~~TH 28~~ 43006

Secure Production and Storage of Digital Documents in an Office Environment

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

TH-28 43006

by

Smriti Kumar Sinha

Under the Supervision of

Professor Gautam Barua

Department Of Computer Science and Engineering

Indian Institute of Technology, Guwahati

to the

Department Of Computer Science

School of Science and Technology

TEZPUR UNIVERSITY

2001

Certificate

Certified that the work contained in the thesis, entitled *Secure Production and Storage of Digital Documents in an Office Environment*, carried out by *Smriti Kumar Sinha*, has been done under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

(Gautam Barua)
Professor,
Department of Computer Science and Engineering,
Indian Institute of Technology, Guwahati,
India.

October, 2001

Acknowledgments

It is a pleasure to thank my thesis supervisor Prof. Gautam Barua for his constant help, encouragement and support throughout the course of this work. Prof. Barua first introduced me to this interesting problem and motivated me to work on it, for which I am grateful to him. Many other people have helped me in this work from time to time. I would specially like to thank Prof. K. M. Pathak, former Vice-Chancellor of Tezpur University and Prof. D. N. Buragohain, Director of IIT, Guwahati for specially permitting me to carry out the research work. I am grateful to them too. I also thank Prof. M. Dutta and Prof. D. K. Saikia, who constantly encouraged me during the work and awaited eagerly, along with me, for the completion of the work. I thank my friends and colleagues in Tezpur University for their help and encouragement. I remember with gratitude, the help rendered by staff of Computer Centre and Library of IIT, Guwahati. I would like to thank my family members for bearing with my whims and for shouldering my social responsibilities during the tenure, to make me free. Finally, even-though do not know how to, but I would like to express my thanks and gratitude to my personal computer, my closest companion during the tenure of this work, for rendering trouble-free service.

Smriti K. Sinha

Abstract

An office is termed as an information processing centre of an organization. The objectives of an office are to store and provide timely, accurate and relevant information for effective decision making. In a conventional office, information is captured in paper documents. But paper-based offices are failing to realize these objectives. With the advent of information technology, a paradigm shift has occurred in office automation. As a result, a transformation from a paper-based office to a paper-less office or an e-office is being observed. In an e-office, digital documents capture organizational information. Document production, storage and retrieval is a common work in almost all offices. The scope of the thesis is limited to this common work within an office. Inter-office document flow is beyond the scope of this 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 dispatched. We can term the process as Document Production Work-flow(DPW). The resultant document 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 other parts in between are the comments of other reviewers, that means, the reactions. Each part of a MPMSD is signed by the corresponding reviewer. The first reviewer is also termed as the originator. MPMSD is a generic framework. Parts belonging to different cases may also form a MPMSD. Moreover, multiple versions of a document may also be defined as a MPMSD. A reviewer creates a part of a MPMSD in the context of a set of existing documents constituted of rules, precedents and other support documents, which in turn may be MPMSDs. Rules, precedents and support documents constitute a *reference space* of an office.

A reviewer navigates through a subspace of the reference space before producing a new document and draws citations wherever necessary to substantiate the rules position, the precedent position etc. of the new document. This subspace is called the *context* of the DPW. The process of navigation through the context is called the *case examination*. The context of a DPW changes with time. As soon a new document, relevant to a DPW, is created or a new part is added to a document of the context, it is to be incorporated automatically. During case examination, when a part of a precedent is perused, the context of creation of the part, is to be recreated. Security, production and storage are the three aspects of the DPW problem, which are studied. Different issues of DPW are identified and solutions for addressing those issues are proposed.

