PAGE CUBE: A Model for Storage and Retrieval of Documents Relevent to a Document Production Workflow in an Office

Smriti K. Sinha
Dept. of Computer Science
Tezpur University
Tezpur- 784028, India
email: *smriti@agnigarh.tezu.ernet.in*
Gautam Barua
Dept. of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati-781001, India
email:*gb@iitg.ernet.in*

Abstract

A model for storage and retrieval of relevent documents during a document production workflow in an office is presented. The objective of the workflow is to review a document by multiple reviewers, where each reviewer contributes her own comments during review. Such a document is a case of the workflow and is called a multi-part multi-signature document, where each part is basically the comments of the reviewer. Before commenting, the case is examined by the reviewer with respect to a set of documents containing standing rules, precedents etc. Such a set of documents constitute the context or the rationale of the comment of the reviewer. The context is dynamic in nature. The objective of the model is to construct the dynamic context automatically during review. The model also enables to retrive the static context of a comment of a precedent, created in the past.
Key Words: *office automation, document production workflow, multi-part multi-signature document, active database, document database, context, multi-dimensional model, page cube.*

1 Introduction

It is not simple to describe all the work performed in an office in a common framework without referring to the specific organization. But document production and storage is a common work in almost all offices. The scope of our discussion is limited to this common work. Document production in an office is based on a *request-reaction-response* paradigm. When a document containing a request is received in an office, the office reacts to the request. The reactions are recorded in the form of comments on the document and finally a response document is despatched. We can term the process as Document Production Workflow (DPW). The resultant doocument of a DPW is termed as a multi-part multi-signature document (MPMSD). Therefore, a MPMSD is a *case* of the DPW. The first part of a MPMSD is the request document and the last part is the response document and the the parts in between are the comments of the reviewers. Each part of a MPMSD is signed by the corresponding reviewer, the first part is signed by the originator of the request.

1.1 A Scenerio

To understand the salient features of a DPW in an office let us consider the following scenario of a DPW in a University. An employee, A submits an application, m_A regarding her travel plans for approval to the head, B of the department, where she is working. B verifies the travel plans in the context of previous cases of employees from the department already in travel, type of leave to be granted for A during travel, pending cases handled by A, resolutions on travel taken in departmental advisory committee, standing rules, etc. and adds her comment, m_B and forwards it to the finance officer, C. C also examines the case by verifying the budget allocation status under the head of account for travel, TA/DA rules in such cases, circulars from UGC on travel expenditure, and adds her comment, m_C on the amount that may be granted and forwards it to the director, D. The director

also justifies the previous comments, checks whether there exists any standing office order on travel, Board of Management's decision on such travel and approves the travel plans and adds the note of approval, may be in the form of office order,m_D. A copy of the whole multi-part document or only the office order m_D may finally go back to the originator, A and the original multi-part document is stored in a folder. The flow of the document is recorded in logbooks. This is a case of the travel plan workflow.

1.2 Motivation

In an office, a new document is produced in the context of a set of existing documents constituted of rules, precedents and other support documents. Rules are framed almost on all topics to prevent possibility of arbitrary decisions and are generally well defined. Where rules are not clear, we look for similar cases handled earlier, that is, precedents. Here, rules include regulations, office orders, meeting proceedings etc. and the precedents are already produced cases following concered rules and regulations. Certain decisions require support documents (for example, a purchase indent to sanction a purchase). Rules, precedent and such suppot documents constitute a *reference space*. A user navigates through a subspace of the reference space before producing a new document. This subspace is the *context* of the workflow. The navigation through the context is called the *case examination*. Therefore, the basic objective of the present work is to study automatic retrieval of documents relevent to a DPW and thus form the context for the case examination. After production, the new document may be linked to different documents of the office. It may, in tern, be included in the context of some other documents. Classical office document architecture proposed by ISO [1] failed to capture the context and the linking of documents.

Organization of office documents is a very important aspect, so far efficient retrieval is concerned. In paper-based office, documents are organized in folders. Related documents are stored in the same folder either as a stack or as a queue. As a result, we get a *linear structure* of document storage. The major disadvantages of the such a linear filing structure are:

- When a document is referred in more than one folder, then multiple copies of the same document are to be stored in different folders.
- If the folders are created topicwise then from these linear structure it is very hard to find out documents of particular type or the vice versa.
- The problem gets compounded when, say, the documents of all type and on a particular topic, created in a particular year are required.

To overcome these problems, we see in manual offices yearwise different folders both on topics as well as on types are created. This results in redundant folders and unnecessary duplication of documents.

In automated office systems, attempts are made to resolve the issues of filing, using database concepts. The approaches to organize the documents in digital office systems try to mimic the manual linear file structure and filing procedures and thus provide digital equivalent of folders, cabinets etc., to group related documents and folders respectively. Commercially most successfull office automation groupware products, *Lotus Notes* and *CabinetNG* (*www.cabinetng.com*) also mimic traditional folder structure and both of them failed to provide automatic context generation. Now, the fundamental questions are: is it at all necessary to follow the traditional filing procedure and continue using linear structures like stack and queue? Can we liberate the user of paperless office of tomorrow from the hessels of filing the digital document in proper folder? Our model liberates the documents in digital office from skewed linear structure to an open space, called *page space*.

Security issues of DPW are discussed in detail in [2]. An overall architecture of DPW is discussed in detail in another paper [3]. In the present paper we shall discuss in detail the storage and retrieval of office documents so that the relevent documents are retrieved automatically during the examination of a case.

1.3 The Issues

The problem is basically grouping of office documents in different groups based on different perspectives and to retrieve the relevent documents automatically from the corpus of documents. In this section we enumerate the issues of storage and retrieval of documents in an office with respect to a DPW.

- **Dynamic Context**: The issue is how to construct the dynamic context of a DPW automatically, so that the user can peruse the required documents from the context, while creating a new document. As soon as a new document is created, it is to be incorporated in the concerned contexts automatically.

- **Citation Set**: During creation of a new document, the user can select a subset of documents from the dynamic context and draw citations at different points of the new document. Thus the document citing and the set of documents cited form a set called citation set. While perusing the document at a future point of time, the user should be able to reach all the cited documents. Similarly, from a given document, a user should be able to reach all those documents, where the document in question is cited.
- **Static Context**: The context of an already created document is static in nature. It is a snapshot of the dynamic context at the time of creation of the document. Given a document, we can find out the context of the document. It signifies the set of related documents made available to the user at the time of creation of the document. If a user play ignorant of existing of a relevent document, it can be addressed by establishing the availibility of the document from the context at the time of creation of the document.
- **Multi-Part Document**: Documents related by *part of* relationship form a group, cutting across class boundaries. For example, documents may have multiple versions. Each version is a different document. *version of* is a *part of* relation. A document may have multiple comments on it. The comments are different documents. *comment on* is a part of relation. When a document belonging to a muli-part group is included in a context of a document, all the other parts of the group are also automatically included in the context. The user may cite a part or the whole group in a document.

