEResult{Sender(Name), Receiver(Name), StdCd(Code), NoticeDate(Date), SemTaken(Semester), Outcome(Result), Recommendation(Contents), CC(Names)}.

   - the frame template for Updated Transcript:

     UpdatedTranscript{StdId(Id), StdName(Name), CourseGrades(Grades), GPA(GrdPtAvg), TransDate(Date)}.

2. Publications which is the name of the Publications folder. The frame template associated with Publications is:

- the frame template for Publications:

  Publication{`PubTitle`(Title), `Author`(Names), `Affil`(Affiliation), `Description`(Contents), `PubName`(Publication), `Location`(Place), `YrOfPub`(Year), `Publisher`(Contents), `PageNos`(Pages)}.

3. QExams which is the name of the Qualifying Exams folder. The frame templates associated with QExams include:

   - the frame template for Qualifying Exam Applicants:

     QEApplicant{`StdName`(Name), `ExamTime`(Semester), `FirstChoice`(TopicArea), `SecondChoice`(TopicArea)}.

   - the frame template for Qualifying Exam Questions:

     QEQuestion{`Paper`(TopicArea), `Examiner`(Name), `ExamTime`(Semester), `Problems`(Questions)}.

Table 1 lists the finite set of attribute types ($\mathcal{T}$) corresponding to the above document file.

Table 1: Attribute types for the example (in alphabetical order)

Attribute Type T	deg(T)	dom(T)
Affiliation	1	$TEXT$
City	1	$SetOfCharString$
Code	1	$AlphaNumeric$
Contents	1	$TEXT$
Course	1	$SetOfCharString$
CourseGrade	2	dom(Course) $\times$ dom(Grade)
Date	3	dom(Month) $\times$ dom(Day) $\times$ dom(Year)
Day	1	$Integer$
FName	1	$SetOfCharString$
Grade	1	$\{A, B+, B, C+, C, F\}$
Grades	1	$\{dom(\text{CourseGrade})\}^+$
GrdPtAvg	1	$Real$
Id	1	$Integer$
LName	1	$SetOfCharString$
MName	1	$SetOfCharString$
Month	1	$Integer \mid SetOfCharString$
Name	3	dom(FName) $\times$ dom(LName) $\times$ dom(MName)
Names	1	$\{dom(\text{Name})\}^+$
Pages	1	$Real \times Real$
Place	2	dom(City) $\times$ dom(State)
Points	1	$Real$
Publication	1	$TEXT$
Quest	1	$TEXT$
Question	3	dom(QuestNo) $\times$ dom(Quest) $\times$ dom(Points)
Questions	1	$\{dom(\text{Question})\}^+$
QuestNo	1	$Real$
Result	1	$\{Pass, Fail, Conditional\}$
Season	1	$\{Fall, Spring, Summer\}$
Semester	2	dom(Season) $\times$ dom(Year)
State	1	$SetOfCharString$
Title	1	$TEXT$
TopicArea	1	$SetOfCharString$
Year	1	$Integer$

Example 5.1: *List the first name and last name of all PhD students who passed the Qualifying Examination during the Spring of 1990.*

$$\mathsf{PassedPhDStds} := \sigma_{\mathtt{Outcome}=\mathit{Pass} \wedge \mathtt{SemTaken}=\mathit{Spring\ 1990}}(\mathsf{PhDStds}(\mathbf{PhDQEResult}))$$

$$\gamma_{\mathtt{Receiver}_{\{\mathtt{FName},\mathtt{LName}\}}}(\mathsf{PassedPhDStds})$$

This example illustrates the use of the *select* (σ) and *highlight* (γ) operators. First the selection operation finds all the PhD students who passed the Qualifying Examination during the Spring of 1990. Then the names of these PhD students are displayed by highlighting only their first names and last names. □

Example 5.2: *List all the publications of Samantha Adams.*

$$\sigma_{\mathtt{Author}=\mathit{Samantha\ Adams}}(\mu_{\mathtt{Author}}(\mathsf{Publications}(\mathbf{Publication})))$$

Example 5.2 illustrates the use of the *unnest* (μ) operator known from the literature on non-first-normal-form relations. First the *unnest* operation will *flatten* the Publications folder *over* attribute `Author`. The *selection* operation then uses an *exact* match to find all the publications of Samantha Adams. Alternatively, the result can be obtained by using

$$\sigma_{\mathtt{Author} \supseteq \mathit{Samantha\ Adams}}(\mathsf{Publications})$$

□

Example 5.3: *List all the PhD students who applied to take the Qualifying Examination the same semester that Mary Jones applied.*

Let $P :=$ StdName2 $=$ *Mary Jones*

$$\text{QExams2} := \sigma_P(\rho_{(\texttt{StdName2}\leftarrow\texttt{StdName},\texttt{ExamTime2}\leftarrow\texttt{ExamTime})}(\text{QExams}(\textbf{QEApplicant})))$$

$$\pi_{(\texttt{StdName})}((\text{QExams}) \bowtie_{(\texttt{ExamTime}=\texttt{ExamTime2})}(\text{QExams2}))$$

In this example we illustrate the use of the *renaming* (ρ), *project* (π), and *join* ($\bowtie$) operators. We perform a join of the QExams folder with itself. The join is accomplished by first generating a folder QExams2 (containing only *Mary Jones'* application) which is a copy of QExams where StdName is *renamed* to StdName2 and ExamTime is *renamed* to ExamTime2. Then a join operation is performed on the two folders QExams and QExams2 to find all the PhD students from QExams whose ExamTime is the same as ExamTime2 of QExams2. □

Example 5.4: *How many times has Samantha Adams taken the Qualifying Examination?*

$$\text{count}_{\texttt{Receiver}}(\sigma_{\texttt{Receiver}=\textit{Samantha Adams}}(\text{PhDStds}(\textbf{PhDQEResult})))$$

Example 5.4 illustrates the use of the count aggregate operator. In this case we need to know how many times *Samantha Adams received* a PhD Qualifying Exam Result.

The *selection* operation finds all of *Samantha Adams'* qualifying exam results. Then the count operation returns the number of *Samantha Adams'* qualifying exam results.

□

Example 5.5: *Display the Database question which was weighted the most during the Fall 1990 Qualifying Examination.*

DBF90QExams :=

$$\mu_{\texttt{Problems}}\ (\sigma_{\texttt{Paper}=\mathit{Database}\wedge\texttt{ExamTime}=\mathit{Fall\ 1990}}(\mathsf{QExams}(\mathbf{QEQuestion})))$$

$$x := \mathsf{max}_{\gamma_{\texttt{Problems}_{\{\texttt{Points}\}}}}(\mathsf{DBF90QExams})$$

$$\gamma_{\texttt{Problems}_{\{\texttt{Quest}\}}}\ (\sigma_{\gamma_{\texttt{Problems}_{\{\texttt{Points}\}}}=x}(\mathsf{DBF90QExams}))$$

The first *selection* operation finds the database qualifying exam paper that was given during the Fall of 1990. The max operator returns the maximum value of points for a particular question of this paper. Finally we *select* the problem which has this maximum value and project it *over* the question of the problem. Note that in this example `Problems` is both nested and composite, because for any frame instance *fi over* **QEQuestions**, *fi*(`Problems`) is a set of questions where each question also includes a question number and the number of points. □

CHAPTER 6

FORMALISM OF THE $\mathcal{D}$_ALGEBRA

A language is said to be *procedural* if it describes, step by step, the computation of the result from the database instance[14] [5]. The $\mathcal{D}$_algebra is a procedural language. Table 2 lists all the operators supported by the $\mathcal{D}$_algebra, which can be categorized into seven classes.

Table 2: Operators of the $\mathcal{D}$_Algebra

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

[14] In our model, a folder organization corresponds to a database instance.

Some of the operators have only operands (classes 1-4), and some have both operands and parameters (classes 5-7). Each class of operators will be discussed in turn in the following sections.

6.1 Class 1: Set Theoretic Operators

The first class of operators consists of the binary set theoretic operators for folders. These include the *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$).

Definition 6.1: Let f_1 and f_2 be folders.

- The *union* of f_1 and f_2, denoted $f_1 \cup f_2$, is the set of frame instances that belong to either f_1 or f_2 or both, i.e., $f_1 \cup f_2 = \{fi | (fi \in f_1) \vee (fi \in f_2)\}$.
- The *intersection* of f_1 and f_2, denoted $f_1 \cap f_2$, is the set of frame instances that are in both f_1 and f_2, i.e., $f_1 \cap f_2 = \{fi | (fi \in f_1) \wedge (fi \in f_2)\}$.
- The *difference* of f_1 and f_2, denoted $f_1 - f_2$, is the set of frame instances that are in f_1 but not in f_2, i.e., $f_1 - f_2 = \{fi | (fi \in f_1) \wedge (fi \notin f_2)\}$.

Proposition 6.1: Both the union and the intersection operations are commutative; that is, $f_1 \cup f_2 = f_2 \cup f_1$ and $f_1 \cap f_2 = f_2 \cap f_1$ for any two folders f_1 and f_2.

Proposition 6.2: Both the union and the intersection operations are *associative*; that is, $f_1 \cup (f_2 \cup f_3) = (f_1 \cup f_2) \cup f_3$ and $f_1 \cap (f_2 \cap f_3) = (f_1 \cap f_2) \cap f_3$ for any three folders f_1, f_2 and f_3.

Proposition 6.3: The difference operation is **not** commutative; that is, $f_1 - f_2 \neq f_2 - f_1$ for some folder f_1 and f_2. And the difference operation is **not** associative; that is, $f_1 - (f_2 - f_3) \neq (f_1 - f_2) - f_3$ for some folders f_1, f_2 and f_3.

6.2 Class 2: Project Operator

The second class of operators is the unary restrictive operator *project* (π) for folders. Given a folder f, the projection of f onto a subset of attributes $\mathcal{S}$, denoted $\pi_{\mathcal{S}}(f)$, yields a new folder g which is a restriction of f to the attributes in $\mathcal{S}$.

Definition 6.2: Let f be a folder, and let $\mathcal{S} = \{A_1, \ldots, A_k\}$ where A_j is of type $T_j \in \mathcal{T}$. The projection of f onto $\mathcal{S}$, denoted $\pi_{\mathcal{S}}(f)$, yields the set $\{fi(\mathcal{S}) \mid fi \in f$ and $\mathcal{S}$ is a subset of the frame template of $fi\}$.

Example 6.1: Consider the folder Blake in Figure 8. Then $\pi_{\texttt{Title,Authors,Date}}(\text{Blake})$ returns a folder of frame instances having attributes `Title`, `Authors` and `Date`, shown in Figure 11. □

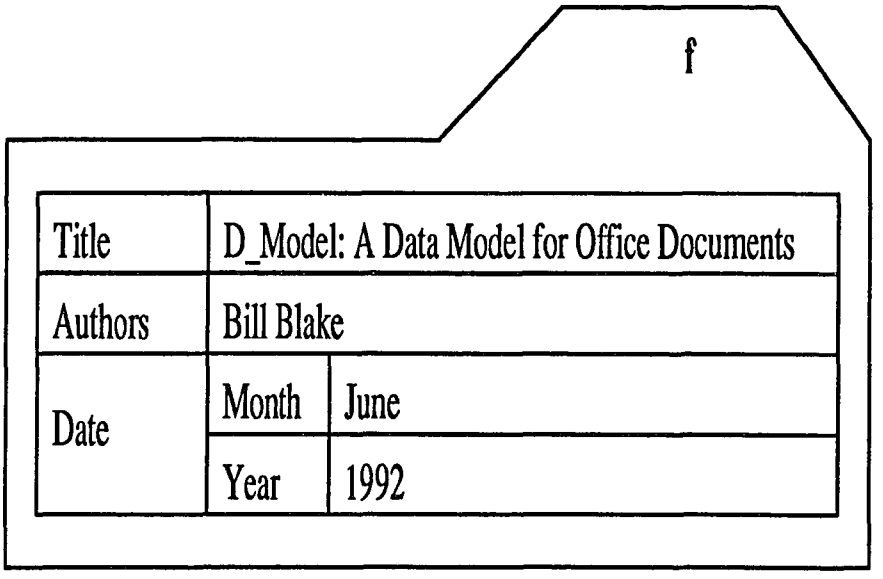


Figure 11: Result of projecting Blake onto `Title,Authors,Date`

The following asserts that projection is distributive over the binary boolean operations for folders.

Proposition 6.4: Let f_1 and f_2 be two folders. Let β be $\cup$, $\cap$, or $-$. Then, $\pi_S(f_1 \beta f_2) = \pi_S(f_1) \beta \pi_S(f_2)$.

6.3 Class 3: Join Operators

We now discuss operators that are similar to the join operation used in the relational algebra. The join operation is defined in terms of the *concatenation* and *cartesian product* whose syntax and semantics are given below.

Definition 6.3: Let $fi_1 = \{< \mathtt{A}_1, V_1 >, \ldots, < \mathtt{A}_m, V_m >\}$ and $fi_2 = \{< \mathtt{B}_1, V_1' > , \ldots < \mathtt{B}_n, V_n' >\}$ be two frame instances over frame templates $\mathbf{F}_1 = \{\mathtt{A}_1, \ldots, \mathtt{A}_m\}$ and $\mathbf{F}_2 = \{\mathtt{B}_1, \ldots, \mathtt{B}_n\}$ respectively. Assume that the attributes in $\mathbf{F}_1$ and $\mathbf{F}_2$ are all distinct. (If there are common attributes, we can rename them, as will be explained later.) Then the *concatenation* of fi_1 and fi_2, denoted $fi_1 \bullet fi_2$, is a frame instance fi over $\mathbf{F}_1 \cup \mathbf{F}_2$, where $fi = \{< \mathtt{A}_1, V_1 >, \ldots, < \mathtt{A}_m, V_m >, < \mathtt{B}_1, V_1' >, \ldots < \mathtt{B}_n, V_n' >\}$.

A cartesian product of two folders $\mathtt{f}_1$ and $\mathtt{f}_2$ is a set of frame instances which are formed as a result of the concatenation of *every* frame instance of $\mathtt{f}_1$ with *every* frame instance of $\mathtt{f}_2$.

Definition 6.4: Let $\mathtt{f}_1$ and $\mathtt{f}_2$ be two folders. Then, the *cartesian product* of $\mathtt{f}_1$ and $\mathtt{f}_2$, denoted $\mathtt{f}_1 \times \mathtt{f}_2$, is the folder $\{fi_1 \bullet fi_2 | fi_1 \in \mathtt{f}_1, fi_2 \in \mathtt{f}_2\}$.

The *join* ($\bowtie$) operator is applied to two folders. The join of folders $\mathtt{f}_1$ and $\mathtt{f}_2$ based on a first-order predicate P, denoted $\mathtt{f}_1 \bowtie_P \mathtt{f}_2$, is the set of frame instances in the cartesian product of $\mathtt{f}_1$ and $\mathtt{f}_2$ that satisfy P. The join operation is formally defined as follows:

Definition 6.5: Let f_1 and f_2 be two folders and let P be a predicate clause. Then $f_1 \bowtie_P f_2 = \{fi_1 \bullet fi_2 | (\exists fi_1 \in f_1)(\exists\ fi_2 \in f_2)(P(fi_1 \bullet fi_2)$ is true)$\}$ where $fi_1 \bullet fi_2$ is the concatenation of fi_1 to fi_2.

Example 6.2: Consider the two folders Doe and Assistantships shown below. Assume that the folder Assistantships is associated with the frame template **Assistantship.**

Doe

Name	John Doe	
Status	ABD	
University_ Attended	Univ_Name	NYU
	Degree_ Sought	MS
	Year	1988
GRE_ Scores	Verbal	500
	Quantitative	780
	Analytical	680

Name	John Doe	
Sex	M	
Address	Street	2 Bay St
	City	Newark
	State	NJ
	Zip	07102
GPA	3.80	

Assistantships

StdName	John Doe
Assistantship_ Type	GA
Duty	CIS library attendant
Supervisor	P. A. Ng

StdName	James Jones
Assistantship_ Type	TA
Duty	Teach CIS 431
Supervisor	P. A. Ng

Then, Doe $\bowtie_{\text{Assistantship.Std\_Name}=John\ Doe}$ Assistantships returns the folder f =

f

Name	John Doe	
Status	ABD	
University_ Attended	Univ_Name	NYU
	Degree_ Sought	MS
	Year	1988
GRE_ Scores	Verbal	500
	Quantitative	780
	Analytical	680
Std_Name	John Doe	
Assistantship_ Type	GA	
Duty	CIS library attendant	
Supervisor	P. A. Ng	

Name	John Doe	
Sex	M	
Address	Street	2 Bay St
	City	Newark
	State	NJ
	Zip	07102
GPA	3.80	
Std_Name	John Doe	
Assistantship_ Type	GA	
Duty	CIS library attendant	
Supervisor	P. A. Ng	

□

An important operation in dealing with self-join [85] (i.e., a folder is joined with itself) is *renaming*. This operator helps avoid the ambiguity when referring to an attribute in the corresponding frame templates. The syntax and semantics of the operator is given below:

Definition 6.6: Let $\mathcal{F} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n\}$ be the set of frame templates associated with a folder f. Let $fi = \{< \mathtt{A}_1, V_1 >, \ldots, < \mathtt{A}_k, V_k >\}$ be a frame instance over frame template $\mathbf{F} = \{\mathtt{A}_1(\mathrm{T}_1), \mathtt{A}_2(\mathrm{T}_2), \ldots, \mathtt{A}_k(\mathrm{T}_k)\}$ in $\mathcal{F}$. Suppose $\mathtt{A} \notin \{\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_k\}$ with A and $\mathtt{A}_j$ having the same type T_j, for some $1 \leq j \leq k$. Then

- renaming $\mathtt{A}_j$ to A on fi, denoted $\rho_{\mathtt{A} \leftarrow \mathtt{A}_j}(fi)$, returns a frame instance $\{< \mathtt{A}_1, V_1 >, \ldots, < \mathtt{A}_{j-1}, V_{j-1} >, < \mathtt{A}, V_j >, < \mathtt{A}_{j+1}, V_{j+1} >, \ldots, < \mathtt{A}_k, V_k >\}$;
- renaming $\mathtt{A}_j$ to A on the set $S \subseteq \mathsf{f}$ of all the frame instances over the template $\mathbf{F}$, denoted $\rho_{\mathtt{A} \leftarrow \mathtt{A}_j}(\mathsf{f}(\mathbf{F}))$, yields $\{\rho_{\mathtt{A} \leftarrow \mathtt{A}_j}(fi) | fi \in S\}$;
- renaming $\mathtt{A}_j$ to A on f, denoted $\rho_{\mathtt{A} \leftarrow \mathtt{A}_j}(\mathsf{f})$, yields

$$\bigcup_{\mathbf{F} \in \mathcal{F}} (\rho_{\mathtt{A} \leftarrow \mathtt{A}_j}(\mathsf{f}(\mathbf{F}))).$$

6.4 Class 4: Select Operator

The fourth class of operators is the unary restrictive operator *select* (σ) for folders. The syntax of the selection operation on a folder f is given by $\sigma_P(\mathsf{f})$, where P is a

first-order predicate clause.