The main objective of storage of information in an office is to keep track of the *history* of *who* did *what, when, why* and *how*. Thus storage in an office serves as the organizational memory, where the documents are the neurons. Therefore, the central issue is to store the documents in such a way that they can be identified, located and retrieved in an efficient way. Moreover, from a document thus retrieved, all the related documents, including the context, should be reachable in a simple and straight forward way. An office docu-

ment has three aspects: *profile, content* and a *presentation.* The profile is the *bio-data* of the office document. It contains meta-data of the document which provides a detailed description of the document. The profile comprises of a set of *keywords* and three types of records: *production record, storage record* and *flow record.* Most existing text retrieval techniques rely on indexing keywords or indexing terms. There are standard models for keyword based retrieval, like vector space model. We excluded keywords from the discussion of our model but it can be easily incorporated. Unfortunately, keywords alone cannot adequately capture the contents of office documents. We need other attributes, like record attributes to complement the keyword description of an office document. The production record consists of production related attributes, like *class, type, topic, date of production* etc., of an office document. The storage record consists of storage related attributes, like *address, size, authorization, state* etc. of a document. The flow record pertains to the flow of a document from one point to the other. The content of a document may be multimedia information. But, for the present work, we assume that it contains only text. The content may be unstructured, structured or semi-structured. The content of a document is presented for display or for printing in a layout framework. The layout framework provides the *get up* of a document. For storage and retrieval of office documents, the focus of our discussion is on the profiles of the documents. Therefore, content encoding and presentation aspects of a document is excluded from the rest of our discussion.

A model for storage and retrieval of documents in an e-office during document production workflow with the context as the main binding element is proposed. The office documents are considered here as *pages.* The model is termed as *Page Cube (PC).* A PC is a collection of registered pages of an

office. Here pages are the main entities. A page has a *profile*, which describes the page and is defined by a set of attribute-value pairs. *Registration* of a page means adding and recording a new page to the page cube and assigning a unique pageId to the new page. PC has two components: *page space* and *page graphs*. The page space is an n-dimensional space defined by n orthogonal dimensions. Each dimension represents a *theme* and is defined by an attribute. An attribute may have attributes and further 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, for example are, *type, topic, time, class, category, state* etc. The pages of a PC are linked to a given page either implicitly or explicitly. Implicitly linked pages are those pages which satisfy a *predicate* defined over the dimensional values of the pages. Explicitly linked pages are those pages which are linked by explicit hyper links. Thus, the pages, which are explicitly linked, form a directed graph, where the pages are the vertices and the hyper 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. The edges of DGs may be weighted. The construction of the DGs of dimension category is a mandatory one for a DPW and is discussed in detail. The algorithms for production of the DPW Context of a DPW and the Case Context of a precedent case are also proposed.

Next, the mechanism of querying a page cube is discussed. Two equivalent query languages are proposed. The first language is called Page Algebra (PA), an algebraic language which uses specialized operators on the sets of pages to specify queries. The second one is called Page Structured Query Language (PSQL), a user-friendly pseudo-natural language with a simple means for expressing queries using a natural language form. The languages are similar to Relational Algebra and SQL respectively.

A framework for secure production and storage of digital documents is presented. DPW is a group work. Therefore, a distributed protocol without any central authority, like the SMTP of standard e-mail service, may prop up as a natural choice. In such a distributed approach, the digital document flows from mailbox to mailbox and the reviewers add their comments to the document. This mimics the flow of paper documents in manual offices. But this approach, fails to address the security issues pertaining to MPMSD, like part integrity, reuse of parts, non-repudiation of sending as well as receiving the document etc. To address these issues an in-line Trusted Third Party(TTP), called an arbiter is mandatory. The arbiter serves as the trusted intermediate agent in between the current reviewer and the next. To make a protocol as general as possible, researchers attempt to avoid the use of an arbiter. However, to provide non-repudiation with time information, an in-line TTP is necessary. To provide non-repudiation of a digital signature, time of the signature is essential as the key used in the signature may become public at a later time. In our environment, documents are persistent and so non-repudiation of a digital signature is essential. So an in-line TTP will be required. Further, to prevent the reuse of parts, an in-line TTP will evidently be required as immediate detection of such reuse will be necessary to prevent the taking of wrong decisions. Since an in-line TTP is mandatory for addressing issues like repudiation of sending and receiving documents, we can address other issues on MPMSD as well, with an arbiter as an in-line TTP. A protocol for production of MPMSD with an arbiter is proposed, where

a reviewer submits a comment(part) for a MPMSD to the arbiter and the arbiter adds the part to the MPMSD. This protocol addresses the security issues of MPMSD. A three-tier conceptual architecture with split logic is also developed.