1.4 Related Work

Document modelling is a well researhed problem in Computer Science. There are different approaches for retrieval of relevent documents. In convensional information retrieval (IR) approach a document is treated as a long sequence of unstructured text. Searches are performed primarily by matching keywords, usually combined with boolean operators, against the text. Prominent work in this approach are found in [4, 5, 6]. Another approach is mapping document structures and meta-data into database schema, and import the documents into database objects of a host system. The capabilities of the host system can then be used to perform queries. The drawbacks of this approach are, conversion process is expensive and the imcompatibilities between document structures and database schema often result in loss of information [7]. In another approach documents are represented as graphs with nodes and links. Labels at the nodes or edges in the graph represent the data and the meta-data in the documents, and query processing involves traversal and transformation of the graphs. Prominet works in this approach are [8, 9]. In another approach, documents are considered as fully structured, stuctured in SGML, HTML or XML and the Document Type Definitions (DTD) are considerd as schema and the documents as instances of corresponding DTD and thus documents can be queried using formal query languages [9, 7]. Our approach is in between. The office documents may be structured, unstructured or semi-structuted, highly interlinked and have meta-data. The meta-data is modelled as dimensional attributes and interconnectvity of pages as graphs.

Office information systems are also well researched problem [10]. The advent of Computer Supported Cooperative Work (CSCW)[11, 12] in general and workflow technology in particular introduced a new paradigm of modelling office work as workflow [13, 14]. The document production in office is modelled as workflow and detail architecture and protocols are discussed in [3, 2]. The organizational memory is a focused area of CSCW and the importance of context in organizational work is discussed in [15]. The automatic construction of context for document production workflow is a new concept and is the subject matter of the present work. For automatic retrieval of documents we follow the style of Starburst active database rules [16].

2 The Framework

In this section we discuss a generic framework of document production in an office, where office work is represented as a workflow of one or more tasks and a document is represented as a multi-part document with one or more parts. The framework comprises of two components: office document and a document production workflow.

2.1 Office Documents

Office documents may be structured, unstructured or semi-structured texts. Apart from the text, an office document has different attributes and the values of these attributes form the *meta-data* of a document. Therefore, an office document has three aspects: *profile, content* and a *presentation.*

2.1.1 Profile

A profile is the *bio-data* of the document. It contains meta-data of the document which provides a detail description of the document. The profile comprises of a set of *keywords* and three types of records: *production record, storage record* and *flow record.*

- *keywords*: This is a limited collection of representative terms from the vocabulary of the office concerned, which represent the content of the document. Most existing text retrieval techniques rely on indexing keywords or indexing terms. There are standard models for keyword based retrieval, like vector space model, detail is nicely discussed in [6]. We excluded keywords from the discussion of our model but it can be easily incorpotated to. Unfortunately, keywords alone cannot adequately capture the document contents, resulting in poor retrieval performance. We need other attributes, like record attributes to complement the keyword description of an office document. The record attributes can be categorized as follows:

- *production record*: This record consists of production related attributes like, *class, type, topic, date of production* etc., of an office document. A document may belong to one of the classes like *rule, case, support document* etc. A document may be created using some templates, called types or forms. For example, *office order, notice, casual leave application* etc. are different types of documents. Moreover, a document may be on one or more topics.

- *storage record*: This record consists of storage related attributes like address, size, authorization etc. of a document. An office document may be in one of the three states: *active, reference* and *archive.* Accordingly, there are three storage types. A document in an office migrates from one storage storage type to another as soon as the document changes state. Therefore, a migratory office document has a pair of addresses: an invarient logical address and a variable physical address.

- *flow record*: It is a record pertaining to the flow of a document from one point to the other. The attributes may be senderId, receiverId, time of sending, time of receiving the document.

2.1.2 Content

The content of a document may be multimedia information. But, for the present work, we assume that it contains only text. The simplest type of digital document is plain text, which contains only the natural language text of the document with much restricted formatting and structural information. The advent of word processing and text formatting systems introduced "tagged" or "marked up" documents to substitute for plain text documents. Generally speaking a markup tag is simply some extra information embedded in the document using either a text editor or a word-processor.

2.1.3 Presentation

The content of a document is presented for display or for printing in a layout framework. The layout framework associates the contents with a hierarchy of layout objects such as pages, columns etc. The layout structure is also hierarchal in nature. It also includes presentation rules, like a chapter should be in a new page, the content should be justified both left and right etc. HTML is now a defacto layout framework for web apges. Thus the layout framework provides the *getup* of a document. For storage and retrieval of office documents, the focus of our discussion is on the contents and on the profiles of the documents. Therefore, presentation aspect of a document is excluded from the rest of our discussion.

2.2 Document Production

Here we discuss the framework for document production. As discussed earlier, document producction in an office is based on request-reaction-response paradigm and the process can be described as a workflow.

Definition 1 *A* **workflow** *W can be represented as a partially ordered set of tasks $\{w_1, w_2, w_3, ..., w_n\}$, that involve manipulation of certain objects by certain subjects [17].*

Definition 2 *A* **subject** *is an active entity of the system, which accesses objects. These accesses must be controlled to ensure they match the security requirements.*

Definition 3 *An* **object** *is an passive entity of the system, which contain information to be protected from unauthorized accesses.*

Definition 4 *A* **task** w_i *is defined as* $\{OP_i, T_{IN_i}, T_{OUT_i}\}$*, where* OP_i *is the set of operations to be performed in* w_i*,* T_{IN_i} *is the set of object types allowed as inputs,* T_{OUT_i} *is the set of object types expected as outputs.*

Definition 5 *A* **case** *is an instance of a workflow. A case is defined as the ordered set of task instances of a workflow and the order is the same as the order of execution of the tasks in the workflow. A task instance* w_inst_i *is defined as* $(OPER_i, IN_i, OUT_i)$ *where* $OPER_i$ *is the set of operations performed during the execution of* w_i*,* IN_i *is the set of input objects to* w_i *and* OUT_i *is the set of output objects from* w_i*.*

In a Document Production Workflow(DPW), all the tasks have the same function, namely to *review* documents, and this is done by the office workers, who are the subjects and the documents are the objects. The set of operations for the tasks consists of receive a document, examine the case, add a comment, sign and send the document to the next reviewer. The main input object to tasks is a MPMSD. To review this MPMSD the reviewer refers to a *context*. The objects in the context are the other input objects to tasks. The output object of tasks consists of the MPMSD after the addition of the reviewer's comment and the evidence of receipt of the document for review.