Definition 6.7: The value of a selection operation can be determined according to the following cases, depending on whether all attributes appearing in P (a predicate clause) come from the same frame template. Let A and C be an attribute and constant, respectively.

- All attributes appearing in P come from the same frame template:

  1. If P is a predicate atom

     - case 1.1: For $P = \mathtt{A}\ comp\ C$, then $P(fi)$ is true if ($fi[\mathtt{A}]\ comp\ C$) is true.

     - case 1.2: For $P = \mathtt{A}_1\ comp\ \mathtt{A}_2$, then $P(fi)$ is true if ($fi[\mathtt{A}_1]\ comp\ fi[\mathtt{A}_2]$) is true.

  2. Let $P = P_1 \alpha P_2$, where P_1 and P_2 are predicate clauses.

     $P(fi)$ is true if $P_1(fi)\alpha P_2(fi)$ is true where $\alpha \in \{\wedge, \vee\}$.

  Then, $\sigma_P(\mathtt{f}) = \{fi | fi \in \mathtt{f} \wedge P(fi) \text{ is true}\}$.

- Attributes appearing in P come from different frame templates:

  Create a new folder f′ containing frame instances. Each of these frame instances is formed by joining the frame instances of the given folder f, which contain some or all attributes appearing in P. Then $\sigma_P(\mathtt{f}) = \sigma_P(\mathtt{f}') = \{fi | fi \in \mathtt{f}' \wedge P(fi) \text{ is true}\}$. Formally, this can be stated as follows. Let $\mathcal{A} = \{\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_k\}$

be the finite set of attributes appearing in P. Let $\mathcal{F} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n\}$ be the finite set of frame templates such that for each $\mathbf{F}_i \in \mathcal{F}$, $\mathbf{F}_i \cap \mathcal{A} \neq \emptyset$.

$\mathbf{F} := \mathbf{F}_1 \in \mathcal{F}$;

$\mathrm{f}' := \mathrm{f}(\mathbf{F})$;

$\alpha := \emptyset$; $\beta := \emptyset$;

$i := 2$;

while $i \leq n$ do

begin

for each $\mathtt{A} \in (\mathbf{F} \cap (\mathbf{F}_i \in \mathcal{F}))$ do

begin

$\rho_{\mathtt{A}'/\mathtt{A}}(\mathrm{f}(\mathbf{F}_i))$;

$\alpha := \alpha \cup \mathtt{A}'$;

$\beta := \beta \cup \mathtt{A}$;

end_for;

$\mathrm{f}' := \mathrm{f}' \bowtie_{\mathbf{F}.\beta = \mathbf{F}_i.\alpha} \mathrm{f}(\mathbf{F}_i)$;

$\mathbf{F} := \mathbf{F} \cup \mathbf{F}_i$;

$\alpha := \emptyset$; $\beta := \emptyset$;

$i := i + 1$;

end_while;

Then, $\sigma_P(\mathrm{f}) = \sigma_P(\mathrm{f}') = \{fi | fi \in \mathrm{f}' \wedge P(fi) \text{ is true}\}$.

Example 6.3: Consider the folder organization presented in Figure 9 and the query: *List the PhD students who were accepted in the Fall of 1989 and have passed the Qualifying Examination in or before the Spring of 1991.*

Let P_1 = `SemesterTaken` $\leq$ *Spring 1991* $\wedge$ `SemesterAccepted` = *Fall 1989*. Let P_2 = **PhDAcceptLetter**.`Receiver` = **PhDQEResult**.`Receiver`.

$\pi_{\texttt{Receiver}}(\sigma_{P_1}(\text{PhDStds}(\textbf{PhDAcceptLetter}) \bowtie_{P_2} \text{PhDStds}(\textbf{PhDQEResult})))$.

In this case there is no frame template of a frame instance in the folder PhD-Stds that contains both the attributes `SemesterTaken` and `SemesterAccepted`. Before performing the selection, we first perform a join operation on the folders PhD-Stds(**PhDAcceptLetter**) and PhDStds(**PhDQEResult**), (or, equivalently, the join operation is first applied on frame instances of different frame templates within the folder). □

Proposition 6.5: Let f be a folder and let P be a predicate clause.

(i) If P is $(\neg P')$, then $\sigma_P(\mathsf{f}) = \mathsf{f} - \sigma_{P'}(\mathsf{f})$.

(ii) If P is $(P_1 \wedge P_2)$, then $\sigma_P(\mathsf{f}) = \sigma_{P_1}(\mathsf{f}) \cap \sigma_{P_2}(\mathsf{f})$.

(iii) If P is $(P_1 \vee P_2)$, then $\sigma_P(\mathsf{f}) = \sigma_{P_1}(\mathsf{f}) \cup \sigma_{P_2}(\mathsf{f})$.

The following proposition says that select operators commute.

Proposition 6.6: Let f be a folder and let P_1 and P_2 be two predicates. Then $\sigma_{P_1}(\sigma_{P_2}(\mathsf{f})) = \sigma_{P_2}(\sigma_{P_1}(\mathsf{f}))$.

The following asserts that select is distributive over the binary boolean operations $\cup, \cap$ and $-$ for folders.

Proposition 6.7: Let f_1 and f_2 be two folders. Let β be $\cup$, $\cap$, or $-$. Then, $\sigma_P(\mathsf{f}_1 \beta \mathsf{f}_2) = \sigma_P(\mathsf{f}_1) \beta \sigma_P(\mathsf{f}_2)$.

6.5 Class 5: Restructuring Operators

As suggested by the non-first-normal-form (NF^2) relational algebra [1, 4, 28, 54, 67, 74], there are two operators that are useful in dealing with set-valued attributes: *nest* and *unnest*. Consider the folder f containing the three frame instances shown in Figure 12.

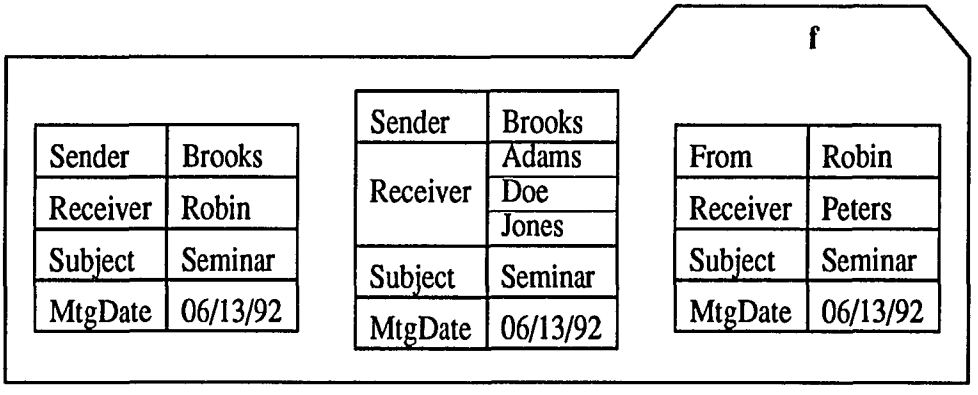


Figure 12: A folder containing three frame instances

Then *nesting* f over attribute Receiver, denoted $\eta_{\texttt{Receiver}}(\mathrm{f})$, yields a new folder g, containing two frame instances shown in Figure 13. Thus, the remaining attributes (excluding the attribute Receiver) of the frame instances in the folder must be identical in order for them to be nested within the folder; that is, the frame instances must be of the same type.

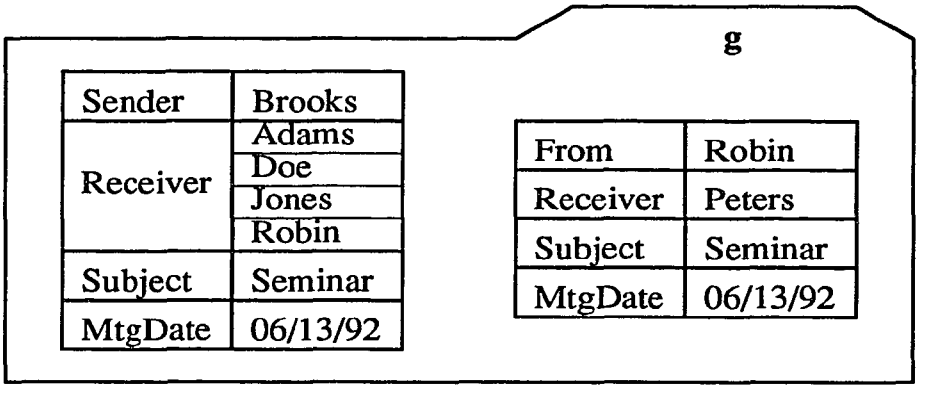


Figure 13: Result of nesting the folder of Figure 12 *over* attribute Receiver

The precise syntax and semantics of the *nest* operator are given below (recalling that f(**F**) represents the subset of frame instances in f over the same template **F**):

Definition 6.8: Let f be a folder and let A be the attribute *over* which η is to be applied. Let $\mathcal{F} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n\}$ be the set of frame templates associated with f. For each $\mathbf{F} = \{\mathrm{A}_1(\mathrm{T}_1), \mathrm{A}_2(\mathrm{T}_2), \ldots, \mathrm{A}_k(\mathrm{T}_k)\} \in \mathcal{F}$ where there exists A_j for some $j, 1 \leq j \leq k$ such that $\mathrm{A} = \mathrm{A}_j$, let $\beta = \mathbf{F} - \{\mathrm{A}_j(\mathrm{T}_j)\}$. For each $w \in \pi_\beta(\mathrm{f}(\mathbf{F}))$, let $\mathrm{V}_w = \{fi[\mathrm{A}_j] | fi \in \mathrm{f}(\mathbf{F}) \wedge fi(\beta) = w\}$.[15] (Recall that $fi[\mathrm{A}_j]$ represents the value V in the

[15] V_w is the set of all the A-values of the frame instances in f(**F**) that agree on the attributes in β.

pair $< \mathtt{A}_j, V >$ of *fi*.) Define $\eta_{\mathtt{A}}(\mathsf{f}(\mathbf{F})) = \{t | t$ is a frame instance over $\mathbf{F}, t(\beta) = w$ for some $w \in \pi_{\beta}(\mathsf{f}(\mathbf{F})), t[\mathtt{A}_j] = \mathsf{V}_w\}$. Then the semantics of the nest operator is given by

$$\eta_{\mathtt{A}}(\mathsf{f}) = \bigcup_{\mathbf{F} \in \mathcal{F}} (\eta_{\mathtt{A}}(\mathsf{f}(\mathbf{F}))).$$

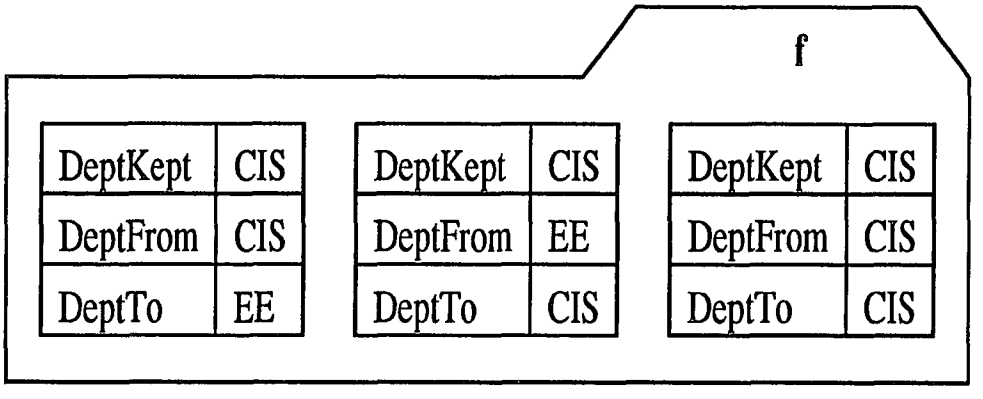
The following states that nest operators do not necessarily commute.

Proposition 6.8: The following equality does not always hold:

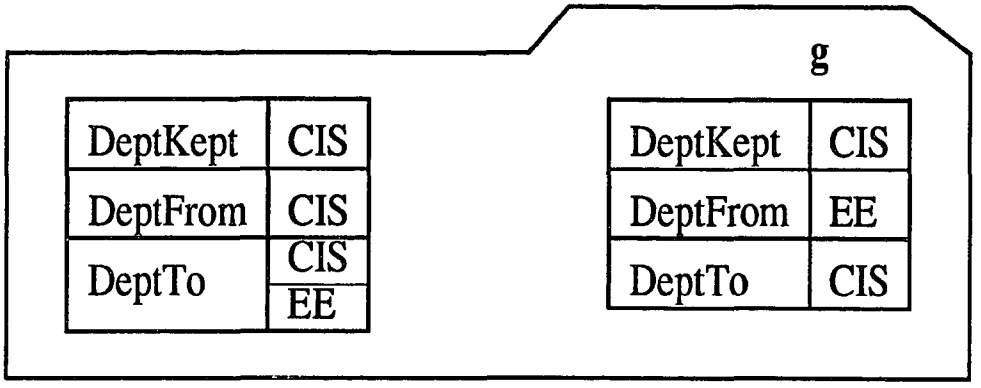
$$\eta_{\mathtt{A}_1} \eta_{\mathtt{A}_2}(\mathsf{f}) = \eta_{\mathtt{A}_2} \eta_{\mathtt{A}_1}(\mathsf{f})$$

for any folder f and two attributes $\mathtt{A}_1$ and $\mathtt{A}_2$ in the frame templates associated with f.

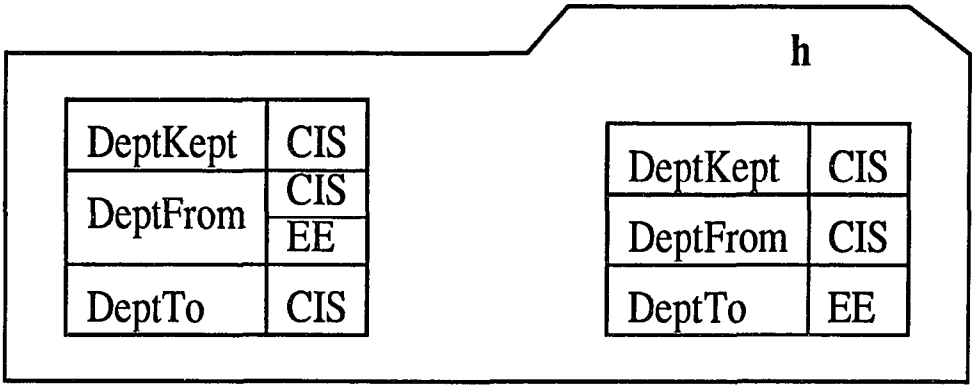
Proof: Consider the following folder f containing information about memos kept in the CIS department.