In most of the office solutions, public-key based digital signatures are used. Commonly, the RSA algorithm is used. The computational complexity of the algorithm is high. Therefore, digital signatures based on RSA are slow. In an e-office a reviewer may have to sign many documents per day. This may lead to performance degradation of an e-office system. In our scheme, we have a novel idea of using signed session keys for the digital signatures of the reviewers on a MPMSD. Actually, this idea makes our protocols efficient. During a session, each reviewer has a session key, and a signed copy of the same, signed by the reviewer and certified and time-stamped by the arbiter, is available to only the reviewer and the arbiter. Now, since the session key is known only to the reviewer and the arbiter and the arbiter is trusted and will not cheat as per our assumption, any message encrypted with this session key during the session can be treated as the digital signature of the reviewer on the message.

Authorization is an important security aspect of DPW. All users of an office are not authorized to access all pages of the page cube. Authorization in DPW environment is dynamic in nature. Synchronization of *authorization flow* with the workflow is a fundamental security requirement in workflow environment. Other essential requirements include role based security policy, separation of duties and negative authorization etc. An event-based dynamic DPW Authorization Model (DPWAM)is proposed as an extension of the page cube model.

Finally, we tried to answer the question, where does our research work

stand in the space populated by the current academic as well as commercial solutions for office automation, like POLITeam project, Lotus Notes etc. followed by a discussion on implementation issues of the page cube in relational database as well as in XML document database models. Some conclusive remarks and future directions of work are also given.

Contents

List of Figures

Chapter 1

Introduction

In today's fast moving and competitive world, information is considered as an important and vital resource of an organization. *Timely, accurate* and *relevant* information is important for effective decision making to achieve the objectives of an organization. To ensure that, every organization has a set of centres for processing organizational information and such centres are called *offices.* Therefore, an office can be viewed as an information processing centre. An office receives, stores, structures, processes and provides information. Information is thus the basic commodity of an office. Like any other commodity, it is produced, stored and distributed. In a conventional office, information is captured in documents. Thus a document is one of the most essential objects present in an office. Therefore, production, storage and distribution of documents occupy a major portion of office activities.

With the increase of complexity and competition in the world, modern offices have to deal with a huge volume of information. Paper-based offices are failing to serve the basic purpose of ensuring timely, accurate and relevant information. Moreover, paper-based documents consume a large amount of storage space. It also takes longer to retrieve a document and process it.

1.1 Document Production

Document production is a major activity in an office. Development of word processors is an important event in office productivity. A digital document has two aspects to look into: the content and the presentation. The content is captured as normal text encoded with some markup elements for processing and presentation. The markup is an encoding for making a particular interpretation of structure, layout etc. of the body of a text clear and explicit during processing and presentation. There are two types of markup languages: *procedural* and *descriptive*. Procedural markup languages are not human readable and also not application independent. The descriptive markup languages on the other hand are both human and computer readable and application independent. The *Standard Generalized Markup Language (SGML)* is an international standard for the formal definition of device, system, and application independent digital text using descriptive markup. *Hyper-Text Markup Language (HTML)* is a derivative of SGML. HTML is about *describing* content with annotations for presentation. It lags in its ability to describe the semantics of a document. Recently developed markup language, which is said to define the content of the future Web, is *Extensible Markup Language (XML)*. The tags of XML are user defined and hence flexible. Thus content of a page can be organized more semantically using XML tags. Detailed specifications and standards of these markup languages are available in the web site *www.w3c.org.*

Compound document is another concept increasingly becoming popular. It is a component based software concept. Component software is based on the notion of a component, a reusable object, which can be plugged into other components from other vendors with relatively little effort. When components are used in content-centric documents, the resulting document contains components or parts pertaining to different applications and is termed a compound document. A compound document is a *multi-part document.* A simple example of a compound document is a Microsoft WORD document consisting of a part belonging to a graphic package, another that is a spreadsheet and another containing a paragraph belonging to the word processor. The compound document model with different parts that pertain to various applications has important implications for groupware systems. Perhaps the most important of these is that the individual parts could actually be located on physically distributed sites in a network. Thus the components or parts can inter-operate and be located not only in different geographical site in a network, but also on different operating systems. The two leading proposals for compound document interoperability are *OLE2 (Object Linking and Embedding)* standard from *Microsoft* and *OpenDoc* from Component Integration Laboratory(CILabs) [25].