2.2.1 Multi-Part Multi-Signature Document

A MPMSD is an instance, a case of a DPW. As mentioned earlier, in a MPMSD, the first part is the request document, the last part is the response document the other parts in between are the reactions of the reviewers. A reviewer can read the previous parts but cannot modify the content of any of the previous parts, even cannot reorder or drop any of the previous parts.

Definition 6 *A* **multi-part multi-signature document (MPMSD)** *is an* n*-tuple,* d_n*, of parts,* $n \geq 1$*, such that* $d_n = (p_1, p_2, p_3, \cdots, p_n)$*. Each part* p_i *in turn is defined as a 3-tuple* (m_i, c_i, σ_i)*, where* m_i *is the comment of subject* s_i*, ,* c_i *is the context based on which the comment is made and* σ_i *is the signature of* s_i*.* c_i *is formed from a default context,* c_i^d *which is the same for all i, and is provided by the DPW designer.*

2.2.2 Context

In this work, our aim is to automatically generate the context, whose contents are relevent to the tasks of a DPW. However, perfect systems are not available. Such system should retrieve each relevent document and only relevent documents. The conventional parameters for measuring the quality of the result set obtained after retrieval operation are *precision* and *recall*. The parameters are defined as follows:

precision = no. of relevent document retrieved / total no. of documents retrieved

recall = no. of relevent document retrieved / total no. of relevent documents in the document space

In a perfect system, the recall and the precision should both be equal to one, meaning that system should return all relevent documents without introducing any irrelevent document in the result set. Unfortunately, this is impossible to achieve in practice. If we try to improve recall, precision suffers; likewise, we can only improve precision at the expense of recall. The main reason is that obtaining precise description of a document, as well as obtaining precise description of the user need is not so easy [6]. Thus the central issue is a definition of a representation of document allowing an efficient retrieving mechanism. In our scheme, representation of an office document is done by the profile of the document. Thus most of the queries related to attributes of the office documents can be replied by processing the profiles whereas only navigational queries can be answered by traversing the links of interpage graphs.

The MPMSDs are produced as cases of DPWs. The productions are done using an inline trusted third party (TTP), called an arbiter. A TTP is inline if it the intermediary, serving as the real-time means of communication between two entites A and B. It is shown in [2] that without an inline TTP, it is not possible to design a protocol to produce MPMSDs which can address all the security issues of MPMSDs. The arbiter in the scheme is heavy-weight. The arbiter also serves as storage manager, time server and certifying authority within an office domain. Detail of the production protocol in available in [2] and the architecture of DPW in detail is available in [3].

3 The Page Cube Model

Here we discuss a model for storage and retrieval of documents in an office during document production workflow with context as the main binding element. The office documents are considered here as *pages*. We term the model as *Page Cube (PC)*. A PC is a collection of registered pages of an office. Here pages are the main entities. *Registration* of a page means adding and recording a new page to the page cube and assign a unique pageId to the new page. PC has two components: *page space* and *page graphs*.

3.1 The Page Space

The page space is an n-dimensional space defined by n orthogonal dimensions. Each dimension represents a *theme* and is defined by a set of attributes. An attribute of a dimension may have sub attributes and a sub attribute may sub sub attributes. Thus, attributes of a dimension form a dimensional hierarchy. Therefore, we can say that the page space is defined by n orthogonal hierarchies. A page is represented in this space as a point, whose coordinate is an n-tuple. The dimensions of the page space are basically the attributes of the profiles of the pages. The main dimensions includes the following but not limited to:

- **Time**: It is a hierarchical dimension of constant height. Moreover, the attrubutes have fixed length. The hierarchy is
  year. month.day.hour.min.sec.
  Time dimension provides the time of creation of a page or better to say the time of submission of a page to the central authority called the arbiter.

- **Topic**: Topic is also an important dimension. A page may be in one or more topics. A topic may have subtopics and a subtopic may have subsubtopic and so on. Thus it forms a topic hierarchy. Topic is a growing hierarchy.

- **Type**: A page may be of a type. *office order, circular, notice, comment* etc. are the examples of types. Types are basically templates or forms using which pages are generated. We assume in our model that a page is created uning some predefined form. Type dimension signifies the forms. A type may have subtypes and thus forms a type hierarchy. This hierarchy not fixed but growing. In time, a new type may be created under an existing leaf type.

- **Category**: A page may belong to one of the three categories: *context, document* and *part*. The pages of category part are the elementary pages. A page of category document contains a set of links to the pages of category part. The links are ordered on the time of creation. The pages of category documents are the MPMSDs discussed in the framework. A page of category context contains a set of links to pages of category document. A DPW will have a context page asssociated with it.

- **Class**: Pages may be classified based on the themes of the content of the pages. The classes are *rule, case* and *support*. A page may contain rules on some topics, may be a part of a case of a workflow or may be a support page. A page may belong to more than one class. For example, content of a part of a case may be rules on some topics. For example, resolutions of the Board of Management (BoM) is the last part of a case of BoM meeting workflow, at the same time, it is a rule by default to concerned topics. Since page cube is a closed model, as discussed later, where a resultset of a query will also be a page containing links to the pages satisfying the query and thus serves as a hub, therefore *resultset* is also may be a class. Moreover, the queries are also stored in the page cube as pages, therefore *query* may be another class.

- **User**: In DPW, user is an important dimension. A user belonging to an office may be the reviewer of some cases and therefore the author of some pages. For some cases, like the pages of category document or context, the author is the arbiter itself.

- **Domain**:The users of an office may belong to different domains of the office. Therefore, a page may be originated from a domain of an office. An office may have many domains. A domain may have subdomains. Thus domain forms a hierarchy. The tasks of a workflow may be distributed across different domains.

- **DPW**: In our model a page is produced in a DPW. This dimension gives the concerned workflow of a page.

- **State**: The pages in a DPW will in one of the three states: *active, reference* and *archive*. Pages of category context, documents of class case(under review), documents of class rules, are active pages; whereas, pages of category part, documents of class case, which are closed (precedent) are in reference state. The old pages which are neither active nor are refence are in archive state. The characteristic which differenciates these states is the accessibility. Archive pages have no access previleges for the users. The reference pages are read only and active pages may have previleges like read, write, modify etc.