The result of $\eta_{\texttt{DeptFrom}}\eta_{\texttt{DeptTo}}(\mathsf{f})$ yields the folder g =

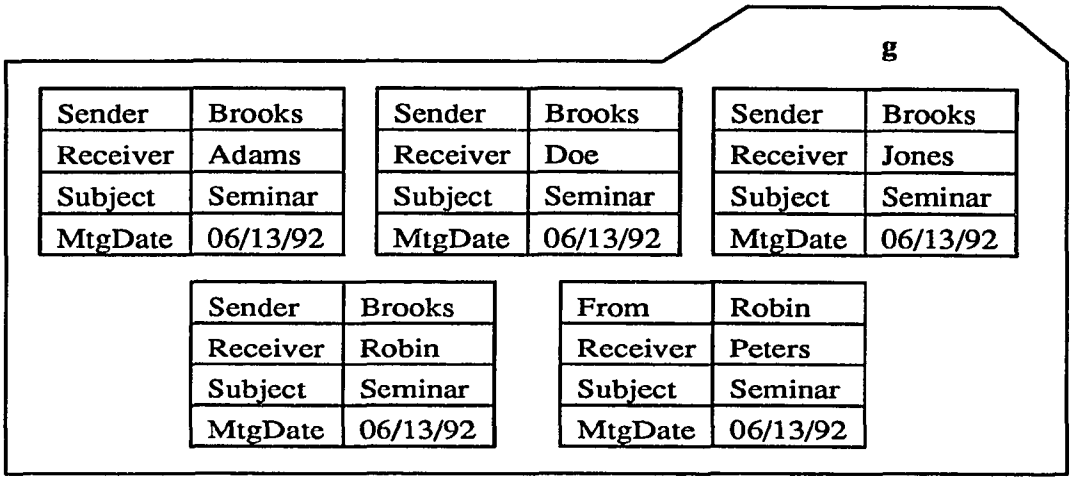


The result of $\eta_{\texttt{DeptTo}}\eta_{\texttt{DeptFrom}}(\mathsf{f})$ yields the folder h =



Clearly, g $\neq$ h. □

The *unnest* (μ) operator, when applied to a frame instance that is nested over some attribute A, will generate a set of frame instances that are flat over attribute A. For example, consider again the folder f in Figure 12. Then *unnesting* f *over* attribute Receiver, denoted $\mu_{\texttt{Receiver}}(\texttt{f})$, yields a new folder g =



The precise syntax and semantics of the unnest operator are given below.

Definition 6.9: Let f be a folder and let A be the attribute *over* which μ is to be applied. Let $fi \in \texttt{f}$ be a frame instance over $\mathbf{F} = \{< \texttt{A}_1(\texttt{T}_1), \texttt{A}_2(\texttt{T}_2), \ldots, \texttt{A}_k(\texttt{T}_k) >\}$. We define $\mu_{\texttt{A}}(\{fi\})$ by considering the following two cases:

1. $\texttt{A} = \texttt{A}_j$ for some $j, 1 \leq j \leq k$. Let $\beta = \mathbf{F} - \{\texttt{A}(\texttt{T})\}$. Then $\mu_{\texttt{A}}(\{fi\}) = \{t | t$ is a frame instance over $\mathbf{F}, t[\texttt{A}] \in fi[\texttt{A}] \wedge\ t(\beta) = fi(\beta)\}$.

2. $\texttt{A} \neq \texttt{A}_j$ for all j (i.e., A is not fi's attribute). Then $\mu_{\texttt{A}}(\{fi\}) = \{fi\}$;

The semantics of the unnest operator is given by

$$\mu_{\mathbf{A}}(\mathsf{f}) = \bigcup_{fi \in \mathsf{f}} (\mu_{\mathbf{A}}(\{fi\})).$$

The following asserts that unnest is distributive over the union operator for folders.

Proposition 6.9: Let f_1 and f_2 be two folders. Then, $\mu_{\mathbf{A}}(\mathsf{f}_1 \cup \mathsf{f}_2) = \mu_{\mathbf{A}}(\mathsf{f}_1) \cup \mu_{\mathbf{A}}(\mathsf{f}_2)$.

Proof: $\mu_{\mathbf{A}}(\mathsf{f}_1) \cup \mu_{\mathbf{A}}(\mathsf{f}_2) = \mathsf{f}'$

$= \{t | (\exists fi \in \mathsf{f}_1)(fi \text{ is } over \text{ a frame template } \mathbf{F})\ (t(\mathbf{F} - \mathbf{A}) = fi(\mathbf{F} - \mathbf{A}) \wedge t[\mathbf{A}] = fi[\mathbf{A}])\} \cup$

$\{t | (\exists fi \in \mathsf{f}_2)(fi \text{ is } over \text{ a frame template } \mathbf{F})(t(\mathbf{F} - \mathbf{A}) = fi(\mathbf{F} - \mathbf{A}) \wedge t[\mathbf{A}] = fi[\mathbf{A}])\}.$

$\mathsf{f}' = \{t | (\exists fi \text{ such that } (fi \in \mathsf{f}_1 \vee fi \in \mathsf{f}_2)) \wedge (t(\mathbf{F} - \mathbf{A}) = fi(\mathbf{F} - \mathbf{A}) \wedge t[\mathbf{A}] = fi[\mathbf{A}])\}$

$= \mu_{\mathbf{A}}(\mathsf{f}_1 \cup \mathsf{f}_2).$ □

The following states that unnest operators commute.

Proposition 6.10: $\mu_{\mathbf{A}_1} \mu_{\mathbf{A}_2}(\mathsf{f}) = \mu_{\mathbf{A}_2} \mu_{\mathbf{A}_1}(\mathsf{f})$ for any folder f and two attributes $\mathbf{A}_1$ and $\mathbf{A}_2$ in the templates associated with f.

Proof: This property follows readily from the definition of *nest*: $\mu_{\mathbf{A}_2} \mu_{\mathbf{A}_1}(\mathsf{f})$

$= \mu_{\mathbf{A}_2} (\{(t | (\exists fi \in \mathsf{f})(t(\mathbf{F} - \mathbf{A}_1) = fi(\mathbf{F} - \mathbf{A}_1 \wedge t[\mathbf{A}_1] \in fi[\mathbf{A}_1])\})$

$= \{t' | \exists t \in \{(t | (\exists fi \in \mathsf{f})(t(\mathbf{F} - \mathbf{A}_1) = fi(\mathbf{F} - \mathbf{A}_1 \wedge t[\mathbf{A}_1] \in fi[\mathbf{A}_1])\} |\ t'(\mathbf{F} - \mathbf{A}_2) =$

$t(\mathbf{F} - \mathtt{A}_2 \wedge t'[\mathtt{A}_2] \in t[\mathtt{A}_2])\}$

$= \{t' | \exists t \in \{(t|(\exists fi \in \mathsf{f})(t(\mathbf{F} - \mathtt{A}_2) = fi(\mathbf{F} - \mathtt{A}_2 \wedge t[\mathtt{A}_2] \in fi[\mathtt{A}_2])\}|\ t'(\mathbf{F} - \mathtt{A}_1) =$

$t(\mathbf{F} - \mathtt{A}_1 \wedge t'[\mathtt{A}_1] \in t[\mathtt{A}_1])\}$

$= \mu_{\mathtt{A}_1}\ (\{(t|(\exists fi \in \mathsf{f})(t(\mathbf{F} - \mathtt{A}_2) = fi(\mathbf{F} - \mathtt{A}_2 \wedge t[\mathtt{A}_2] \in fi[\mathtt{A}_2])\})$

$= \mu_{\mathtt{A}_1}\mu_{\mathtt{A}_2}(\mathsf{f})$. □

The following says that projection commutes with unnest.

Proposition 6.11: Let f be a folder and let A be the attribute over which μ is to be applied. Let $\mathcal{S} = \{\mathtt{A}_1, \ldots, \mathtt{A}_k\}$ where $\mathtt{A}_j$ is an attribute of type $\mathrm{T}_j \in \mathcal{T}$. Then, $\pi_{\mathcal{S}}(\mu_{\mathtt{A}}(\mathsf{f})) = \mu_{\mathtt{A}}(\pi_{\mathcal{S}}(\mathsf{f}))$.

Proof: If $\mathtt{A} \notin \mathcal{S}$, the result is trivially true.

Otherwise, note that for any $fi \in \mathsf{f}$, $\pi_{\mathcal{S}}(\mu_{\mathtt{A}}(\{fi\})) = \mu_{\mathtt{A}}(\{fi(\mathcal{S})\})$. Thus,

$\pi_{\mathcal{S}}(\mu_{\mathtt{A}}(\mathsf{f}))$

$= \pi_{\mathcal{S}} \bigcup_{fi \in \mathsf{f}}(\mu_{\mathtt{A}}(\{fi\}))$ (by definition 6.9)

$= \bigcup_{fi \in \mathsf{f}} \pi_{\mathcal{S}}(\mu_{\mathtt{A}}(\{fi\}))$ (by proposition 6.4)

$= \bigcup_{fi \in \mathsf{f}} \mu_{\mathtt{A}}(\{fi(\mathcal{S})\})$

$= \bigcup_{fi \in \mathsf{f}} \mu_{\mathtt{A}}(\{fi(\mathcal{S})\})$

$= \mu_{\mathtt{A}}(\bigcup_{fi \in \mathsf{f}} \{fi(\mathcal{S})\})$ (by proposition 6.9)

$= \mu_{\mathtt{A}}(\pi_{\mathcal{S}}(\mathsf{f}))$. □

The following proposition says that if a frame instance fi is unnested over attribute A and then the resulting frame instances are nested over the same attribute (or visa versa), the result is $\{fi\}$.

Proposition 6.12: Let fi be a frame instance over $\mathbf{F}$ and let A be the attribute over which μ and η are applied. Then

(i) $\eta_{\mathtt{A}}\mu_{\mathtt{A}}(\{fi\}) = \{fi\}$;

(ii) $\mu_{\mathtt{A}}\eta_{\mathtt{A}}(\{fi\}) = \{fi\}$.

Proof: We show (i) only. The result follows immediately by observing that $\eta_{\mathtt{A}}(\mu_{\mathtt{A}}(\{fi\}))$ $= \eta_{\mathtt{A}}(\{t|t$ is a frame instance over $\mathbf{F}, t[\mathtt{A}] \in fi[\mathtt{A}] \wedge\ t(\mathbf{F} - \{\mathtt{A}\}) = fi(\mathbf{F} - \{\mathtt{A}\})\}) = \{fi\}$.□

However, the same result does not hold for folders, as indicated in the following proposition.

Proposition 6.13: There exists a folder f and a nested attribute A in some template associated with f such that

(i) $\eta_{\mathtt{A}}\mu_{\mathtt{A}}(\mathsf{f}) \neq \mathsf{f}$;

(ii) $\mu_{\mathtt{A}}\eta_{\mathtt{A}}(\mathsf{f}) \neq \mathsf{f}$.

Proof: We show (i) only. (ii) can be proved similarly. Consider the `Receiver`

attribute in the folder of Figure 12. $\mu_{\texttt{Receiver}}(\mathsf{f}) =$

g

Sender	Brooks
Receiver	Adams
Subject	Seminar
MtgDate	06/13/92

Sender	Brooks
Receiver	Doe
Subject	Seminar
MtgDate	06/13/92

Sender	Brooks
Receiver	Jones
Subject	Seminar
MtgDate	06/13/92

Sender	Brooks
Receiver	Robin
Subject	Seminar
MtgDate	06/13/92

From	Robin
Receiver	Peters
Subject	Seminar
MtgDate	06/13/92

Applying $\eta_{\texttt{Receiver}}$ to the above result gives

h

Sender	Brooks
Receiver	Adams
	Doe
	Jones
	Robin
Subject	Seminar
MtgDate	06/13/92

From	Robin
Receiver	Peters
Subject	Seminar
MtgDate	06/13/92

which is not equal to f. □

6.6 Class 6: Path Notation Operator

Recall that a composite attribute A is an attribute which contains a collection of attributes $\{B_1, B_2, \ldots, B_m\}$ for some integer m where B_i is of type $T_i \in \mathcal{T}$, $1 \leq i \leq m$. Each of the B_i's is a component attribute of A. Such *aggregation* [90] (i.e., an attribute is a collection of other attributes) allows the user to either gradually decompose objects into their detailed components or to aggregate them into higher-level objects. Often, *path-notation* [9, 82, 86] is used in referencing values of particular components of composite attributes. If the aggregation hierarchy becomes very deep, path-notation becomes tedious. For example, in order to reference the values for the attributes `MtgDate` and `MtgPlace` of the frame instance shown in Figure 6, two path-notations are needed (one for each attribute). The path-notation for `MtgDate` is `MtgDescription.MtgDay.MtgDate`. The path-notation for `MtgPlace` is `MtgDescription.Place`. Instead of using path-notation, a powerful operator called *highlight* is introduced which allows users to highlight certain components of composite attributes directly without specifying any paths.

Let $fi = \{< A_1, V_1 >, \ldots, < A_i, V_i >, \ldots, < A_l, V_l >\}$ be a frame instance. Suppose A_i is a composite attribute. Let β be a subset of the descendant attributes of A_i. The *minimal cover* of β, denoted by β_{min}, is defined to be a subset of β (i.e., $\beta_{min} \subseteq \beta$) such that:

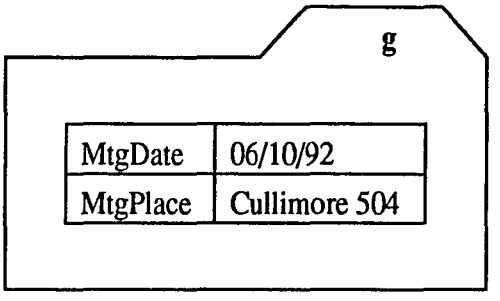
1. every element in $\beta - \beta_{min}$ is a descendant of an element in β_{min} and,
2. no element of β_{min} is a descendant of any other element in β_{min}.

The β_{min} is well-defined because there exists a unique subset that satisfies the conditions 1 and 2 above. The β-value of fi with respect to $\mathtt{A}_i$, denoted by $fi_{\mathtt{A}_i}(\beta)$, is the frame instance $\{< \mathtt{B}_j, W_j > | \mathtt{B}_j \in \beta_{min}, W_j \subseteq dom(\mathtt{B}_j)$ is the component value of $\mathtt{B}_j$ in $fi[\mathtt{A}_i], 1 \leq j \leq |\beta_{min}|\}$.

Example 6.4: Refer to the frame instance shown in Figure 6. Consider the attribute `MtgDescription`. Then each of the attributes `MtgDay`, `MtgPlace`, `Synopsis`, `MtgDate`, and `MtgTime` is a descendant attribute of `MtgDescription`. Let β be the subset {`MtgDay`, `MtgTime`, `MtgPlace`}. Then β_{min} is the subset {`MtgDay`, `MtgPlace`}. □

Definition 6.10: Let f be a folder and let A be a composite attribute of type T. Let β contain any subset of the descendant attributes of A. Then *highlighting* f on attribute A *onto* β, denoted by $\gamma_{\mathtt{A}_\beta}(\mathtt{f})$, yields the folder $\{fi_{\mathtt{A}}(\beta) | fi \in \mathtt{f}\}$.

Example 6.5: Suppose that there is a folder f containing the frame instance in Figure 6. Then $\gamma_{\mathtt{MtgDescription}_{\{\mathtt{MtgDate},\mathtt{MtgPlace}\}}}(\mathtt{f})$, returns a folder g =



□

6.7 Class 7: Aggregate Operators

Class 7 includes five aggregate operators: **count**, **sum**, **avg**, **min**, and **max**. These operators take a set of frame instances (a folder) as an argument and produce a single value as a result. Their syntax and semantics are described as follows.

Definition 6.11: Let f be a folder and let $\mathcal{F} = \{\mathbf{F}_1, \ldots, \mathbf{F}_n\}$ be a set of templates associated with f. The syntax for an aggregate operator o on attribute A of f is $o_{\mathtt{A}}(\mathtt{f})$, where $\mathtt{A} \in \mathbf{F}_j$ for some $j, 1 \leq j \leq n$. (Except for the **count** operator, the domain of A must be reals or integers.) Let S be the frame instances fi in the folder f where fi has the attribute A. (Recall that $fi[\mathtt{A}]$ represents the value V in the pair $< \mathtt{A}, V >$ of fi.) The semantics of the five operators are given below.

1. $\mathsf{count}_{\mathtt{A}}(\mathtt{f}) = |S|$.

2. $\mathsf{sum}_{\mathtt{A}}(\mathtt{f}) = \sum_{fi \in S} fi[\mathtt{A}]$ if $|S| > 0$, or $\mathsf{sum}_{\mathtt{A}}(\mathtt{f}) = 0$ if $|S| = 0$.