1.2 Document Storage and Retrieval

During the last two decades significant development has occurred in storage technology. In hardware, the development of high capacity storage devices like Microfilm, Microfiche, hard disks, optical disk etc. are the milestones for storing both operational as well as analytical data. In the software side, a milestone in storage and retrieval of information is the development of the concept of databases. The evolution of database systems is marked by three generations. *First generation* database systems included the *hierarchical* and *network* database systems . These systems are implementation dependent. *Second generation* database systems included the *Relational Database Management System (RDBMS).* RDBMS introduced the concept of data independence. *Third generation* database systems, consisting of *Object-Oriented Database Management System (OODBMS)* and *Object-Relational Database Systems (ORDBMS)*, accommodate complex structures in the data model, thus improving the expressive power of the model. However the increased expressive power also implied an increased complexity of the query languages [35]. In addition to these three generations, consisting mainly of generic databases, emergence of domain specific database systems in recent years pave the way for *fourth generation* systems. This generation includes *document, active, spatial, temporal, multi-media* database systems. Out of them document or text database systems are of special interest for office automation. Document databases are deserving more and more attention, due to their diverse applications: World Wide Web, paper-less offices, digital libraries etc. Research on document database models emphasize for efficient storage and retrieval. The pertinent problem is to extract relevant information from these documents. Documents may be unstructured or semi-structured.

1. *Unstructured*: Unstructured documents are the plain texts or ASCII texts without any form of tagging or structural information. The unstructured document is assumed as a sequences of words or phrases including keywords and stop words - often referred as *terms*. In conventional Information Retrieval(IR) methods, documents are retrieved against a query by matching keywords belonging to the document with those belonging to the query, defined as a Boolean combination of keywords. Vector space model [16], signature file model, inverted file model [11] etc. are the standard works based on this approach.

2. *Semi-Structured*: Researchers have found semi-structured data to be different from fully structured data, like relational or object oriented

data. It is defined as irregular or incomplete data and whose structure change rapidly and unpredictably. The Lore project, started in Stanford University around 1995, (*http://www-db.stanford.edu/lore*), is a pioneering work in this direction. The major contributions of the project are: a complete database management system for semi-structured data, a schema-less self-describing data model, called Object Exchange Model (OEM) and a query language, called Lorel. The first public release of XML version of Lore was made in May 1999. Recent works, like SGML document databases[35] and XML document databases[40], also belong to this approach.

1.3 Data Warehousing and Data Mining

Another significant development is a conjugate pair of concepts: *data warehousing* and *data mining*. A data warehouse is a wider perspective of a database. Data warehousing is the process of integrating enterprise-wide corporate data into a single repository from which end-users can easily run queries, make reports and perform analysis. The basic criterion is that a data warehouse holds *read-only* data where as a normal database holds operational data. A data warehouse is important for heterogeneous database integration. Many organizations typically collect diverse kinds of data and maintain large databases from multiple, heterogeneous, autonomous, and distributed sources. To integrate such data, and provide easy and efficient access to it is sought to be done through Data Warehouses[28].

The term data mining refers to the finding of relevant and useful information from databases. Data mining in databases or data warehouses is a new interdisciplinary field with the merging of ideas from statistics, machine learning, databases and parallel computing. The fundamental goals of data mining are *prediction* and *description*. Prediction makes use of existing variables in the databases to predict unknown or future values of interest and description focuses on finding patterns describing the data and the subsequent presentation for user interpretation.