This set of dimensions, common for all DPWs, is only a representative one. An office can identify more useful dimensions specific to the office concerned.

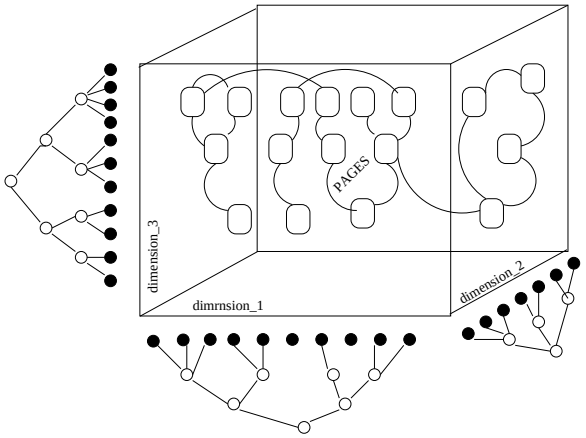

Figure 1: Page Cube

3.2 The Page Graph Component

The pages of a PC are linked to a given page either implicitly or explicitly. Implicitly liked pages are those pages which satisfy a *predicate* defined over the dimensional values the pages. The predicate is expresed as a query. Query languages for PC are discussed in section 6. Explicitly linked pages are those pages which are linked by explicit hyperlinks. Thus, the pages, which are explicitly linked, form a directed graph, where the pages are the vertices and the links are the edges. In addition to the implicit links provided by the attributes of the dimensional hierarchy, the pages belonging to a dimension may be explicitly linked forming a *dimensional graph(DG)* of the concerned dimension. Thus the page graph component of PC is a set of DGs.

Let V is a set of pages belonging to a dimansion d, E is a set of directed edges among these pages. Then, $G_d(V, E)$ is a digraph representing a dimension graph of dimension d. The vertices $\in V$ may be of different types. Similarly edges $\in E$ may also be of different types. A link in PC is a bidirectional link, comprises of a forward link and a reverse link. Accordingly, a bidirectional link is represented in $G_d(V, E)$ as a pair of edges, one forward and one backward. Therefore, V and E can be defined as :
$V = \{V_\alpha | V_\alpha$ is a set of vertices of type $\alpha\}$ and
$E = \{E^+_\beta, E^-_\beta | E^+_\beta$ is a set of forward edges of type β and E^-_β is a set of reverse edges of type $\beta.\}$.
Moreover, the edges $\in E$ may be weighted, making $G_d(V, E)$ a *weighted graph*. The weight may be defined differently in different dimensional graphs. Linking of different types of pages by different types of links in a dimension graph is subject to the satisfaction of a set of constraints C_d.

4 The Category Graph

Among the different common dimensions discussed in section 3.1, the dimension *category* is a very important dimension. For a DPW, the construction of dimensional graph for category is a mandatory one. Therefore, we shall discuss the category graph in detail in this section. A page may belong to one of the three categories: *part, document* and *context*, as discussed earlier. A part may cite other parts during production of the part. Similarly a part may be cited in many parts. Therefore, pages of category part are linked by *cite* type of links. A document may plug many parts and a part may be plugged in many documents. Therefore, pages of category document and part are linked by the links of type *docPart*. A context may plug many documents and

a document may be plugged in many contexts. Therefore, pages of category context and document are linked by the links of type *conDoc*. Ofcourse, the model has no restriction on document and document or context and context linking by corresponding types of links but we leave it to design and implemention of office specific systems. For brevity, we will restrain from discussing about that here.
Let P is a page cube,
$T_V = \{part, document, context\}$ is a set of types of vertices,
$T_E = \{cite, docPart, conDoc\}$ is a set of types of edges,
$V = \{V_{part}, V_{document}, V_{context}\}$, where V_{part} is a set of vertices of type *part*, $V_{document}$ is a set of vertices of type *document* and $V_{context}$ is a set of vertices of type *context*.
$E = \{E_{cite}, E_{docPart}, E_{conDoc}\}$,
where E_{cite} is a set of edges of type *cite*,
$E_{docPart}$ is a set of edges of type *docPart* and
E_{conDoc} is a set of edges of type *conDoc*.
Since the links are bidirectional, a subset E_T of type T, $T \in T_E$, in turn consists of two sets: a set of forward links of type T, E_T^+ and a set of reverse links of type T, E_T^-.
The category graph is defined as $G_{category}(V, E)$. The edges $\in E$ are weighted. A page may be plugged to another page at some time and may be unplugged at some other time. The weight of an edge is a tuple (t_+, t_-), where t_+ is the time of plugging and t_- is the time of unplugging. The significance of the temporal weight is that a link remains active till it is unplugged. A link remains active during the period defined by t_+ and t_-. Therefore, links in $G_{category}$ are persistent in nature. Deactivation or unplugging of a link does not delete the link, only changes its weight. Therefore, the temporal weights keep the history of linking pages. This temporal weight is used in reconstruction of the context of a part and will be discussed later. Since there is a possibility of plugging as well as unplugging more than once between the same pair of vertices, parallel edges with different weghts may exist.

The graph $G_{category}(V, E)$ is constructed by linking the pages by the edges of type $\in T_E$. The linking is subject to the satisfaction of the following constraints:

- **Citation Constraints**: A page p_i cites another page p_j iff the following conformability conditions on operands are satisfied:
  a. $p_i, p_j \in V_{part}$.
  b. $p_i \notin P$ and $p_j \in P$, means at the time citing, of citing page is a new page and the cited page is from the page cube. Only a page belonging to P can be cited in a new page.
- **DocumentPart Constraints**: A page p_j is plugged to page p_i iff the following conformability conditions are satisfied:
  a. $p_i \in V_{document}$ and $p_j \in V_{part}$
  b. p_i is in active state.
  c. $p_i, p_j \in P$ means, at the time of plugging the operands should be in the page cube. The significance of this restriction is that generation of pages of categories document and context is an internal process of the page cube and confined to registered pages only. For example, when a new page of category document is to be created, first a null document is created then other pages are plugged.
- **ContextDocument Constraints**: A page p_j is plugged to page p_i iff the following conformability conditions are satisfied:
  a. $p_i \in V_{context}$ and $p_j \in V_{document}$
  b. p_i is in active state.
  c. $p_i, p_j \in P$

4.1 Closure

Page Cube is a closed model. Simply put, this means that page is to be used as the input as well as the output format for queries. Closure is prominant in relational database model, as tables are both inputs to queries and outpus from queries. In addition using the QBE query language, queries are formulated also using tables. Similarly, in page cube, queries are also pages. The advantages of closed model are: [7]

- closure enables the possibility of using *views* in the same manner as the data itself, thus allowing views to be used as inputs to further query processing.
- closure allows the input and output to be handled in the same way; no separate mechanisms need to be devised for presentation and storage of query results, as they can inherit such properties from the input data inself.
- In addition, having queries in the same format as the input and output data enables the possibility of storing, presenting and managing queries in tha same way as the data. Queries on such stored queries can be used for future performance tuning.