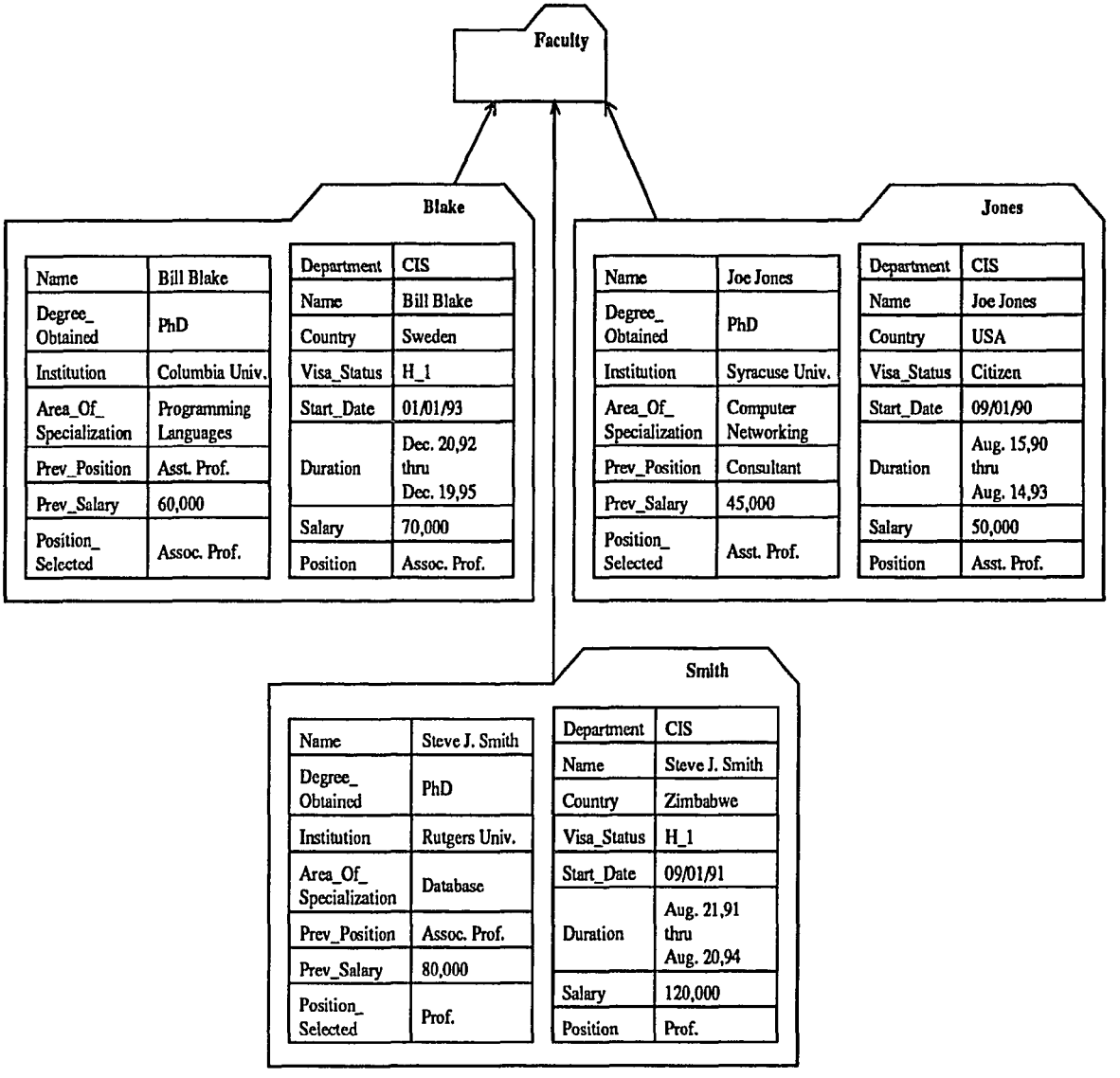
3. $\mathsf{avg}_{\mathtt{A}}(\mathtt{f}) = (1/|\mathtt{S}|)\sum_{fi \in S} fi[\mathtt{A}]$ if $|S| > 0$, or $\mathsf{avg}_{\mathtt{A}}(\mathtt{f})$ is undefined if $|S| = 0$.

4. $\mathsf{max}_{\mathtt{A}}(\mathtt{f}) = max_{fi \in S} fi[\mathtt{A}]$ if $|S| > 0$.

5. $\mathsf{min}_{\mathtt{A}}(\mathtt{f}) = min_{fi \in S} fi[\mathtt{A}]$ if $|S| > 0$, or $\mathsf{min}_{\mathtt{A}}(\mathtt{f})$ is undefined if $|S| = 0$.

An aggregate can be calculated independently from the rest of the query and then simply replaced by its value.

Example 6.6: Consider the folder organization shown below.



Then, $count_{\texttt{Salary}}(\text{Faculty}) = 3.$ $sum_{\texttt{Salary}}(\text{Faculty}) = 240{,}000.$

$avg_{\texttt{Salary}}(\text{Faculty}) = 80{,}000.$ $max_{\texttt{Salary}}(\text{Faculty}) = 120{,}000.$

$min_{\texttt{Salary}}(\text{Faculty}) = 50{,}000.$ □

We now conclude this chapter by formally defining $\mathcal{D}$_algebra.

Definition 6.12: The $\mathcal{D}$_algebra *over* $\mathcal{T}, \mathcal{D}$, *dom*, $\mathcal{D}_t\mathcal{H}, \mathcal{FO}$, and Θ is the 7-tuple $\mathcal{DA} = < \mathcal{T}, \mathcal{D}$, *dom*, $\mathcal{D}_t\mathcal{H}, \mathcal{FO}$, $\Theta, \mathrm{O} >$ where:

- $\mathcal{T} = \{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_n\}$ is a finite set of attribute types;
- $\mathcal{D} = \{\mathrm{D}_1, \mathrm{D}_2, \ldots, \mathrm{D}_l\}$ is a finite set of domains;
- *dom* is a total function that associates each atomic attribute type $\mathrm{T} \in \mathcal{T}$ with a domain *dom*(T) in $\mathcal{D}$;
- $\mathcal{D}_t\mathcal{H}$ is the document type hierarchy comprising a finite set of frame templates;
- $\mathcal{FO}$ is the folder organization comprising a finite set of folders;
- $\Theta = \{<, \leq, >, \geq, =, \neq, \subset, \subseteq, \supset, \supseteq, \in, \notin\}$ is a set of comparators *over* domains in $\mathcal{D}$;
- O is the set of operators $\cup, \cap, -, \pi, \eta, \mu, \bowtie, \rho, \gamma$, count, sum, avg, min, and max using attributes of attribute types in $\mathcal{T}$, σ using comparators in Θ, and the logical connectives $\wedge, \vee, \neg$.

CHAPTER 7

EVALUATING THE $\mathcal{D}$_ALGEBRA UNDER THE GIVEN FORMAL MODEL

A reasonable question to ask at this point is the following: *Based on the query algebra, will TEXPROS behave according to the user's perceptions in retrieving information from the user's personal filing system, under the proposed formal model?* Restated, the question is: *Is the $\mathcal{D}$_Algebra powerful enough to express all queries of interest for TEXPROS?* An answer to this question immediately calls for an evaluation of the $\mathcal{D}$_algebra.

An integral part of this research is the examination of the algebra's expressive power. The relational algebra and calculus of Codd [19] are often used as models of a query language, i.e., each of them is considered as a minimum set of operations that every query language should have. They were proposed in [20] as a reference for measuring the "completeness" of query languages. The completeness notion has indeed attracted many discussions. Paredaens [75] and Bancilhorn [6] present a formal study of the relational algebra that characterizes completeness. Paredaens shows that if $\mathcal{U}$ is the union of all domains of relations of a given database, the relational algebra is complete in the sense that it can express all relations that can be defined on $\mathcal{U}$. Bancilhorn points out that the output of queries should preserve symmetry among the elements in the database. That is, a mapping which is an automorphism[16]

[16]An automorphism is a renaming of the values in the active domain that leaves the database instance invariant [5].

on the database should leave the output unchanged. Since then, the question of the expressive power of the relational (and extended relational) algebra has been addressed (e.g., [3, 14, 40, 50]).

This chapter addresses the question of the expressive power of $\mathcal{D}$_algebra relative to its operation on the objects of the $\mathcal{D}$_model. We show that a subset of the $\mathcal{D}$_algebra is more expressive than the relational algebra. A formal definition of the relational algebra is given first in section 7.1. In section 7.2, we prove that the $\mathcal{D}$_algebra is as expressive as the relational algebra. Examples are given to show what the $\mathcal{D}$_algebra can do and the relational algebra cannot do in section 7.3. Section 7.4 discusses some significant differences between the $\mathcal{D}$_model and the relational model.

7.1 The Relational Algebra

Definition 7.1[66, 96]: Let $\mathsf{U} = \{\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_u\}$ be a finite universal set of attributes. Let D be a finite set of domains and let *dom* be a total function from U to D. Let $\mathsf{R} = \{R_1, R_2, \ldots, R_p\}$ be a finite set of distinct relation schemes, where $R_i \subseteq \mathsf{U}, 1 \leq i \leq p$. Let $d = \{r_1, r_2, \ldots, r_p\}$ be a finite set of relations such that r_i is a relation on $R_i, 1 \leq i \leq p$. The *relational algebra over* U,D,*dom*,R,d,Θ is the 7-tuple $\mathcal{R} = < \mathsf{U}, \mathsf{D}, dom, \mathsf{R}, d, \Theta, O >$ where $\Theta = \{<, \leq, >, \geq, =, \neq\}$ is a set of comparators *over* domains in D, and O is the set of operators union,difference,cartesian product, project, and renaming using attributes in U, and select using comparators in Θ.

7.2 Reducing Relational Algebra to $\mathcal{D}$_Algebra

In this section we define a subset of the $\mathcal{D}$_algebra and show that it is at least as expressive as the relational algebra in a restricted sense. The $\mathcal{D}$_algebra is restricted to the subset $\mathcal{DA}^-$ consisting of *renaming, union, difference, cartesian product, project,* and *select.* The *unnest* and *nest* are excluded because relational algebra expressions only reference single-valued attributes. Unlike the relational algebra, used in modeling enterprises, the $\mathcal{D}$_algebra is mainly for office environments. Thus, we will only focus on schemes that are meaningful in such a domain.

Definition 7.2: A set $\mathcal{S}$ of attributes is meaningful if there exists a folder organization $\mathcal{FO}$ such that for any $\mathtt{A} \in \mathcal{S}$, A belongs to a frame template **F** of some folder in $\mathcal{FO}$.

Theorem 7.1: If E^R is a relational algebra expression against scheme **R** over a meaningful set $\mathcal{S}$ of attributes, then there exists an equivalent expression E^D in $\mathcal{DA}^-$ against scheme **F** over $\mathcal{S}$.

Proof: The proof is by induction on the operators in E^R. In the basis step, we construct an equivalent $\mathcal{D}$_model schema **F** and demonstrate the case of 0 operators. In the induction step, we provide a translation for each operator of the relational algebra.

1. *Basis Step:* E^R has 0 operators. This is straightforward. We construct a $\mathcal{D}$_model schema **F** from the relation scheme **R** as follows. Each relation scheme **R** becomes a frame template **F** where attributes of **F** are atomic and single-valued. Each folder is on a single frame template.

2. *Induction Step:* Assume the theorem holds for any relational algebra expression with fewer than k operators. Let E^R have k operators. We provide a translation for each operator of relational algebra, and construct E^D from E^R as follows:

   (a) *(attribute renaming)*: $E^R = \delta_{A_1,A_2,\ldots,A_m \leftarrow B_1,B_2,\ldots,B_m}(E_1)$. E_1 has less than k operators. Then $E^D = \rho_{A_1,A_2,\ldots,A_m \leftarrow B_1,B_2,\ldots,B_m}(E_1')$, where E_1' denotes a $\mathcal{D}$_algebra expression that is equivalent to the relational algebra expression E_1.

   (b) *(union)*: $E^R = E_1 \cup E_2$ is equivalent to $E_1' \cup E_2'$ where E_i' denotes a $\mathcal{D}$_algebra expression that is equivalent to relational algebra expression E_i.

   (c) *(difference)*: $E^R = E_1 - E_2$ is equivalent to $E_1' - E_2'$ where E_i' denotes a $\mathcal{D}$_algebra expression that is equivalent to relational algebra expression E_i.

   (d) *(cartesian product)*: $E^R = E_1 \times E_2$ is equivalent to $E_1' \times E_2'$ where E_i' denotes a $\mathcal{D}$_algebra expression that is equivalent to relational algebra expression E_i.

   (e) *(projection)*: $E^R = \pi_{A_1,A_2,\ldots,A_n}(E_1)$ is equivalent to $\pi_{A_1,A_2,\ldots,A_n}(E_1')$.

   (f) *(selection)*: $E^R = \sigma_P(E_1)$ is equivalent to $\sigma_P(E_1')$. □

7.3 Examples showing the Expressiveness of the $\mathcal{D}$_Algebra over the Relational Algebra

Note that since the $\mathcal{D}$_model doesn't have the notion of keys, foreign keys, functional dependencies, or referential integrity, the above translation becomes quite straightforward. Also note that the reverse direction of the above theorem does not hold, because in general when frame templates are mapped to *first normal form* (1NF) relation schemas[17], they may have certain undesirable properties. The following examples illustrate some problems that may arise when transforming folders into relations.

Example 7.1 shows how transforming folders into relations may constitute a bad relational database design.

Example 7.1: Consider the folder Jones shown in Figure 14. This folder is associated with two frame templates: **PhDQEApplicationForm** and **PhDQEResult**. In the folder, there is one frame instance for Jones' qualifying examination application form. This frame instance is of the type **PhDQEApplicationForm**. There are two frame instances of the type **PhDQEResult**, one for the first time that he sat for the exam in Fall 1992 and passed conditionally and the other for the second time that he took it in Spring 1993 and passed. Note that Joe Jones' address had changed by Spring 1993.

[17]A relation schema $\mathbf{R} = \{A_1, A_2, \ldots, A_n\}$ is in *first normal form* if every attribute $A_i \in \mathbf{R}$ is atomic.

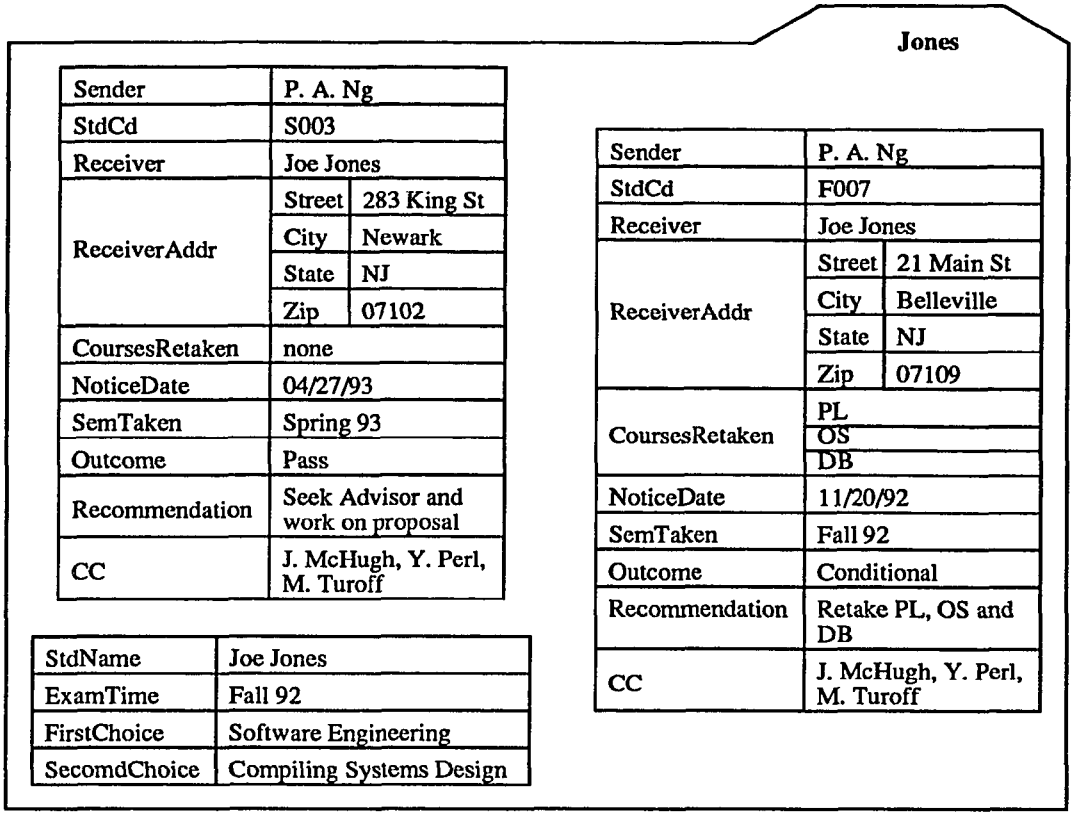


Figure 14: A qualifying examination folder for the Phd student Jones

Figure 15 shows two 1NF relations that are a result of transforming the folder Jones into the relational model. In this transformation, every composite attribute is broken down into its indivisible atomic attributes and the value of any attribute in a tuple is a single value from the domain of that attribute. Hence, the attribute `ReceiverAddr` is broken down into the attributes `Street`, `City`, `State` and `Zip`, and, the nested attribute `CoursesRetaken` is flattened so that its value is always a single value for any tuple in the resulting relation.

Jones_PhD_QE_Result

Sender	StdCd	Receiver	Street	City	State	Zip	---	CoursesRetaken	---	CC
P. A. Ng	F007	Joe Jones	21 Main St	Belleville	NJ	07109	---	PL	---	J. McHugh, Y. Perl, M. Turoff
P. A. Ng	F007	Joe Jones	21 Main St	Belleville	NJ	07109	---	OS	---	J. McHugh, Y. Perl, M. Turoff
P. A. Ng	F007	Joe Jones	21 Main St	Belleville	NJ	07109	---	DB	---	J. McHugh, Y. Perl, M. Turoff
P. A. Ng	S003	Joe Jones	283 King St	Newark	NJ	07102	---	none	---	J. McHugh, Y. Perl, M. Turoff

Jones_PhD_QE_Application_Form

StdName	ExamTime	FirstChoice	SecondChoice
Joe Jones	Fall 92	Software Engineering	Compiling Systems Design

Figure 15: A transformation of the folder Jones into two 1NF relations

This transformation constitutes a bad relational database design. Consider the relation Jones_PhD_QE_Result. We can see some problems here:

1. *Redundancy*: The address of the student (receiver in this case) is repeated for each tuple. If the attribute CC was nested (i.e., if it contained a set of three faculty members instead of string characters) then this relation would contain twelve tuples!