1.4 CSCW and Groupware

In the next phase, we observe automation of group work. As a result, a new field of identifiable research in computer science has emerged, where role of computer in group work is focused. The new field is called *Computer Supported Cooperative Work (CSCW)*. CSCW [17] is defined as computer-assisted coordinated activity, such as problem solving and communication carried out by a group of collaborating individuals. CSCW addresses the organizational issues in collaborative work done by a group of individuals and the multi-user software supporting such collaborative work is termed as groupware [6]. Groupware represent a paradigm shift for computer science, one in which human-human rather then human-machine communications and problem solving are emphasized. The paradigm shift has resulted from a number of converging phenomena, like, pervasive computer networking, workgroup computing, increasing interest in telecommuting, electronic mail etc. Groupware is distinguished from normal software by the basic assumption it makes: groupware makes the user aware that he or she is a part of a group, while most other software seeks to hide and protect users from each other. The major group work supported by groupware are coauthoring of documents, conferencing, meeting scheduling etc. The majority of CSCW applications are fundamentally distributed and are dependent on the facilities provided by the existing distributed computing platforms. People cooperate synchronously and asynchronously. Synchronous cooperation requires the presence of all cooperating users, while asynchronous cooperation occurs over a longer period of time and does not require the simultaneous interaction of all users. A traditional problem with cooperation in distributed systems is the need to recognize the autonomy of individual sites in a network. Indeed, full cooperation and full autonomy are actually two extremes in a spectrum of possibilities. Increasing autonomy of a system decreases the support for cooperation and vice-versa [7].

1.5 Workflow

We have seen that the initial stage of automation of office work is to use computers for word processing. The next stage is to use a database to store information. The current trend is to move towards what is termed *work processing.* One of the major component of work processing is *workflow automation.* It is a major component of CSCW. An office work comprises of a set of tasks. The genesis of workflow automation is in accomplishing the tasks of an office work in a predefined order by routing the objects of work in the predefined routes following predefined rules by some roles. Rules define both the conditions, the workflow must meet to traverse to the next step and how to handle exceptions. Roles define job functions independent of the people who do it. A *Workflow Management System (WfMS)* is a software system that supports the specification, execution and management of workflows. A job in a workflow system is known as a *case.* Workflow coordinates user and system participants together with appropriate data resources, which may be held on- or off-line to achieve defined objectives. The coordination involves passing tasks from participants to participants in correct sequence, ensuring that all fulfill their required contributions and take default actions when necessary [20].

A WfMS is a client/server application where the client is called a workflow client and the server is called the workflow engine. The workflow client contains a workflow description tool for designing a workflow template and a workflow activation tool for activating the workflow template. Handshaking between the client and the workflow engine when starting a workflow, terminating a workflow and suspending a workflow, is done by the activation tool. The workflow tracking tool displays the status of various active workflows including the time taken to complete various tasks. The workflow engine provides workflow management services like interpretation of workflow templates, route management, rule management, workflow tracking management, user and role management etc. It interacts with workflow databases through interfaces and uses persistence and concurrency control capabilities of relational DBMS or object-oriented DBMS to allow workflow objects to be defined, created, searched and updated [20]. Workflow is today considered as the heart of E-Business. Some of the common workflow software are Office.IQ, WorkMAN, Visual WorkFlo, CabinetNG etc. Lotus Notes/ Domino and Oracle contain a WfMS as an important component. Standardization of WfMSs is done by the Workflow Management Coalition (WMC) formed in 1993. It is a field of active research now.

1.6 Security

Security is a major concern in an office environment in which the computing is usually distributed. The security aspects in an office are not limited to secure transmission and reception of documents. Documents have to be signed and they have to be stored securely. The proof of receipt and the proof of sending documents have also to be stored securely. With documents having a long life time, the issue of repudiation of signatures has to be handled. There are significant advancements made in cryptography during the last few decades and office software today uses these cryptographic concepts. Office related security concepts are discussed below. The security in a digital office can be discussed under the following heads:

- **User Authentication**: User authentication establishes a level of confidence about the user's identity. The main objective of authentication mechanism, in general, is to identify an entity uniquely and unforgeably.