A page may migrate from one storage to the other as soon as it changes its state and accordingly the physical address of the page may change from time to time. But at a particular time a page will have one and only one physical address. Therefore, each page in the page cube will have a pair of addresses: a time varing physical address and an invariant logical address. Let *pageId* is the unique invarient logical address of a page. A page of a particular type may be on more than a topic. In that case, a page may have multiple n-tuple coordinates. That is the reason, a distinct pageId is chosen as the logical address of a document. Otherwise the n-tuple coordinate would have been an ideal logical address of a page.

5 Production of Pages

From the above discussion it is clear that there are different categories of pages and the rules for production of different pages are different. In this section we will discuss how different pages are produced, what are the production rules etc.

5.1 Production of Parts

Pages of category part are the basic pages produced by the reviewers of a DPW. Production rules for pages of category part are:

P1. when a user selects a dpw, if the user is an authorized reviewer of the dpw then the context page of the dpw and the list of case pages of the workflow under review, are displayed.

P2. when a user selects a case for review, if the user is the current reviewer of the case then the case document is displayed and options for *add part, close case* are asked.

P3. when a user opts for add part option, if the case is not closed then type hierarchy is displayed; user selects a type (form); form is displayed; user either selects one or more topics from the topic hierarchy or creates new topic node in the topic hierarchy; selects additional classes in addition to default class = "case" of the page, selects the next reviewer; compose the page.

P4. when a user selects for create new case if the user is authorized to create a new case then type tree is displayed, user selects a type (form), form is displayed and the rest is same as rule P3.

P5. when composing a page, if the user wants to cite a page then the user selects the page navigating through the path *context.document.part*, the address of the cited page will be automatically incorporated in citation point of the citing page. Only parts reachable from the context can be cited.

P6. when the user submits the form on completion of composing, signature of the user on the page and the profile of the page are generated automatically defining other dimensional attributes from default values, send the profile and the page to the arbiter.

P7. when a profile and a page is received by the arbiter, time of receiept is immediately generated, verifies the profile, verifies the citations. If verification succeeds then register the page with a pageId, store the page in a location in the storage device and get the url of the location, record profile parameters in tables production, storage and flow, add corresponding edges to the citation graph, acknowledge the registration to the user by returning pageId else reject page.

During production of pages of category part, the focal point is the verification. Type, topic can be verified using dimesion tables. Citation path can be verified using $G_{category}(V, E)$. Now the question is how to verify whether a user is authorized and current reviewer. For that we can defined a matrix *UserDpw* over the dimensions *user* and *Dpw* as follows.
UserDpw Matrix: In this matrix rows represent the users and the columns represent the dpws. The $(i, j)^{th}$ cell represents the association of i^{th} user with j^{th} dpw and is defined by the following values:

- -1 if i^{th} role is not authorized to as reviewer of j^{th} dpw.
- k, $0 \leq k$ if i^{th} role is authorized as the k^th reviewer of j^th dpw. 0^{th} reviewer is somtimes called the*originator* of a new case.

In this scheme if (i, j)th cell of the UserDpw matrix contains a positive value then the i^{th} user is an authorized reviewer of j^{th} dpw and value signifies the position of the reviewer in the dpw. Now, if we define the size of a page of category document p_d, as the no of pages of class part, plugged to it, then a user u_i is the current reviewer of p_d, a case of dpw w_j, iff $size(p_d) = UserDpw(i, j)$, similarly u_i is the valid next reviewer iff $size(p_d) + 1 = UserDpw(i, j)$

5.2 Production of Documents

Production of pages of category documents are as follows:

D1. when a new part is registered, if the user has selected it as a new case then a new page of category = "document" and class = "case" of the dpw is created and is registered, the new part is plugged to it and thus produces a new document.

D2. when a new part is registered, if the user has selected a case for review then the new part is plugged to it.

D3. when a new part is registered, if the user has selected either a case for review or a new case and in addition classified the page as rule then for each topic of the part if there exists a page of category = "document" and class = "rule" on the topic then the new part is plugged to it else register a new page of catagory = "document" class = "rule" and then plug the new part is to it.

D4. when a new part is registered, if the user has selected either a case for review or a new case and in addition classified the page as support then for each topic of the part if there exists a page of category = "document" and class = "support" on the topic then the new part is plugged to it else register a new page of catagory ="document", class = "support" and then plug the new part to it.

5.3 Production of Dynamic Contexts

We recall, a context is a collection pages of category = "document" relevent to the dpw. A dpw has only one context page. Relevence of pages to a dpw is defined by the designer of a dpw using a set of requirement templates. A reviewer other than the designer of the dpw cannot delete or modify the given default context but can expand the context by adding new templates. A requirement template is defined a set of attribute value pairs. The attributes are dimensional attributes of the pagecube. For example,

- T_1 = {category="document", class="rule", topic="s1", time=t1-t2} signifies that documents containing rules on $s1$ produced during the tenure $t1-t2$ are relevent to the dpw concerned.
- T_2 = {category="document", class="case", time=t1-*, dpw=w} signifies that case documents of workflow w produced after $t1$ are relevent to the workflow concerned.
- T_3 = {category="document", class="support", type=f, topic= s2,} signifies that support documents of type f on topic $s2$ are relevent to the workflow concerned.

The production rules for contexts are:

C1. when a new requirement template is submitted to include in the template list, if the user is authorized to include a template then generate a list of pages from the cube satisfying the template and plug the pages to the context if not already plugged.

C2. when a new page of category = "document" is registered and if the page satisfy any one of the requirement templates of the dpw, then the page is plugged to the concerned context page.

C3. when a template is submitted for exclusion from the requirement template list, if the user is authorized for exclusion of any template, generate a list of documents satisfying the template, unplug the documents from the context iff the document does not satify any other template belonging to the list.

C4. when a page of category ="document" is deregistered and if the page is plugged to one or more context pages then unplug the document from the concerned pages.