2. *Potential inconsistency*:

   (a) *Update anomalies*: Since there is an update operation in the relational model, one may want to update the address in the first three tuples in order to maintain consistency on the address. However, the user may

inadvertently update the address only in one of the tuples while leaving it fixed in the other tuples. This leaves the database stating the fact that Jones had different addresses within the same semester.

(b) *Deletion anomalies*: If we delete all of the courses retaken by the student from the relation and only keep the tuple that shows the final pass outcome for the student, we may inadvertently leave some of the tuples in the relation (i.e., we did not completely delete the first three tuples throughout the case). □

Example 7.2 illustrates that the $\mathcal{D}$-algebra can capture certain semantic meaning which cannot be captured by the relational algebra according to the user's perception and the way information is stored.

Example 7.2: Consider the folder Memo shown in Figure 16. The folder contains four memos that were written on the same date. There are two *Call for P&T meeting* memos; one written to Bill Blake and the other *copied* to three persons as the receivers of this memo. One of the memos is a legal document written to John Smith regarding his termination from the PhD program. The last memo, copied to two persons, regards a book chapter publication deadline.

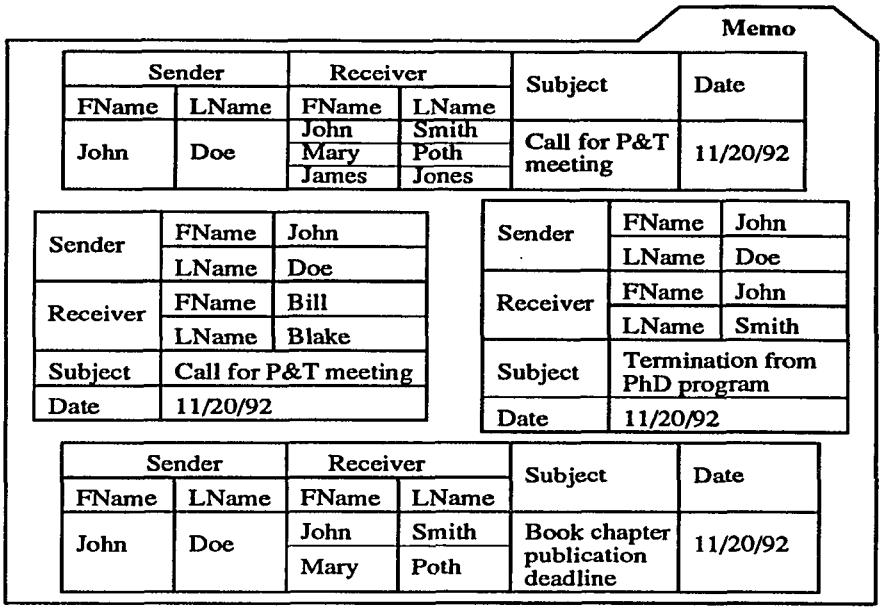


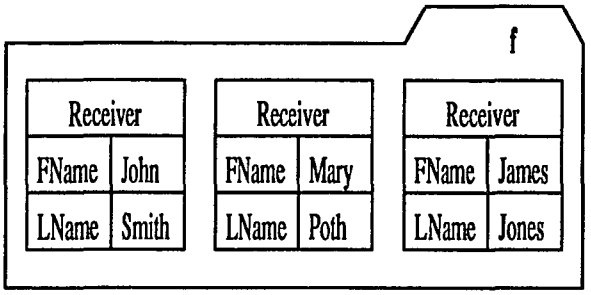
Figure 16: A folder Memo containing four memos

Assume that John Doe remembers that he copied a memo to John Smith and Mary Poth regarding a call for P&T meeting and that he also wrote the same memo to Bill Blake. However, he thinks that he may not have reminded all the members who are on the P&T committee about the meeting. He knows that there could be other persons, who as a group, received the same memo that he copied to John Smith and Mary Poth. He needs to know who these persons are so he can send the memo to the rest of the members that have not received it. John Doe's query, *Which other persons received the copy of the memo, regarding a call for P&T meeting, in which John Smith and Mary Poth are the receivers*, will be handled as follows in $\mathcal{D}$-algebra:

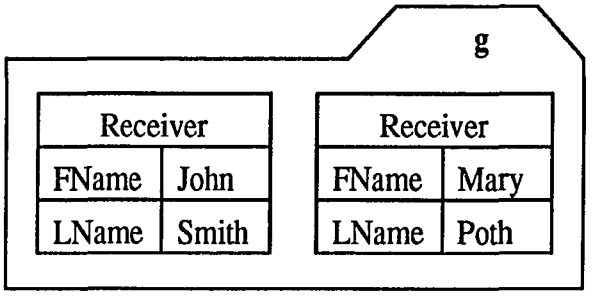
Let f =

$\mu_{\text{Receiver}}(\pi_{\text{Receiver}}(\sigma_{\text{Receiver}\supset\{\textit{John Smith,Mary Poth}\}\wedge\text{Subject}=\textit{Call for P\&T meeting}}(\text{Memo})))$

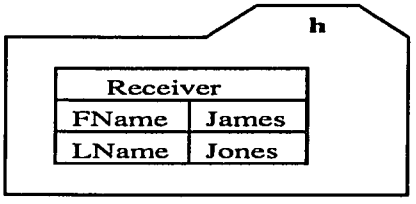
=



Let g = $\sigma_{\text{Receiver}=\textit{John Smith}\ \vee\ \text{Receiver}=\textit{Mary Poth}}(\text{f})$ =



The result is the folder h = f−g =



which is exactly what is needed.

Figure 17 shows a 1NF relation that is a result of transforming the folder Memo into the relational model. In this transformation, there is no connection between the receivers John Smith, Mary Poth and James Jones who, *as a group*, received the same copy of the memo. An independent tuple is constructed for each of these receivers.

Memo

SFName	SLName	RFName	FLName	Subject	Date
John	Doe	John	Smith	Call for P&T meeting	11/20/92
John	Doe	Mary	Poth	Call for P&T meeting	11/20/92
John	Doe	James	Jones	Call for P&T meeting	11/20/92
John	Doe	Bill	Blake	Call for P&T meeting	11/20/92
John	Doe	John	Smith	Termination from PhD program	11/20/92
John	Doe	John	Smith	Book chapter publication deadline	11/20/92
John	Doe	Mary	Poth	Book chapter publication deadline	11/20/92

Figure 17: A transformation of the folder Memo into a 1NF relation

The relational algebra cannot capture the semantic meaning that these three receivers are intended to be treated as a group and not individually. The same query cannot be expressed in relational algebra. □

In addition, the $\mathcal{D}$-algebra has more expressive power than the relational algebra because of additional operations such as aggregate functions.

Example 7.3: Let $\mathbf{R}= \{A_1, \ldots, A_i, \ldots, A_n\}$ be a relation schema. Assume that $dom(A_i) = Real$. Let r be a relation on $\mathbf{R}$. Then the aggregate operations such as $sum_{A_i}(r)$, $avg_{A_i}(r)$, and so forth, cannot be done since the relational algebra is not a *many-sorted*[18] algebra. Relational algebra requires all database objects, and thus all query inputs and outputs, to be sets. All relational algebra operations take one or two relations as input and produce a new relation as their result.

In the $\mathcal{D}$_algebra, aggregate operations can be treated inside the model and need not be introduced as a query language extension to the model. The algebra can serve as a high-level query language. Consequently, we could say that the $\mathcal{D}$_algebra is a practically usable query language with precisely defined semantics. □

7.4 Differences Between the $\mathcal{D}$_Model and the Relational Model

In light of evaluating the $\mathcal{D}$_algebra relative to the relational algebra, it is worthy to mention some significant differences between the $\mathcal{D}$_model and the relational model.

7.4.1 The Relational Model

The set-theoretic *relation* is the mathematical concept that underlies the relational model. A relation is any subset of the Cartesian product of one or more domains. The members of a relation are called *tuples.* A relation is viewed as a table, where

[18]In a many-sorted algebra, the algebraic structures are not necessarily of the same type (or *sort*)[98].

each row is a tuple and each column corresponds to one component. The columns are given names called *attributes.* The set of attribute names for a relation is called a *relation schema.* Each attribute A in a relation schema is the name of a role played by some domain D in the relation schema. D is called the domain of A and is denoted by $dom(\mathtt{A})$. If $\mathbf{R}$ is a relation schema comprising the set of attributes $\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_k$, then a relation r on $\mathbf{R}$, denoted by $r(\mathbf{R})$, is a finite set of k-tuples $\{t_1, t_2, \ldots, t_n\}$ where each k-tuple $t_i, 1 \leq i \leq n$, is *over* $\mathbf{R}$. (A tuple is considered as a set of <attribute,value> pairs, where each pair gives the value of the mapping from an attribute $\mathtt{A}_i$ to a value v_i from $dom(\mathtt{A}_i)$.) A *relational database schema* S is a set of relation schemas $S = \{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_m\}$ and a set of integrity constraints IC. A *relational database* (or *relational database instance*) DB of S is a set of relations $DB = \{r_1, r_2, \ldots, r_m\}$ such that r_i is a relation on $\mathbf{R}_i$, $1 \leq i \leq m$. The set IC of integrity constraints is introduced as a tool for improving the description of the application of interest by means of a relational database. Constraints of specific forms (keys, functional dependencies, referential integrity or subset dependencies) are used to evaluate the quality of a database schema and to suggest its decomposition.

The relational algebra is a collection of operations that are used to manipulate relations. The result of each operation is a new relation, which can be further manipulated by the relational algebra operations. The binary set theoretic operators *union* ($\cup$), *intersection* ($\cap$) and *difference* ($-$) of the relational algebra are applied only to union-compatible relations. The relational algebra omits set comparison operators in

the predicate clause of a selection operation. This is because a value v of attribute A is not considered as a subset of the domain of A but as an element of the domain of A.

7.4.2 The $\mathcal{D}$_Model

In contrast, the $\mathcal{D}$_model is a dual model which describes a user's document filing system as the tuple $< \mathcal{D}_t\mathcal{H}, \mathcal{FO} >$, which is an organization of the of the document type hierarchy and the folder organization. The $\mathcal{D}$_algebra includes a family of operators which together comprise the fundamental query language for the $\mathcal{D}$_model.

The Document Type Hierarchy

The $\mathcal{D}_t\mathcal{H}$ describes the frame templates and the is-a relationships among them. The concept of the frame template corresponds to the notion of the schema in the relational model, albeit the significant differences. Attributes in a frame template can either be atomic or composite. Ability to define composite attributes gives the designer flexibility of defining frame templates at different levels of detail. Besides, a value v of an attribute A is considered not as an element of the domain of A, but rather, as a subset of the domain of A. This enables the $\mathcal{D}$_model to capture the notion that a value can be treated as a *group* of elements and not always as an *individual* element.

The frame templates in the document type hierarchy are connected through the is-a relationship. Note that relation schemas of the relational model are connected

through the referential integrity constraint (or subset dependencies). The notion of referential integrity comes as a result of keys and functional dependencies. There is no notion of keys and functional dependencies in the $\mathcal{D}$_model.

The Folder Organization

The $\mathcal{FO}$ corresponds to the actual contents of folders and the depends-on relationships between them. Unlike a relation which is associated with precisely one schema, a folder can be associated with several frame templates. Intuitively, a folder can thus be viewed as a relational database. Formally, let $\mathcal{R}$=$\{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_n\}$ be a finite set of relation schemas. A folder can map into $\{r_1, r_2, \ldots, r_n\}$ where r_i is a relation on $\mathbf{R}_i \in \mathcal{R}$, $1 \leq i \leq n$. This is illustrated as follows. Consider a folder f consisting of several frame instances. Assume that f is associated with the finite set of frame templates $\mathcal{F} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n\}$. Then, each frame template $\mathbf{F}_i \in \mathcal{F}$ can map into a relation schema $\mathbf{R}_i, 1 \leq i \leq n$. Each frame instance *fi* over $\mathbf{F}_i$ can map into one or more tuples, each over relation schema $\mathbf{R}_i$, depending on whether there are nested attributes with respect to *fi*. Furthermore, the frame instances of the same frame template from different folders are mapped into different relations.

The folders in a folder organization are connected through the depends-on relationship. If the existence of a folder f depends on the existence of a folder g, then f is a subset of g. There is no such relationship among relations in the relational model.

The $\mathcal{D}$-algebra

The $\mathcal{D}$-algebra is a many-sorted algebra. It includes aggregate operators which map folders to numeric values. In the relational algebra, all the algebraic structures are of the same type (the *set*). Aggregate operators, which map a set into a numeric value, are not a part of the relational algebra.

One interesting phenomenon is the predicate clause of the selection operation. The predicate P can span several frame templates of a folder. This is of course due to the way we define a folder (as a heterogeneous set of frame instances). Recall that a folder can map into a relational database. A predicate P in the relational algebra is not applied to the relational database (a set of relations possibly on different relation schemas) but to a relation which is defined on precisely one schema. In addition, set comparison operators in a predicate clause of a relational algebraic operation would be omitted since set-valued attributes are disallowed in the relational model.

Note also that the definition of a folder automatically extends the notion of the set-theoretic operators $\cup, \cap$, and $-$. Here, they are not restricted to union-compatible sets of frame instances.

CHAPTER 8

A LOGICAL APPROACH TO THE $\mathcal{D}$_MODEL THEORY

A database can be considered from the viewpoint of logic in two different ways: either as a *model* of a theory or as a *theory* [5]. These two approaches are called *model-theoretic* and *proof-theoretic*, respectively. In [81], Reiter discusses the model-theoretic and proof-theoretic characterizations of the relational database theory. In the model-theoretic approach, the database is viewed as a particular kind of first-order interpretation and queries are first-order formulas to be evaluated with respect to this interpretation. Liu and Sunderraman [62] view the relational database through the model-theoretic perspective to address the problem of representing disjunctive and maybe information. A model-theoretic approach defines a three-valued logic [21], based on the truth-values (*true, false, unknown*) and denotes any occurrence of the value-unknown type, called a null value, by the special symbol ω. Figure 18 shows the truth tables for the three-valued logic.

X	NOT(X)
true	false
false	true
unknown	unknown

AND	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

OR	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

Figure 18: Tables for three-valued logic

The model-theoretic view of a database has been criticized by several authors [35, 99] since it does not distinguish between several unknown values. For example, two unknown values for a faculty member's date of birth and salary would be denoted by the symbol ω in some frame instance. Although these two unknown values are denoted by the same symbol, they cannot be treated as equal.

Reiter [80, 81] gives precise solutions to some of the problems of the model-theoretic paradigm on the basis of the proof-theoretic view of databases with null values. The proof-theoretic approach views the database as a set of well-formed formulas that constitute a first-order theory. In this approach, queries are first-order formulas that have to be proven given the database as premises. Reiter shows that the proof-theoretic approach provides clear formal semantics of null values. Here, the database is viewed as a class of first-order theories that incorporate the following assumptions:

1. The *domain closure assumption* [79]. The individuals occurring in the database are all and the only existing individuals.

2. The *unique name assumption* [79]. Individuals with unique names are distinct.

3. The *closed world assumption* [78]. The only possible instances of the database are those implied by the database.

In this chapter, we first give an overview of first-order languages. We then provide a proof-theoretic characterization of the $\mathcal{D}$_model. Finally we define the semantics of null values within the proof-theoretic paradigm of the $\mathcal{D}$_model.

8.1 First-Order Languages

This section provides a brief overview of first-order languages as it relates to the work described in this dissertation. (A more detailed description of first-order languages may be found in [69].) A first-order language $\mathcal{L}$ is a pair $\mathcal{L} = (\mathcal{A}, \mathcal{W})$, where $\mathcal{A}$ is an alphabet of symbols and $\mathcal{W}$ is a set of well-formed formulas (wffs) that constitute the legal statements[19] in the language and are specified using the symbols of $\mathcal{A}$. The symbols of $\mathcal{A}$ includes a collection of:

- *variables*
- *constants*
- *predicates*
- *logical connectives*: $\supset$ (implies), $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\equiv$ (iff)
- *quantifiers*: $\forall$ (universal), $\exists$ (existential).

Using the symbols of $\mathcal{A}$, the set $\mathcal{W}$ of wffs can be constructed as follows:

- Every variable or constant in $\mathcal{L}$ is a *term.*
- If P is an n-ary predicate symbol of $\mathcal{A}$, and $t_1, t_2, \ldots, t_n$ are terms, then $\mathsf{P}(t_1, t_2, \ldots, t_n)$ is an *atomic formula.* $\mathsf{P}(t_1, t_2, \ldots, t_n)$ is a *ground formula* iff $t_1, t_2, \ldots, t_n$ are all constants.

[19] i.e., they abide by the rules of the language.