5.4 Production of Static Context

Production of the static context of a given page of category part is process of recreation of history. Basic objective is to regenerate the context prevailing at the time of production of a given page. The static context of a given page is the set of documents plugged to the context before the time of production of the given page and either unplugged or even if unplugged, that happened after the time of production of the page. From the page cube we can regenerate the static context of a given page using the following algorithm.

Algorithm to regenerate the static context of a given page

Let $p \in P$ is a given page of category part belonging to the page cube P and t_p is the time of production of p. We are to regenerate the context C_{t_p} of p.

BEGIN

1 Find the dpwId w of the part p from P

2 Find the pageId p_c of the page of category = "context" and dpw = "w" from P

3 Find the set of forward edges from the $G_{category}(V, E)$, $e_c^+ = \{e = (v_i, v_j, (t_+, t_-))|e \in E_{conDoc}^+$ and $v_i \in V_{context}$ is the initial vertex , $v_j \in V_{document}$ is the terminal vertext of the edge and $v_i = p_c$ and $t_+ < t_p$ and ($t_- = 0 \vee t_- > t_p$)}.

4 Produce the context C_{t_p} using e_c^+.

5 On selection of a document with pageId $p_d \in C_{t_p}$

do

5.1 Find the set of edges from the $G_{category}(V, E)$,
$e_d^+ = \{e = (v_i, v_j, (t_+, t_-))|e \in E_{docPart}$ and $v_i \in V_{document}$ is the initial vertex , $v_j \in V_{part}$ is the terminal vertext of the edge and $v_i = p_d$ and $t_+ < t_p$ and ($t_- = 0 \vee t_- > t_p$})

5.2 Produce a document D_{t_p} using e_d^+.

enddo

END

Here, the pages C_{t_p} and D_{t_p} are not the persistent pages but the states of the concerned pages at time t_p.

6 Query Languages

A query is a an expression denoting a set of pages described by a formula ϕ of the form $\{p|\phi(p)\}$. A query language provide a user with a means for expressing questions in the form that can be handled by the model enabling the model to answer to the questions asked in a reasonable time. In this section we describe two equivalent query languages: The first language is Page Algebra (PA), a procedural language which uses specialized operators on the sets of pages to specify queries and a Page Structured Query Language (PSQL), a user-friendly pseudo-natural language with a simple means for expressing queries using a natural language form. The cental concepts are *pages, path expressions, queries* and *active rules* using path expressions.

6.1 Pages

Pages are the main entities to deal with. Therefore, before we discuss the query languages, we should have a clear understanding of a page and its constituent elements in the context of the page space and page graph components of the page cube. To define a page, we first define a few sets as follows: Let S is a countably infinite set of strings, $A \subset S$ is a set of variables called attributes, $V \subset S$ is a set of allowed values for the attributes $\in A$, L is a set of links and is defined as $L = \{(p_i, p_j, w, t, dir) | p_i$ is the linking page, p_j is the linked page, w is the weight, t is the type of the link, dir is the direction of the link: either forward(+) or the backward(-)$\}$. Linking page p_i is always the page containing the link. Therefore, it is implied and need not be explitly mensioned.

Definition 7 *A profile B is defined as a 2-tuple, an attribute and a list of valules*
$B = \{(a, \{v_1, v_2, \cdots, v_n\}) | a \in A, v_i \in V, i = 1, 2, 3, \cdots, n\}$

Definition 8 *A page p is defined as a quadruple $p = (B, X, L^+, L^-)$, B is the profile, $X \subset S$ is a set of strings defining the content of the page, $L^+ \subset L$ is the set of forward links, $L^- \subset L$ is the set of reverse links.*

6.2 Path expression

The notion of path expressions(PE) came from two different areas: (i) graph query languages and (ii) object-oriented query languages. For graph query lanugae, a path expression defines a path from one node in the graph to another in terms of intermediate node and edge labels. For object-orientd query language, a path expression defines a path from one object to another using membership and inheritance relationships. PEs in graphs are used in navigation oriented queries. Where as PEs in object-oriented databases are used to express long chain of inheritance or membership. In our model, navigation in the dimensional graphs is a common feature for the queries. Moreover, the dimensions of the PC are also hierarchical. Therefore, values of the attributes can be expressed as PEs. Use of path expressions in document databases are available in []. We follow the simplified PE discussed in [7]. The standard wildcat character "*" is used in a path expression to signify all all successor of a node in the dimension hierarchy as well as in the dimensional graph. The standard "." operator used commonly to denote relation attributes can now be cascaded to express listed path. In addition a ".." operator is introduced, which is used to construct an abbreviated path from a listed path. A PE is one of the following form: [7]

- *Basic Path*: Every vertex $v \in G$, where G is a graph may be a dimension hierarchy or a dimension graph of PC is a basic path. For example, p_1 where p_1 is a page on a topic "leave" in the topic hierarchy , is basic path.
- *Listed Path*: A listed path is a fully expanded PE, of the form $v_1.v_2.v_3.....v_n$, where $n \geq 1$, v_i is a basic path.
  For example, topic="leave.casual.special" signifies a full listed hierarchical value of the dimension topic. Again time="2000.09.28.*" signifies all pages created on 28th September, 2000.
- *Abbreviated Path*: An abbreviated path is of the form $..v_1.v_2..v_n$, $n \geq 1$ and v_i is a basic path. For example, a user may not be sure of exact path, therefore an abbreviated path "$p_1..p_n$, where p_1, p_n are pages, say p_1 is a context and p_n is a part. Here actual vaues of path variables are not important. Formally, the expression evaluates to
  $\exists p PATH(p) \wedge /p_1.p.p_n,$ where p is apath variable.
  The expression indicates that there exists a path between p_1 and p_n but the actual path inself is not of significance.