- Well-formed formulas are built inductively from the atomic formulas using parentheses, logical connectives and quantifiers as follows:

  1. An atomic formula is a well-formed formula (wff).
  2. If F is a wff and x is a variable, then $(\forall x)(\mathsf{F})$ and $(\exists x)(\mathsf{F})$ are wffs.
  3. If F and G are wffs, then so are $(\mathsf{F} \vee \mathsf{G}), (\mathsf{F} \wedge \mathsf{G}), (\mathsf{F} \supset \mathsf{G})$, $(\mathsf{F} \equiv \mathsf{G})$, and $\neg\mathsf{F}$.
  4. All wffs of $\mathcal{W}$ are defined by 1-3.

A *closed* wff is one that does not contain any *free* variable (i.e., it contains only quantified variables and constants).

8.1.1 Semantics

This section provides the semantics of first-order languages. Precise meaning is assigned to each of the symbols of the alphabet $\mathcal{A}$ of a first-order language $\mathcal{L} = (\mathcal{A}, \mathcal{W})$, and based on this assignment, truth values of wffs in $\mathcal{W}$ are defined. Standard definitions (see, for example, [69, 80]) are used in describing the semantics of first-order languages.

Definition 8.1[69, 80]: Let $\mathcal{L} = (\mathcal{A}, \mathcal{W})$ be a first-order language. An *interpretation* for $\mathcal{L}$ is a triple $I = <D,K,E>$ where

1. D is a non-empty set of objects, called the *domain* of I, over which the variables of $\mathcal{A}$ range.

2. K:$\{c|c$ is a constant of $\mathcal{A}\} \longrightarrow D$. Thus K is a mapping that assigns one element of D to each constant of $\mathcal{A}$ (i.e., for each constant c of $\mathcal{A}$, $K(c) \in D$).

3. E is a mapping from the predicates of $\mathcal{A}$ into sets of tuples of elements of D (i.e., for each n-ary predicate symbol P of $\mathcal{A}$, $E(P) \subseteq D^n$, where $D^n = \{(x_1, x_2, \ldots, x_n)|x_i \in D, 1 \leq i \leq n\}$). $E(P)$ is called the *extension* of P in the interpretation I.

8.2 A Proof-Theoretic Characterization of the $\mathcal{D}$_Model

This section presents a first-order definition of the $\mathcal{D}$_model. Prior to this, several notations and definitions are introduced first.

Let $\mathbf{F} = \{\mathtt{A}_1(\mathrm{T}_1), \mathtt{A}_2(\mathrm{T}_2), \ldots, \mathtt{A}_m(\mathrm{T}_m)\}$ be a frame template. A frame instance *fi over* $\mathbf{F}$ (or, we say a frame instance *fi* of the type $\mathbf{F}$) is a set of attribute-value pairs $\{< \mathtt{A}_1, V_1 >, < \mathtt{A}_2, V_2 >, \ldots, < \mathtt{A}_m, V_m >\}$, where $\mathtt{A}_j$ is an attribute of $\mathbf{F}$, and $V_j \subseteq dom(\mathrm{T}_j), 1 \leq j \leq m$. The notation $\mathbf{F}(x)$ is to represent that x is a variable that ranges over the frame instances of the frame template type $\mathbf{F}$. We also use the notation $\mathbf{F}(\{V_1, V_2, \ldots, V_m\})$ to denote the frame instance *fi* of type $\mathbf{F}$ if the attributes are clearly understood.

Let $\mathtt{A}$ be an attribute of type T. Let x be a variable that ranges over the frame instances of the frame template type $\mathbf{F}$. Then $\mathrm{T}(y)$ is to denote that y is a value variable of an attribute type T. $\mathtt{A}(x, y)$ is to denote that $< \mathtt{A}, y >$ is an attribute-value variable pair of x.

For convenience, we use the following abbreviations:

- The notation $(\forall x/\mathbf{F})(W)$ abbreviates $(\forall x)(\mathbf{F}(x) \supset W)$. This is read, "for all x that are of type $\mathbf{F}$, W is the case". The notation $(\exists x/\mathbf{F})(W)$ abbreviates $(\exists x)(\mathbf{F}(x) \wedge W)$. This is read, "there exists an x whose type is $\mathbf{F}$ such that W is the case".

- if $x = x_1, \ldots, x_n$ where x_i $(1 \leq i \leq n)$ is either a variable or a function symbol, then $W(x)$ abbreviates $W(x_1, \ldots, x_n)$, and $(\forall x)(W(x))$ abbreviates $(\forall x_1) \ldots (\forall x_n)(W(x_1, \ldots, x_n))$.

Definition 8.2: Let $\mathbf{F}$ and $\mathbf{F}'$ be two frame templates. Let x be a variable that ranges over all the frame instances of type $\mathbf{F}$. $\mathbf{F}$ *is-a* $\mathbf{F}'$ is specified by the wff

$$(\forall x)[\mathbf{F}(x) \supset \mathbf{F}'(x)].$$

Example 8.1: Figure 7 illustrates the is-a relationship between the **Publications** frame template and the **Proceedings**, **Book_Chapters**, **Technical_Reports** and **Journals** frame templates. The semantics of this hierarchy can be specified by the following first-order wffs:

$$(\forall x)[\mathbf{Journals}(x) \supset \mathbf{Publications}(x)]$$

$$(\forall x)[\mathbf{Proceedings}(x) \supset \mathbf{Publications}(x)]$$

$$(\forall x)[\mathbf{Book\_Chapters}(x) \supset \mathbf{Publications}(x)]$$

$$(\forall x)[\textbf{Technical\_Reports}(x) \supset \textbf{Publications}(x)]$$

The following are some of the wffs that specify the disjointness property of these frame templates:

$$(\forall x)\neg[\textbf{Journals}(x) \wedge \textbf{Proceedings}(x)]$$

$$(\forall x)\neg[\textbf{Journals}(x) \wedge \textbf{Book\_Chapters}(x)]$$

$$(\forall x)\neg[\textbf{Proceedings}(x) \wedge \textbf{Book\_Chapters}(x)]$$

The following wff asserts that all **Publications** have attribute `Authors`.

$$(\forall x/\textbf{Publications})(\exists y/\mathit{Names})\texttt{Authors}(x, y).$$

The same can be written as $(\forall x)[\textbf{Publications}(x) \supset [(\exists y)(\mathit{Names}(y) \wedge \texttt{Authors}(x, y))]$.

□

Definition 8.3 Let f be a folder and let **F** be a frame template. **F** is associated with f, denoted by *assoc-with*(**F**,f) iff

$$(\exists x)[\mathit{member}(x, \mathbf{f}) \wedge \mathbf{F}(x)],$$

where $\mathit{member}(x, \mathbf{f})$ is to denote that variable x is a member of f. That is, x is a variable which ranges over the frame instances of type **F** in the folder f.

Definition 8.4: Let f and g be two folders. f depends-on g, denoted by *depends-on*(f,g) iff

$$(\forall x)[member(x, \mathbf{f}) \supset member(x, \mathbf{g})].$$

8.2.1 A First-Order Language

This section defines a proper subset of first-order languages as a precept of formally defining[20] $\mathcal{D}$_Model databases within the framework of first-order logic.

Definition 8.5: Let $\mathcal{L} = (\mathcal{A}, \mathcal{W})$ be a first-order language. $\mathcal{L}$ is a *$\mathcal{D}$_model language* iff $\mathcal{A}$ has the following properties:

1. $\mathcal{A}$ includes function[21] symbols. Terms may now be defined recursively by

   (a) Every variable or constant of $\mathcal{A}$ is a term.

   (b) If f is a function symbol, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

2. There are only finitely many constants in $\mathcal{A}$, but at least one.

[20] In this dissertation, the first-order language is necessary *only* for the theoretical reformulation of the $\mathcal{D}$_model as a first-order theory. The language is not considered to be the fundamental query language for the $\mathcal{D}$_model.

[21] A simple use of function symbols is to incorporate composite attributes into frame templates. Ullman [97] elaborates on the use of function symbols in the same way. For example, a memorandum frame template could have the description **Memo**= {`Sender(FName,LName)`,`Receiver(FName, LName)`,`MemoDate(Month,Day,Year)`, `Subject`}. Here, **Memo** is a predicate symbol with the four terms `Sender`,`Receiver`,`MemoDate` and `Subject`. The first term, `Sender`, is a binary function symbol applied to arguments `FName` and `LName` which represent the first and last name of a sender. The variables `FName` and `LName` will, in each frame instance of the frame template type **Memo**, be replaced by two appropriate constants.

3. There is a finite set of predicates in $\mathcal{A}$. Along with each predicate is associated an integer $n \geq 0$, its *arity*, denoting the number of arguments it takes. (A constant will be treated as a predicate of arity zero [31].)

4. There is a binary predicate E that functions as the equality relation. Let E be the equality predicate and $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_n$ be sequences of terms. Then, $\mathsf{E}(x, y)$ abbreviates $\mathsf{E}(x_1, y_1) \wedge \ldots \wedge \mathsf{E}(x_n, y_n)$, and $\neg\mathsf{E}(x, y)$ abbreviates $\neg\mathsf{E}(x_1, y_1) \wedge \ldots \wedge \neg\mathsf{E}(x_n, y_n)$.

5. Among the predicates of $\mathcal{A}$ is a distinguished nonempty subset of unary predicates called *frame templates.*

Definition 8.6: Let $\mathcal{D} = (\mathcal{A}, \mathcal{W})$ be a $\mathcal{D}$_model language. A first-order theory $T \subseteq \mathcal{W}$ of $\mathcal{D}$ is a $\mathcal{D}$_model theory iff it satisfies the following properties[22]:

1. *Assertions.* Let f be a folder. Let $\mathcal{F}_{\mathsf{f}} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n | \mathit{assoc\text{-}with}(\mathbf{F}_i, \mathsf{f}), 1 \leq i \leq n\}$. For each $\mathbf{F} = \{\mathsf{A}_1(\mathsf{T}_1), \mathsf{A}_2(\mathsf{T}_2), \ldots, \mathsf{A}_k(\mathsf{T}_k)\} \in \mathcal{F}_{\mathsf{f}}$ and any frame instance $\{< \mathsf{A}_1, x_1 >, < \mathsf{A}_2, x_2 >, \ldots, < \mathsf{A}_k, x_k >\} \in \mathsf{f}$, an axiom $\mathbf{F}_{\mathsf{f}}(< x_1, x_2, \ldots, x_k >) \in T$, where the x_i's are value variables or function symbols.

2. *Particularization Axioms.* These axioms explicitly state the database assumptions that govern query evaluation:

[22]This extends the notations given in [30, 80] for folders and composite attributes in frame templates in our work.

(a) For some set $\Delta \subseteq \mathcal{W}$ of ground atomic formulas, in which no predicate is the equality predicate, $\Delta \subseteq T$.

For each m-ary predicate P of $\mathcal{A}$ distinct from the equality predicate, define a set C_p of n-tuples $(n \geq m \geq 0)$[23] of constants by

$$C_p = \{\vec{c} | P(\vec{c}) \in \Delta\}$$

where, $\vec{c}$ denotes an n-tuple of constants $< c_1, \ldots, c_n >$. The set C_p is called the *extension of* P *in* T. Suppose $C_p = \{< c_1^{(1)}, \ldots, c_n^{(1)} >, \ldots, < c_1^{(s)}, \ldots, c_n^{(s)} >\}$ is the set of all the n-tuples of constants in the extension of P. Then in addition to the wffs of Δ, T contains the following *completion axiom* for P:

$$(\forall x_1) \ldots (\forall x_n)[P(< x_1, \ldots, x_n >) \supset \mathsf{E}(x_1, c_1^{(1)}) \wedge \ldots \wedge \mathsf{E}(x_n, c_n^{(1)}) \vee \ldots \vee \mathsf{E}(x_1, c_1^{(s)}) \wedge \ldots \wedge \mathsf{E}(x_n, c_n^{(s)})]$$

If $C_p = \{\ \}$, then T's completion axiom is

$$(\forall x_1) \ldots (\forall x_n) \neg P(< x_1, \ldots, x_n >)$$

[23] $n > m$ if we have composite types.

Define a folder to be any subset of

$$\bigcup_{P_i \in \Delta} C_{p_i}$$

(Intuitively, the completion axiom states that the only values frame instances that the frame template P can have are

$$\{< c_1^{(1)}, \ldots, c_n^{(1)} >, \ldots, < c_1^{(s)}, \ldots, c_n^{(s)} >\}.)$$

(b) If $c_1, c_2, \ldots, c_n$ are all of the constants of $\mathcal{A}$ in the database,

i. T contains the *domain closure axiom*

$$(\forall x)[\mathsf{E}(x, c_1) \vee \mathsf{E}(x, c_2) \vee \ldots \mathsf{E}(x, c_n)]$$

ii. T contains the *unique name axioms*

$$\neg\mathsf{E}(c_1, c_2), \ldots, \neg\mathsf{E}(c_1, c_n), \ldots, \neg\mathsf{E}(c_{n-1}, c_n).$$

3. T contains each of the following *equality axioms*[24]

(a) Reflexivity

$(\forall x)\mathsf{E}(x, x)$

[24]These axioms are necessary since the particularization axioms involve the equality predicate. (They force equality to be interpreted in the standard way.)

(b) Symmetry

$$(\forall x)[\mathsf{E}(x,y) \supset \mathsf{E}(y,x)]$$

(c) Transitivity

$$(\forall x)(\forall y)(\forall z)[\mathsf{E}(x,y) \wedge \mathsf{E}(y,z) \supset \mathsf{E}(x,z)]$$

(d) Principle of substitution of equal terms:

$$\begin{aligned}(\forall x_1)\ldots(\forall x_n)(\forall y_1)\ldots(\forall y_n)\ [\ & P(\{<\mathtt{A}_1,x_1>,\ldots,<\mathtt{A}_n,x_n>\}) \wedge \\ & \mathsf{E}(x_1,y_1) \wedge \ldots \wedge \mathsf{E}(x_n,y_n) \supset \\ & P(\{<\mathtt{A}_1,y_1>,\ldots,<\mathtt{A}_n,y_n>\})]\end{aligned}$$

4. The only wffs of T are those sanctioned by conditions 1-3 above.

As defined above, T provides the proof-theoretic perspective of the database. The database is now viewed as a set of well-formed formulas that constitute the first-order theory T and queries are first-order formulas that have to be proven given the database as premises.

Definition 8.7 A *$\mathcal{D}$_model database* is a triple $(\mathcal{D}, T, IC)$ where:

1. $\mathcal{D}$ is a $\mathcal{D}$_model language.

2. T is a $\mathcal{D}$_model theory for $\mathcal{D}$.

3. IC is a set of wffs of $\mathcal{D}$, called *integrity constraints.* For each n-ary predicate P

distinct from E, *IC* must contain a wff of the form

$$(\forall x_1)\ldots(\forall x_n)\ [\ (\mathsf{P}(\{<\mathtt{A}_1, x_1>,\ldots<\mathtt{A}_n, x_n>\})\ \supset$$

$$\mathrm{T}_1(x_1)\wedge\ldots\wedge\mathrm{T}_n(x_n)\wedge\mathtt{A}_i(\mathrm{T}_i), 1\leq i\leq n] \qquad (1)$$

where the T_i's are attribute types. $\mathrm{T}_1,\ldots,\mathrm{T}_n$ are the *domains* of P.

8.3 Generalizing the Proof-Theoretic Characterization to Accommodate Null Values

In [21], Codd denotes any occurrence of the value-unknown type of null by the special symbol ω. However, the paper does not provide a clear formal semantic of the null value ω, and, Codd himself indicates that his treatment of ω "should be regarded as preliminary and in need of further research." One problem here is that the same symbol ω is used to denote individuals that may have entirely different semantic notions. For instance, consider the folder shown in Figure 19. Here, ω would be used to denote values for `Sender` and `PrevSalary` for the frame instances fi_1 and fi_2 respectively. In this section we show how the null values of the form "value at present unknown" may be defined within the proof-theoretic paradigm for the $\mathcal{D}$_model.

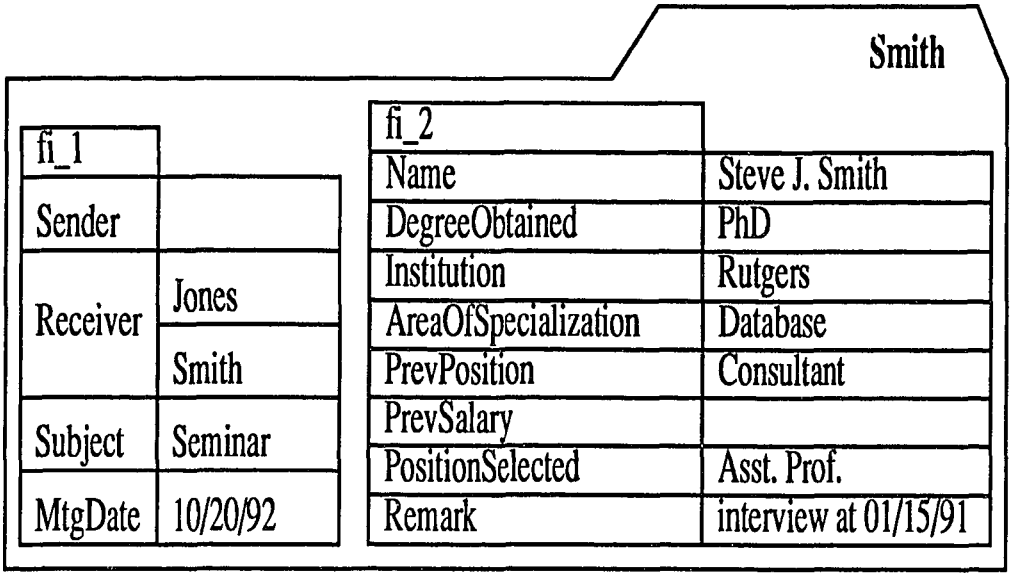


Figure 19: A folder containing two frame instances that have null values

Consider the frame instance fi_1 of the folder shown in Figure 19. Here, this frame instance represents the fact:

> "Some sender sent a memo to Jones and Smith regarding a seminar on 10/20/92 but I don't know who it is."

This fact may be represented by the first-order wff

$$(\exists x)\mathtt{Sender}(x)\mathbf{Memo}(x, \{Jones, Smith\}, seminar, 10/20/92) \qquad (2)$$

which says that there is an individual x that exists with the desired properties. By naming this individual ω, we can eliminate the existential quantifier in (8.1), and

instead assign the properties to ω directly:

$$\texttt{Sender}(\omega)\textbf{Memo}(\omega, \{Jones, Smith\}, seminar, 10/20/92) \quad (3)$$

Formula (8.2) is said to be in *Skolem normal form* since all existential quantifiers are eliminated by replacing variables they quantify with arbitrary functions of all universally quantified variables that precede them in the formula [15, 30, 80]. In logic terminology, ω is called a *Skolem constant* [64] (i.e., a skolem function of zero arguments).

The frame instance fi_2 represents the fact

> "Steve J. Smith, who graduated from Rutgers University with a PhD degree specializing in the area of Database, and is applying for an Assistant Professor position, will be interviewed on 01/15/91; However, his previous salary as a Consultant is unknown."

Here, we must designate a name for this unknown previous salary which must be distinct from the unknown sender of fi_1.[25] This fact may be represented by the first-order wff

$$\texttt{PrevSalary}(\omega') \quad \textbf{Application}(Steve\ J.\ Smith, PhD, Rutgers, Database, Consultant, \omega', Asst.\ Prof., interview\ at\ 01/15/91) \quad (4)$$

[25]The `Sender` value of fi_1 portrays entirely different semantics from the `PrevSalary` value of fi_2.

Hence, each time a new null appears into the theory, it must be designated a fresh name that is distinct from all the other names of the theory. We are thus dealing with *indexed* nulls [81] (or *marked* nulls [5, 51]). This technique enriches the information content of null values by associating subscripts with their various occurrences in order to distinguish between them.[26] These ideas are formalized in the following definition.

Definition 8.8: Let $\mathcal{D} = (\mathcal{A}, \mathcal{W})$ be a $\mathcal{D}$_model language, where the constants of $\mathcal{A}$ are partitioned into two disjoint sets of constants $C = \{c_1, c_2, \ldots, c_n\}$ and $\Omega = \{\omega_1, \omega_2, \ldots, \omega_r\}$. Here, Ω may be empty but C may not be empty. Each ω_i is called a *null value.* A first-order theory $T \subseteq \mathcal{W}$ is a *generalized $\mathcal{D}$_model theory of $\mathcal{D}$ with null values* iff it satisfies the properties of the $\mathcal{D}$_model theory of $\mathcal{D}$ stated in definition 3.6, with the following extensions of 2(a) and 2(b):

2. (a′) For some set $\Delta \subseteq \mathcal{W}$ of ground atomic formulas of $\mathcal{D}$, $\Delta \subseteq T$.

   For each m-ary predicate P of $\mathcal{A}$ distinct from the equality predicate define a set K_p of n-tuples ($n \geq m \geq 0$) of constants from $C \cup \Omega$ by

   $$K_p = \{\vec{k} | P(\vec{k}) \in \Delta\}$$

   where, $\vec{k}$ denotes an n-tuple of constants $< k_1, \ldots, k_n >$.

   Suppose $K_p = \{\{k_1^{(1)}, \ldots, k_n^{(1)}\}, \ldots, \{k_1^{(s)}, \ldots, k_n^{(s)}\}\}$ is the set of all the n-tuples of constants in the extension of P. Then in addition to the wffs of Δ, T contains the following *completion axiom* for P:

[26] Indexed nulls can be used to indicate that certain values are equal even though they are unknown.

$$(\forall x_1)\ldots(\forall x_n)[P(\{< \mathtt{A}_1, x_1 >, \ldots, < \mathtt{A}_n, x_n >\}) \supset$$
$$\mathsf{E}(x_1, k_1^{(1)}) \wedge \ldots \wedge \mathsf{E}(x_n, k_n^{(1)}) \vee \ldots \vee$$
$$\mathsf{E}(x_1, k_1^{(s)}) \wedge \ldots \wedge \mathsf{E}(x_n, k_n^{(s)})]$$

If $K_p = \{\ \}$, then T's completion axiom is

$$(\forall x_1)\ldots(\forall x_n)\neg P(\{< \mathtt{A}_1, x_1 >, \ldots, < \mathtt{A}_n, x_n >\})$$

2.(b′) T contains the domain closure axiom

$$(\forall x)[\mathsf{E}(x, c_1) \vee \mathsf{E}(x, c_2) \vee \ldots \mathsf{E}(x, c_n) \vee \mathsf{E}(x, \omega_1) \vee \mathsf{E}(x, \omega_2) \vee \ldots \mathsf{E}(x, \omega_r)].$$

T contains the unique name axioms

$$\neg\mathsf{E}(c_1, c_2), \ldots, \neg\mathsf{E}(c_1, c_n), \ldots, \neg\mathsf{E}(c_{n-1}, c_n)$$

T *may* contain one or more inequalities of the following forms:

$$\neg\mathsf{E}(\omega_i, c_j) \text{ for some } 1 \leq i \leq r,\ 1 \leq j \leq n.$$

$$\neg\mathsf{E}(\omega_i, \omega_j) \text{ for some } 1 \leq i, j \leq r,\ i < j.$$

Definition 8.9: A *generalized $\mathcal{D}$_model database with null values* is a triple $(\mathcal{D}, T, IC)$ where:

1. $\mathcal{D}$ is a $\mathcal{D}$_model language.

2. T is a $\mathcal{D}$_model theory for $\mathcal{D}$.

3. IC is a set of wffs of $\mathcal{D}$, called *integrity constraints.*

Theorem 8.1: Every *generalized $\mathcal{D}$_model theory* T is consistent[27] as a first-order theory with equality. (The proof is a consequence of [80].)

Proof: The proof is constructed by adding enough inequalities to T to yield a generalized $\mathcal{D}$_model theory. Add to T every inequality $\neg\mathsf{E}(c_i, \omega_j)$ such that neither this inequality nor the inequality $\neg\mathsf{E}(\omega_j, c_i)$ is already present in T. Similarly, add to T every inequality $\neg\mathsf{E}(\omega_i, \omega_j)$ for $i \neq j$ such that neither this inequality nor the inequality $\neg\mathsf{E}(\omega_j, \omega_i)$ is already present in T. The resulting theory is a generalized $\mathcal{D}$_model theory. □

It may also be observed that the *generalized $\mathcal{D}$_model theory* T is decidable since the domain closure axiom restricts the class of its models to those whose domains are no larger than the finite set of constants of the theory. (Objects in the perceived world are seen as a set of constants in a theory and not as a domain in an interpretation (in the model-theoretic perspective). Hence, the finiteness property does not bear any influence on the decidability of whether a given wff is a theorem.)

[27] A theory is inconsistent iff there is a wff W such that both W and ¬W are theorems of it; otherwise, it is consistent [72].

CHAPTER 9

CONCLUDING REMARKS

In this chapter we summarize what has been discussed in this dissertation and then we present an outlook for some on-going research that are based on the work described in this dissertation.

9.1 Summary

The intent of this dissertation is to present a data model and an algebra for an office system called TEXPROS for processing office documents. TEXPROS does not follow the ODA standards to specify document presentation information. Instead of distinguishing between the logical and layout structures of a document, the presentation in TEXPROS is simplified by combining both of these structures and incorporating them into a frame template. A frame template is instantiated by providing it with values to form a frame instance which becomes the *synopsis* of the document. The cost saved in manipulating frame instances can improve the overall performance dramatically.

The data model describes documents using two hierarchies: a document type hierarchy which depicts the structural organization of the documents, and a folder organization which represents the user's real-world document filing system. The document type hierarchy exploits structural commonalities between frame templates. The frame templates in the hierarchy are related by specialization and generalization [90]. Such a hierarchy helps classify various documents. The folder organization mimics

the user's real-world document filing system and provides the user with an intuitively clear view of the filing system. This facilitates document retrieval activities.

We have also presented an algebraic language for retrieving frame instances from a folder organization. The algebra developed by Guting et al. [41] also deals with documents. Following closely the ODA standard, Guting et al. describe documents in terms of schemas, instances and layouts. A schema is represented by ordered labeled trees, which describe the logical structure and data values contained in a class of documents. In contrast to Guting's algebra, we combine both logical and layout structures of a document and incorporate them into the frame template. Moreover, we store the synopsis of a document, rather than the original contents, in the frame instance. Since the information contained in a frame instance does not reflect any particular (logical or layout) structure, the order of the attributes is insignificant.

The algebras for the NF^2 [1, 4, 28, 54, 67, 74, 76, 83] data models handle relations with relation-valued attributes (similar to the composite and nested attributes in the frame templates). Due to the way we organize a frame instance, many important topics concerning the data models such as functional dependencies among attributes [66, 96] become unimportant here in our work.

The algebra presented in this dissertation enables the user to retrieve documents in the folder organization. It is used as a sound basis to express the semantics of a high level query language for TEXPROS. The question of the expressive power of the algebra relative to its operation on the objects of the data model has been

addressed. It has been shown that the algebra is more expressive than the relational algebra.

This dissertation has also focused attention on null values of the form *value at present unknown* for the $\mathcal{D}$_model. A proof-theoretic characterization of the $\mathcal{D}$_model has been provided and the semantics of null values within the proof-theoretic paradigm have been defined. Providing clear and well defined semantics of null values makes it less difficult to process queries in their presence.

9.2 Potential Research Directions

This section presents an overview of some potential research topics that emanate from the work described in this dissertation.

9.2.1 Query Optimization

Research is under way to describe and exemplify a way of exploiting the $\mathcal{D}$_algebra to define a high-level as well as user-friendly query language for the $\mathcal{D}$_model. Translating the algebra into the query language introduces the syntax and the semantics of the high-level query language by means of the algebra definition provided in this dissertation.

Given an expression denoting a query in this high-level query language, in general there are various ways of expressing the query and computing the answer. Each way of expressing the query suggests a strategy for finding the answer. It is not

expected for the user to write queries in a way that suggests the most efficient strategy. Hence, it becomes the responsibility of the system to transform the user's query into an equivalent query which can be computed more efficiently. Query optimization is an important issue to look into since the difference in execution time between a good strategy and a bad one may be huge.

Considerable research has been discussed in the area of query optimization for relational databases. A traditional approach has been to use low-level information such as statistical information about various costs to access individual relations. Systems that have been implemented, for example, System R [13, 84] and other experimental systems [25, 38], have demonstrated that significant gains can be obtained by using such low-level information. Considering statistical information about various costs to access the folders would be a good first step in attacking the query evaluation in TEXPROS.

9.2.2 Support for a Multi-User Environment

Currently, TEXPROS is a personal customizable tool. A central concern here is coordinated access to shared information. Certain considerations need to be made in order for TEXPROS to support a multi-user environment. There will be a need of providing a set of protocols that govern the consistent way of extracting the synopsis of information from given documents in such a fashion that the system will allow sharing retrieval of frame instances. A special user may be delegated to take central control of the system. This user would also regulate the different parts of the information

that the several users can access. This would be achieved by granting different types of authorization to the users. Alternatively, the users could use a message-based communication system such as electronic mail or, better yet, a computer conferencing system such as EIES [46, 47, 94] to define consistent ways of extracting synopses from documents.

In a multi-user environment, several programs may be executed concurrently. It would be necessary for the system to control the interaction among the concurrent programs in order to prevent them from destroying the consistency of the information stored. Concurrency control mechanisms [8] will have to be developed in order to achieve this control of interaction among concurrent users.

9.2.3 A TEXPROS Federated Architecture

A logical extension to the centralized multi-user environment would be a federated architecture for TEXPROS. The term *federated database system* was posited by Hammer and McLeod in 1979 [44]. Since then, a considerable amount of work has been reported in the context of federated database management systems (e.g., [26, 45, 61, 87]). In the TEXPROS federated architecture, the participating TEXPROSs would be loosely coupled[28] and would have a degree of local autonomy. They would be loosely coupled in the sense that the user would be responsible for creating and maintaining the federation without any control enforced by the federated system and its administrators.

[28]Other terms used for loosely coupled federated database systems are *interoperable database systems* [59, 61] and *multidatabase systems* [60, 61].

9.3 Ongoing Research Topics

A prototype of the dual model has been implemented into the TEXPROS document processing system. The hardware platforms include several SUN SPARCstations; the implementation software includes C and X-windows. The structured part of a frame instance is stored in INGRES tables while the unstructured part is stored in UNIX files.

9.3.1 Incorporating Hypertext into TEXPROS

Hypertext [22, 73, 89, 95] has the potential of providing good information management support for a wide variety of documentation efforts in office information systems. An emphasis of these systems is often on reading or browsing existing documentation.

The data model presented in this dissertation defines useful abstractions of modeling office documents. The characteristics of hypertext, such as the links and the navigation paradigm, can add more power and functionality to TEXPROS. Research is currently under way to investigate the incorporation of hypertext into the TEXPROS document model [100]. The work has started by developing two logic models: one for TEXPROS and one for hypertext. The logic model serves as the intermediary between TEXPROS and hypertext. Bridge laws [10] are being defined within the framework of the logic model to map TEXPROS into hypertext. Such laws should encompass all the fundamental aspects of hypertext and incorporate them into TEXPROS.

9.3.2 High Level Language Design

Currently, the algebraic language is being mapped to a high level data retrieval language. First, a calculus for the $\mathcal{D}$_model is being developed [107]. Equivalence will be shown between the calculus and the $\mathcal{D}$_algebra. The calculus will then be mapped to a two-dimensional high level language such as the one described by Zloof [108]. We hope to use the semantics of the resulting high level language to simplify and enhance query transformation and query optimization for the system.

9.3.3 Document Filing and File Reorganization

An agent-based architecture has been employed to automate document filing and to cope with file reorganization [101, 102]. Each folder is monitored by an agent; each agent is associated with a set of criteria and data structures for holding the frame instances. The criteria are used to govern the placement of a frame instance in appropriate folders. The agents are implemented as objects using an object-oriented approach. The approach encapsulates the internal representations of folders with the operations that manipulate them, thereby enhancing reusability of code and information hiding.

9.3.4 Information Retrieval

Finally, an information retrieval architecture has been developed that gives TEXPROS the capability of processing incomplete, imprecise and vague queries [63]. In-

complete and imprecise queries arise when the user cannot memorize the precise folder, template, or attribute names, and does not have time and patience (or even is reluctant) to find out the information by checking the document type hierarchy or logical folder organization. (The latter is particularly true when the type hierarchy and folder organization become too large to browse.) Vague queries arise when the user is not able to represent his request using the query language. (This may happen when the user does not have a clear idea of what he really wants.) A generalizer mechanism is employed to provide the user with informative messages when queries yield null answers. The retrieval environment provides a user-friendly interface which simplifies the learning and using the system. In contrast to conventional database systems, TEXPROS is capable of interpreting uncertain queries given by the user.

REFERENCES

[1] S. Abiteboul and N. Bidiot. Non First Normal Form Relations to Represent Hierarchically Organized Data. In *Proceedings of the third ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pp. 191–200, Waterloo, Ont., April 1984.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, Reading, Mass., 1983.

[3] A. V. Aho and J. D. Ullman. Universality of Data Retrieval Languages. In *Proceedings of the Sixth ACM Symposium on Principles of Programming Languages*, pp. 110–120, San Antonio, TX, January 1979.

[4] H. Arisawa, K. Moriya, and T. Miura. Operations and the Properties on Non-first-normal-form Relational Databases. In *Proceedings of the Ninth International Conference on Very Large Data Bases*, pp. 197–204, Florence, Italy, 1983.

[5] P. Atzeni and V. De Antonellis. *Relational Database Theory.* Benjamin/Cummings, Redwood City, CA, 1992.

[6] F. Bancilhorn. On the Completeness of Query Languages for Relational Databases. In *Mathematical Foundations of Computer Science, LNCS 64*, pp. 112–124. Berlin: Springer-Verlag, 1978.

[7] J. Banerjee, H. T. Chou, J. F. Garza, W. Kim, and D. Woelk. Data Model Issues for Object-Oriented Applications. In *Readings in Object-Oriented Database Systems*, (S. B. Zdonik and D. Maier, editors), pp. 197–208. Morgan Kaufmann Publishers Inc., 1990.

[8] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, Mass., 1987.

[9] E. Bertino, F. Rabitti, and S. Gibbs. Query Processing in a Multimedia Document System. *ACM Transactions on Information Systems*, vol. 6, no. 1, pp. 1–41, January 1988.