6.3 Page Algebra (PA)

PA is an operator based query language for querying pages from a page cube. It is an extension of relational algebra. PA is defined in terms of special set operators that map one or more sets of pages to a new set of pages. The result set of an operation in PA is a page conforming to our original notion of closed model. Every PA expression E represents a set of pages. The main operators are as follows:

- **Selection (σ)**: The selection operation $\sigma_\gamma E$ extracts a subset of pages from an input set E that satisfies the selection condition γ
  $\sigma_\gamma E = \{p | p \in E \wedge \gamma\}$

- **Plug ($\times$)**: Given two PA expressions E_1 and E_2, the expression $E_1 \times E_2$ reproduces the pages belonging to E_1 with forward links to the pages belonging to E_2 and the pages belonging to E_2 with backward links to pages belonging to E_1.
  Let $E_1 = \{p_1, p_2\}$ and $E_2 = \{p_3, p_4\}$. $E_1 \times E_2$ produces the pages with the following forward (+) and backward (-) links.
  $p_1 = (p_1, \{p_3^+, p_4^+\})$ $\quad$ $p_3 = (p_3, \{p_1^-, p_2^-\})$
  $p_2 = (p_2, \{p_3^+, p_4^+\})$ $\quad$ $p_4 = (p_4, \{p_1^-, p_2^-\})$
  The pages p_1 and p_2 each contains forward links to pages p_3 and p_4 and similarly the pages p_3 and p_4 each contains backward links to pages p_1 and p_2 and all the pages are reproduced. Again the links may be different types, say context to document(cd), document to part(dp), part to part(pp). Accordingly the forward and backward links may be qualified. If E_1 contains pages of category document and E_2 contains parts then the qualified links due to $E_1 \times E_2$ will be
  $p_1 = (p_1, \{p_3^{dp+}, p_4^{dp+}\})$ $\quad$ $p_3 = (p_3, \{p_1^{dp-}, p_2^{dp-}\})$
  $p_2 = (p_2, \{p_3^{dp+}, p_4^{dp+}\})$ $\quad$ $p_3 = (p_3, \{p_1^{dp-}, p_2^{dp-}\})$

- **Unplug ($\div$)**: This is the reverse operation of plug. Given two PA expressions E_1 and E_2, the expression $E_1 \div E_2$ reproduces the pages belonging to E_1 removing forward links to the pages belonging to E_2 and also reproduces the pages belonging to E_2 removing the backward links to pages belonging to E_1.

- **Path Selection ($\circ$)**: Given a PA expression E and a PE P, $E \circ P$ returns the set of pages rooted at $last(P)$ obtained by traversing the path P from each ofthe pages in E [7].

- **Union ($\cup$)**: Union is the normal set union operation. Given two PA expression E_1 and E_2. The result of $E_1 \cup E_2$ is a set of pages defined as
  $E_1 \cup E_2 = \{p | p \in E_1 \vee p \in E_2\}$

- **Intersection ($\cap$)**: Intersection is the normal set intersection operation. Given two PA expression E_1 and E_2. The result of $E_1 \cap E_2$ is a set of pages defined as
  $E_1 \cap E_2 = \{p | p \in E_1 \wedge p \in E_2\}$

- **Difference (-)**: is the normal set difference operation. Given two PA expression E_1 and E_2. The result of $E_1 - E_2$ is a set of pages defined as
  $E_1 - E_2 = \{p | p \in E_1 \wedge p \notin E_2\}$

For example, let the query be, find from the page cube P, all the documents containing rules on special casual leave and include them in the context of the dpw "w". In PA it can be expressed as

$\sigma_{category="context" \wedge dpw="w"} P$
$\times$
$\sigma_{category="document" \wedge class="rules" \wedge topic="leave.casual.special"} P$

6.4 Page Structured Query Language

Here we present a language, called PSQL, for interectively processing queries on pages . PSQL is an extended version of SQL. The primary motivation behind such a language is to provide users of datadase systems with a simple means for expressing queries using a natural language form. Standard SQL deals with flat tables and the result set of an SQL query is also a table. In SQL the main retrieval operation is the SELECT operation. To accomodate this feature in PSQL the SELECT clause will have the mechanism to allow the creation of composite page from the constituent pages. The resultant page of a query is basically a multi-part page which contains links to the constituent pages.

For example let us take the following query

```
SELECT  *
FROM P
```

```
WHERE   category = "document"
and     class = "rules"
and     topic IN {"4.3.*', "8..5"}
and     time BETWEEN {"1990.08.*", now}
GROUPBY  topic, time
```

The query produces a resultant page from a page cube P containg links to all the pages containing rules on any of the topics 4.3 or a sub topic of it, or on a topic 5, such that there exists a path from topic 8 to topic 5, produced between August 1990 and *now*. The value of now is defined by the arbiter with the current time stamp. The vaules included in the IN list are generally values of dimensional attributes. It is a shorter form of expressing ORs of attribute-value based predicates. The range of values in BETWEEN clause is a more general expression of IN list.

The expressive power of PSQL and P are equivalent. The expressions in PA can be expressed here in terms of clauses. For Example, the query discussed in PA can be expressed as follows:

```
( SELECT *
  FROM P
  WHERE
   category = "context"
   and dpw = "w" )
   PLUG (SELECT *
     FROM P
     WHERE
     category = "document"
     and class = "rule"
     and topic = "leave.casual.special" )
```

The complete grammer of PSQL is under development.

6.5 Active Rules

For automatic execution of queries and hence for generation of pages some event based features, like triggers, are to be incorporated in PSQL. Without any loss of generality we can incorporate active rule construct of Starbust query language of active databases. We can implement the production rules of pages discussed in the model as rules of active database, which provide a facility, tightly coupled with the database system software, for creating and executing production rules. These rules follow *Event-Condition-Action (ECA)* paradigm; they autonomously react to events occuring on the data, by evaluating a data-dependent condition, and by executing a reaction whenever the condition evaluation yields a truth value. Rules for PC canbe designed in the line of standard of Starburst system. Starburst is the active rule system developed at the IBM Almaden Research Centre[5, 16]. A rule may contain PSQL queries in evaluating conditions or in taking actions. Rules are defined using the CREATE RULE command. The syntext of this command in BNF botation is:

```
<Starburst-rule> ::= CREATE RULE <rule-name>
                     ON <cube-name>
                     WHEN <triggering-operations>
                     [ IF <PSQL-predicate> ]
                     THEN <PSQL-statements>
                     [ PRECEDES <rule-names> ]
                     [ FOLLOWS <rule-names>  ]

<triggering-operation> ::= SUBMITTED | REGISTERED
                           | PLUGGED | UNPLUGGED
```

The WHEN clause specifies what causes the rule to be triggered, IF clause specifies a condition to be evaluated once the rule is triggered, THEN clause specifies a list of actions to be expressed when the rule is triggered and the predicate is true. The PRECEDES and FOLLOWS clauses are used to specify priority ordering between rules. Detail of the rules are available in [5, 16].

6.6 Queries and Rules as Pages

In relational model, there is a query language called Query By Example (QBE), which uses relational skeletons to specify queries. In this language, in adition to to the inputs and outputs, the query language itself is "closed" under the notion of relational representations. Similarly in PC the queries and rules can be stored as pages using the same encoding scheme as for normal pages. For example if the pages are encoded with SGML or XML the queries and rules can be encoded with SGML or XML as well [7, 18, 8]. This enables queries to be queried to extract information which may be useful for performance tuning. This property is commonly known as "reflection" and gives higher expressive power to the language [7].

7 Relational Structure

The page cube model can be mapped to both relational database as well as to object database or to obejct-relational database. In this section we will provide the relational representation of the model.

7.1 The Schema

The profile component can be represented by the star schema or the snowflake schema of multi-dimensional datacube model of data warehousing. It is a common practice in multi-dimensional modelling that when the dimensions are simple in nature, such dimensions are incorpotarted into the cental fact table as an attribute itself when there is no justification of maintaing a separate dimension table. The advantage is that expensive join operations between the centtral fact table and those dimension tables can be avoided thereby the efficiency increases . Ofcourse, there is no clear rule for such justification. It is a design decision.

In the schema of the page cube some of the dimensions are incorporated in the cetral fact tables. Time, Catagory, Class, Status are such dimensions which can be incorpotated to the central tables without any loss of gererality. Time is a significant dimension and needs a special treatment for it. Once time is incopotated as attribute of the central tables then there is no need for separate timeId. Let us term the group of attributes (year, month, day, hour, min, sec) together as *timeStamp*. Since in our scheme central arbiter will provide the timeStamp, a unique timeStamp can be assigned by the arbiter to a page. Therefore, it can be used as a unique identifier for a page. In that case, we donot need to have a separate pageId. TimeStamp will be synonimous to pageId. Moreover, it will have other advantages as discussed later. According to this modification, the arbiter ensures that at a particular time one and only one page will be created. In multi-dimensioanl modelling instead of dealing with normalized relations, denormalized relations are preferred. For example, star schema of a multidimensional cube is denomalized one, where as snow-flake schema of the same cube is normalized one. The adjacency matrices dimensional graphs are basically sparse matrices. Therefore, only nonzero entries are represented as tuples in the corresponding relations. The schema is as follows:

Dimension Tables:

Type (*typeId, typeDesc*, ...)
Topic (*topicId, topicDesc*,.....)
User (*userId, userName, address*, ...)
Domain (*domainId, domainDesc*, ...)
Dpw (*dpwId, dpwDesc*, ...)

Central Fact Tables:

userDpw(*userId, dpwId, value*)
Production (*pageId, topicId, typeId, category, class, userId, domainId, dpwId*)
Storage(*pageId, location, size, signature, status, authorization*)
Flow(*pageId, senderId, sentTime, receivedTime, ReceiverId*)
Citation(*citingPageId, citedPageId, citeCount*)
DocumentPart(*documentId, partId, plugTime, unplugTime*)
ContextDocument(*contextId, documentId, plugTime, unplugTime*)
Notes:
The names of the relations and of the attributes are chosen in such a way that they are self explanatory. For brevity, we restrained to provide a data dictionary for the schema. A page may be on more than one topic,

may also belong to more than one class. Therefore, normalized Production relation will have more than one tuple with same pageId. Therefore, pageId alone cannot be the primary key of the relation. Without any loss of generility, if we consider multiple values of topicId as well as of class as a single atom, where values are delimited somehow, then the given relation serves the purpose. Choice from these two altnernatives is a design decision.

8 Conclusion

In the present work we proposed a model, called Page Cube, for storage and retrieval of documents during a document production workflow. The model provides the mechanism to construct the dynamic context of a DPW automaticaly. It also allows to reconstruct the context of the context of a comment on a precedent. The model can be mapped either to the relational model or to the object-oriented model. Correspondingly, PSQL can be mapped to SQL or to OQL. But detail formalisms of the mapping remains to be future work. The complete grammer for PSQL is under development.

References

[1] K. K. Bajaj, *Office Automation.* Macmillan India Ltd., 1 ed., 1989.

[2] S. K. Sinha and G. Barua, "Secure Flow of Persistent Multi-Part Documents in an Office," in *Proc. of the Int. Conf. on Information Technology at the Dawn of New Millennium, Bangkok, Thiland*, vol. 3, pp. 157–172, 2000.

[3] S. K. Sinha and G. Barua, "An Architecture for Document Production Workflow in an Office," in *Proc. of the Int. Conf. on Information Technology (CIT99), India*, pp. 45–52, 1999.

[4] A. K. Jain, et.al, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 31(3), pp. 265–320, 1999.

[5] C. Zaniolo et.al., ed., *Advanced Database Systems.* Morgan Kaufmann, 1 ed., 1997.

[6] D. Lee et. al., "Document Ranking and the Vector-Space Model," *IEEE Software*, vol. March/April, 1997.

[7] A. Sengupta, *DocBase - A Database Environment for Structured Documents.* Ph. D. Thesis, Indiana University, 1997.

[8] W. Schuetzelhofer et. al, "Graphical Navigation in XML-Databases," in *Proc. of the Int. Conf. on Information Technology at the Dawn of New Millennium, Bangkok, Thiland*, pp. 47–59, 2000.

[9] S. Abiteboul et.al., "The Lorel Query Languages for Semistructured Data," *http://www-db.stanford.edu/ ~ lore/.*

[10] C. A. Ellis and G. J. Nutta, "Office information systems and computer science," in *Computer-Supported Cooperative Work: A Book of Readings*, Morgan Kaufmann, 1988.

[11] I. Greif, ed., *Computer-Supported Cooperative Work: A Book of Readings.* Morgan Kaufmann, 1 ed., 1988.

[12] R. Baeker, ed., *Readings in Groupware and Computer-Supported Cooperative Work.* Morgan Kaufmann, 1 ed., 1993.

[13] D. Malling, W. Croft, "From office automation to intelligent worklow systems," *IEEE EXPERT*, vol. June, pp. 41–47, 1995.

[14] W. Printz and S. Kolvenbach, "Support for Workflows in a Ministerial Environment," in *ACM Proc. Computer-Supported Cooperative Work*, pp. 199–208, ACM, 1996.

[15] J. Conklin, "Capturing Organizational Memory," in *Readings in Groupware and Computer-Supported Cooperative Work*, Morgan Kaufmann, 1993.

[16] J. Widom, "The Starburst Active Database Rule System," *IEEE Tran. on Knowledge and Data Engineering*, 1996.

[17] V. Atluri and W. K. Huang, "An Authorization Model for Workflows," in *Computer Security, ESORICS96, LNCS 1148*, pp. 44–64, Springer Verlog, 1996.

[18] A. Sengupta, "Towards the Union of Databases and Document Management: The Design of Doc-Base," in *COMAD98, Hyderabad, India*, 1998.