[10] M. Bieber and T. Isakowitz. Bridge Laws in Hypertext: A Logic Modeling Approach. Technical Report, Leonard N. Stern School of Business, July 1993.

[11] J. Biskup. A Foundation of Codd's Relational Maybe-Operations. *ACM Transactions on Database Systems*, vol. 8, no. 4, pp. 608–636, December 1983.

[12] J. Biskup. Extending the Relational Algebra for Relations with Maybe Tuples and Existential and Universal Null Values. *Fundamenta Informaticae*, vol. VII, no. 1, pp. 129–150, 1984.

[13] D. D. Chamberlin, M. M. Astrahan, M. W. Blasgen, J. N. Gray, W. F. King, B. G. Lindsay, R. Lorie, J. W. Mehl, T. G. Price, F. Putzolu, P. G. Selinger, M. Schkolnick, D. R. Slutz, I. L. Traiger, B. W. Wade, and R. A. Yost. A History and Evaluation of System R. *Communications of the ACM*, vol. 24, no. 10, pp. 632–646, October 1981.

[14] A. K. Chandra and D. Harel. Computable Queries for Relational Data Bases (Preliminary Report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, pp. 309–318, Atlanta, GA, April–May 1979.

[15] C. L. Chang and R. C. T. Lee. *Symbolic Logic and Mechanical Theory Proving.* Academic Press, New York, 1973.

[16] S. Christodoulakis. Office Filing, In *Office Automation*, (D. Tsichritzis, editor), pp. 67–89. Springer-Verlag, New York, 1985.

[17] S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa, and A. Pathria. Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System. *ACM Transactions on Information Systems*, vol. 4, no. 4, pp. 345–383, October 1986.

[18] H. Clifton, H. Garcia-Molina, and R. Hagmann. The Design of a Document Database. In *Proceedings of the ACM Conference on Document Processing Systems*, pp. 125–134, December 1988.

[19] E. F. Codd. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, June 1970.

[20] E. F. Codd. Relational Completeness of Data Base Sublanguages. In *Data Base Systems*, (R. Rustin, editor), pp. 65–98, Prentice-Hall, Englewood Cliffs, N. J., 1972.

[21] E. F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, vol. 4, no. 4, pp. 397–434, December 1979.

[22] E. J. Conklin. Hypertext: A Survey and Introduction. *IEEE Computer*, vol. 20, no. 9, pp. 17–41, September 1987.

[23] W. B. Croft and D. W. Stemple. Supporting Office Document Architectures with Constrained Types. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 504–509, San Francisco, CA, May 1987.

[24] C. J. Date. *An Introduction to Database Systems*, volume I. Addison-Wesley, Reading, Mass., 5th edition, 1990.

[25] R. Demolombe. Estimation of the Number of Tuples Satisfying a Query Expressed in Predicate Calculus Language. In *Proceedings of the Sixth International Conference on Very Large Data Bases*, pp. 55–63, Montreal, October 1980.

[26] R. Elmasri, J. Larson, and S. Navathe. Schema Integration Algorithms for Federated Databases and Logical Database Design. Technical Report CSC-86-9:8212, Honeywell Computer Systems Development Division, January 1986.

[27] P. C. Fischer and D. V. Gucht. Determining when a Structure is a Nested Relation. In *Proceedings of the Eleventh International Conference on Very Large Data Bases*, pp. 171–180, Stockholm, 1985.

[28] P. C. Fischer and S. J. Thomas. Operators for Non-First-Normal-Form Relations. In *Proceedings of the IEEE Computer Software and Applications Software Conference*, pp. 464–475, New York, October 1983.

[29] A. L. Furtado and L. Kerschberg. An Algebra of Quotient Relations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1–8, Toronto, Canada, 1977.

[30] H. Gallaire, J. Minker, and J.-M. Nicolas. Logic and Databases: A Deductive Approach. *ACM Computing Surveys*, vol. 16, no. 2, pp. 151–184, June 1984.

[31] A. V. Gelder, K. A. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, vol. 38, no. 3, pp. 620–650, July 1991.

[32] S. Gibbs and D. Tsichritzis. A Data Modeling Approach for Office Information Systems. *ACM Transactions on Information Systems*, vol. 1, no. 4, pp. 299–319, October 1983.

[33] S. Gibbs, D. Tsichritzis, E. Casais, O. Nierstrasz, and X. Pintado. CLASS Management for Software Communities. *Communications of the ACM*, vol. 33, no. 9, pp. 90–103, September 1990.

[34] S. J. Gibbs. Conceptual Modeling and Office Information Systems. In *Office Automation*, (D. Tsichritzis, editor), pp. 193–225, Springer-Verlag, Berlin, 1985.

[35] J. Grant. Null Values in a Relational Data Base. *Information Processing Letters*, vol. 6, no. 5, pp. 156–157, October 1977.

[36] J. Grant. Partial Values in a Tabular Database Model. *Information Processing Letters*, vol. 9, no. 2, pp. 97–99, August 1979.

[37] J. Grant. Incomplete Information in a Relational Database. *Fundamenta Informaticae*, vol. III, no. 3, pp. 363–378, 1980.

[38] J. Grant and J. Minker. Optimization in Deductive and Conventional Relational Database Systems. In *Advances in Data Base Theory*, (H. Gallaire, J. Minker, and J.-M. Nicolas, editors), vol. 1, pp. 195–234. Plenum, New York, 1981.

[39] J. Grant and J. Minker. Answering Queries in Indefinite Databases and the Null Value Problem. *Advances in Computing Research*, 1986.

[40] D. V. Gucht. On the Expressive Power of the Extended Relational Algebra for the Unnormalized Relational Model. In *Proceedings of the Sixth ACM SIGART-SIGMOD Symposium on Principles of Database Systems*, pp. 302–312, San Diego, CA, March 1987.

[41] R. H. Guting, R. Zicari, and D. M. Choy. An Algebra for Structured Office Documents. *ACM Transactions on Information Systems*, vol. 7, no. 4, pp. 123–157, April 1989.

[42] M. Gyssens and D. V. Gucht. The Powerset Algebra as a Result of Adding Programming Constructs to the Nested Relational Algebra. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 225–232, Chicago, Illinois, 1988.

[43] M. Hammer and D. McLeod. Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database Systems*, vol. 6, no. 3, pp. 351–386, 1981.

[44] M. Hammer and D. McLeod. On Database Management System Architecture. Technical Report MIT/LCS/TM-141, Massachusetts Institute of Technology, 1986.

[45] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Information Systems*, vol. 3, no. 3, pp. 253–278, July 1985.

[46] R. S. Hiltz and M. Turoff. The Evolution of User Behavior in a Computerized Conferencing System. *Communications of the ACM*, vol. 24, no. 11, pp. 739–751, November 1981.

[47] R. S. Hiltz and M. Turoff. Structuring Computer-Mediated Communication Systems to Avoid Information Overload. *Communications of the ACM*, vol. 28, no. 7, pp. 680–689, July 1985.

[48] P. Hoepner. Synchronizing the Presentation of Multimedia Objects – ODA Extensions –. *ACM SIGOIS Bulletin*, vol. 12, no. 1, pp. 19–32, July 1991.

[49] W. Horak. Office Document Architecture and Office Document Interchange Formats - A Current Status of International Standardization. *IEEE Computer*, vol. 18, no. 10, pp. 50–60, October 1985.

[50] R. Hull and J. Su. On the Expressive Power of Database Queries with Intermediate Types. In *Proceedings of the Seventh ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 39–51, Austin, TX, March 1988.

[51] T. Imielinski. Incomplete Information in Logical Databases. *Data Engineering (Special Issue on Imprecision in Databases)*, vol. 12, no. 2, pp. 29–40, June 1989.

[52] T. Imielinski and W. Lipski. On Representing Incomplete Information in a Relational Database. In *Proceedings of the Seventh International Conference on Very Large Data Bases*, pp. 388–397, Cannes, France, September 1981.

[53] T. Imielinski and W. Lipski. Incomplete Information in Relational Databases. *Journal of the ACM*, vol. 31, no. 4, pp. 761–791, October 1984.

[54] G. Jaeschke and H.-J. Schek. Remarks on the Algebra of Non First Normal Form Relations. In *Proceedings of the SIGACT/SIGMOD Symposium on Principles of Database Systems*, pp. 124–138, Los Angeles, March 1982.

[55] Y. Kambayashi, K. Tanaka, and K. Takeda. Synthesis of Unnormalized Relations Incorporating More Meaning. *Information Sciences*, vol. 29, pp. 201–247, 1983.

[56] T. Korson and J. D. McGregor. Understanding Object-Oriented: A Unified Paradigm. *Communications of the ACM*, vol. 33, no. 9, pp. 40–60, September 1990.

[57] W. Lamersdorf, G. Muller, and J. W. Schimdt. Language Support for Office Modelling. In *Proceedings of the International Conference on Very Large Data Bases*, 1984.

[58] W. Lipski. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transactions on Database Systems*, vol. 4, no. 3, pp. 262–296, 1979.

[59] W. Litwin and A. Abdellatif. An Overview of Multidatabase Manipulation Language MDSL. *IEEE Computer*, vol. 19, no. 12, pp. 10–18, December 1986.

[60] W. Litwin, J. Boudenant, C. Esculier, A. Ferrier, A. Glorieux, J. La-Chirmia, K. Kabbaj, C. Moulinoux, P. Rolin, and C. Stangret. SIRIUS Systems for Distributed Data Management. In *Distributed Data Bases*, (H.-J. Schneider, editor), North-Holland, The Netherlands, 1982.

[61] W. Litwin and L. Mark. Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys*, vol. 22, no. 3, pp. 267–293, September 1990.

[62] K. C. Liu and R. Sunderraman. Indefinite and Maybe Information in Relational Databases. *ACM Transactions on Database Systems*, vol. 15, no. 1, pp. 1–39, 1990.

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

[64] D. Loveland. *Automated Theory Proving.* Springer Verlag, Berlin, 1979.

[65] E. Lutz, H. V. Kleist-Retzow, and K. Hoernig. MAFIA - An Active Mail-Filter-Agent for an Intelligent Document Processing Support. In S. Gibbs and A. A. Verrijn-Stuart, editors, *Multi-User Interfaces and Applications*, (S. Gibbs and A. A. Verrijn-Stuart, editors,) Elsevier Science Publishers B. V., North-Holland, 1990.

[66] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Potomac, Md., 1983.

[67] A. Makinouchi. A Consideration of Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model. In *Proceedings of the International Conference on Very Large Data Bases*, pp. 447–453, Tokyo, Japan, October 1977.

[68] T. W. Malone, K. R. Grant, K. Y. Lai, R. Rao, and D. Rosenblitt. Semistructured Messages are Surprisingly Useful for Computer-Supported Coordination. *ACM Transactions on Information Systems*, vol. 5, no. 2, pp. 115–131, April 1987.

[69] E. Mendelson. *Introduction to Mathematical Logic.* Van Nostrand, Princeton, N.J., 1964.

[70] F. S. Mhlanga, J. T. L. Wang, T. H. Shiau, and P. A. Ng. A Query Algebra for Office Documents. In *Proceedings of the Second International Conference on Systems Integration*, pp. 458–467, Morristown, NJ, June 1992.

[71] J. Minker and D. Perlis. Applications of Protected Circumscription. In *Proceedings of the Conference of Automated Deduction*, pp. 414–425, Napa, CA, May 1984. Springer-Verlag, Berlin and New York.

[72] J.-M. Nicolas and H. Gallaire. Data Base: Theory vs. Interpretation. In *Logic and Databases*, (H. Gallaire and J. Minker, editors), pp. 33–54. Plenum, New York, 1978.

[73] J. Nielsen. The Art of Navigating Through Hypertext. *Communications of the ACM*, vol. 33, no. 3, pp. 296–310, March 1990.

[74] G. Ozsoyoglu, Z. M. Ozsoyoglu, and V. Matos. Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions. *ACM Transactions on Database Systems*, vol. 12, no. 4, pp. 566–592, December 1987.

[75] J. Paredaens. On the Expressive Power of the Relational Algebra. *Information Processing Letters*, vol. 7, no. 2, pp. 107–111, October 1978.

[76] P. Pistor and F. Anderson. Designing a Generalized NF^2 Model with an SQL-Type Interface. In *Proceedings of the International Conference on Very Large Data Bases*, pp. 278–288, Kyoto, August 1986.

[77] F. Rabitti. A Model for Multimedia Documents. In *Office Automation*, (D. Tsichritzis, editor), pp. 227–250, 1985.

[78] R. Reiter. On Closed World Data Bases. In *Logic and Data Bases*, (H. Gallaire and J. Minker, editors), pp. 55–76. Plenum Press, New York, 1978.

[79] R. Reiter. Equality and Domain Closure in First-Order Databases. *Journal of the ACM*, vol. 27, no. 2, pp. 234–249, April 1980.

[80] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, (M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors), pp. 191–233. Springer-Verlag, New York, 1984.

[81] R. Reiter. A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Database with Null Values. *Journal of the ACM*, vol. 33, no. 2, pp. 349–370, April 1986.

[82] L. A. Rowe and M. R. Stonebraker. The POSTGRES Data Model. In *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, pp. 83–96, Brighton, September 1987.

[83] S.-J. Schek and M. H. Scholl. The Relational Model with Relation-Valued Attributes. *Information Systems*, vol. 11, no. 2, pp. 137–147, 1986.

[84] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. Lorie, and T. G. Price. Access Path Selection in a Relational Database Management System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 23–34, Boston, May–June 1979.

[85] D. Shasha and J. T. L. 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.

[86] G. M. Shaw and S. B. Zdonik. A Query Algebra for Object-Oriented Databases. In *Proceedings of the Sixth International Conference on Data Engineering*, Los Angeles, CA, February 1990.

[87] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, vol. 22, no. 3, pp. 183–236, September 1990.

[88] J. R. Shoenfield. *Mathematical Logic.* Addison-Wesley, Reading, Mass., 1967.

[89] J. B. Smith and S. F. Weiss. Hypertext. *Communications of the ACM*, vol. 31, no. 7, pp. 816–819, July 1988.

[90] J. M. Smith and D. C. Smith. Database Abstractions: Aggregation and Generalization. *ACM Transactions on Database Systems*, vol. 2, no. 2, pp. 105–133, March 1977.

[91] S. J. Thomas and P. C. Fischer. Nested Relational Structures. In *Advances in Computing Research III*, (P. C. Kanellakis, editor), pp. 269–307. JAI Press, Inc., Greenwich, CT, 1986.

[92] D. C. Tsichritzis. Form Management. *Communications of the ACM*, vol. 25, no. 7, pp. 453–478, July 1982.

[93] D. C. Tsichritzis, S. Christodoulakis, A. Lee, and J. Vandenbroek. A Multimedia Office Filing System. In *Office Automation*, (D. Tsichritzis, editor), pp. 43–65. Springer-Verlag, Berlin, 1985.

[94] M. Turoff and R. S. Hiltz. Computer Support for Group versus Individual Decisions. *IEEE Transactions on Communication*, vol. 30, no. 1, pp. 82–90, January 1982.

[95] M. Turoff, U. Rao, and S. R. Hiltz. Collaborative Hypertext in Computer Mediated Communications. In *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on Systems Sciences - VOL. IV, IEEE Computer Society*, pp. 357–366, Hawaii, January 1991.

[96] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, Potomac, MD, 2nd edition, 1982.

[97] J. D. Ullman. *Principles of Database and Knowledge-Base Systems.* Volume II: The New Technologies, Computer Science Press, Potomac, Md., 1989.

[98] S. L. Vandenberg and D. J. DeWitt. Algebraic Support for Complex Objects with Arrays, Identity, and Inheritance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 158–167, San Diego, CA, June 1991.

[99] Y. Vassiliou. Null Values in Data Base Management: A Denotational Semantics Approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 162–169, 1979.

[100] J. Wan, M. Bieber, J. T. L. Wang, and P. A. Ng. On Incorporating Hypertext into TEXPROS. Research Report, Department of Computer and Information Science, New Jersey Institute of Technology, March 1993.

[101] J. T. L. Wang, F. S. Mhlanga, Q. H. Liu, W.-C. Shang, and P. A. Ng. Database Support for Software Documentation: The TEXPROS Project. To appear as a book chapter in *Software Automation and Productivity Improvement*, 1994.

[102] J. T. L. Wang, F. S. Mhlanga, Q. H. Liu, W.-C. Shang, and P. A. Ng. An Intelligent Documentation Support Environment. To appear in *Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, June 1993.

[103] J. T. L. Wang and P. A. Ng. TEXPROS: An Intelligent Document Processing System. *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, no. 2, pp. 171–196, June 1992.

[104] 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.

[105] 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*, pp. 311–325, Washington D.C., May 1986.

[106] C. Zaniolo. Database Relations with Null Values. *Journal of Computing and System Sciences*, vol. 28, no. 1, pp. 142–166, February 1984.

[107] Z. Zhu, J. T. L. Wang, and P. A. Ng. A Calculus for Office Documents: $\mathcal{D}$_Calculus. Research Report, Department of Computer and Information Science, New Jersey Institute of Technology, March 1993.

[108] M. M. Zloof. Query By Example: A Data Base Language. *IBM Systems Journal*, vol. 16, no. 4, pp. 324–343, 1977.