- **Document Security**: Once a user has been authenticated, how does a recipient of a document know that it originated where it says it? How to ensure that the content of the document has not been tampered with? How to resolve the cases of repudiation of signing, sending and receiving the documents? How to authenticate the time of signing? How to maintain the confidentiality of the document? Document security addresses these issues.

- **Storage Security**:The office documents normally have a long life time. The security of the persistent office document and the evidences of occurrence of events on office documents like signing, sending, receiving etc. during storage is provided by storage security using encryption and proper authorization mechanisms.

- **Transport Security**: Since documents will be transported from one point to other geographically distant place through computer networks, it should be resilient to network attacks. Hence we need a secure channel for document flow across a network.

1.6.1 Digital Signature

In case of paper documents, hand written signatures on the document resolve the issues of document security to a legally acceptable level. From the difference of handwriting, ink used etc., forgery cases can be detected. Confidentiality can also be provided using sealed envelopes. Handwritten signatures provide only an imperfect solution to these requirements. It has several weaknesses. The weaknesses are: forged signatures are very hard to detect without genuine samples to compare with; it does little to prevent the alteration of a document: that is, it cannot maintain content integrity. Witness signatures are often added to a document to authenticate the main signature, but they suffer from similar weaknesses. Despite these imperfections, handwritten signatures are widely used as an authentication technique for paper documents. Equivalently, for digital documents we have several digital signature schemes [34, 39, 26, 22]. Digital signatures are analogs of handwritten signatures. The ability to provide a digital signature depends on there being something that the principal, who is the original signatory can do that others cannot. Confidentiality can also be provided by using a sealed digital envelope, created by encrypting the document using suitable encryption keys. A digital signature of a message is a number, dependent on some secret, known only to the signer, and, additionally, on the content of the message being signed. Signatures must be verifiable. If a dispute arises as to whether a party signed a document, caused by a lying signer trying to repudiate a signature it did create, or a fraudulent claimant, an unbiased third party should be able to resolve the matter equitably, without requiring access to the signer's secret information. Digital signatures addresses the issues of user authentication, content integrity, non-repudiation and certification.

Public-key cryptography is generally used for digital signatures. Each user has a pair of conjugate keys: a secret key and a public key. The secret key is known only to the user concerned and the public key is public, that is, known to all. A message encrypted with the secret key can be decrypted with the conjugate public key or vice-versa. Let A and B be the two communicating users. s_A and p_A are the secret key and public key of A respectively. Similarly, s_B and p_B are the pair of keys of B. B knows p_A of A and A knows p_B of B. Also, $\{m\}_k$ denotes message m is encrypted with a key k and $\{\{m\}_{s_A}\}_{p_A} = \{\{m\}_{p_A}\}_{s_A} = m$. A can send the signed message to B by transmitting $\{m\}_{s_A}$. On receipt B can verify the signature of A by decrypting the signed message with p_A to get $m = \{\{m\}_{s_A}\}_{p_A}$. Confidentiality can also be incorporated with the signed message by transmitting $\{\{m\}_{s_A}\}_{p_B}$ so that only B can read the message. Since s_B is known only to B, B can decrypt the transmitted message with s_B and then verify the signature. Now, so far as efficiency is concerned, encryption in public key cryptography is much slower than the encryption of symmetric cryptography. Therefore, instead of the entire message m, a digest δ_m is encrypted with the secret key of s_A of A. This serves both the purposes of origin and content integrity of the message. A message digest is a fixed-length bit string computed from the arbitrarily long message using a one-way hash function. It is discussed in detail in [34, 22].

Let $\delta_m = h(m)$ be the digest of message m , where $h()$ is a one-way hash function. A can now transmit the signed message $\{m, \{\delta_m\}_{s_A}\}_k, \{k\}_{p_B}$, where k is a randomly generated symmetric key. B first gets $k = \{\{k\}_{p_B}\}_{s_A}$ and then decrypts $\{m, \{\delta_m\}_{s_A}\}_k$ with k. B can then verify the signature of A and the content integrity of the message. Content integrity is verified by recomputing a digest $\delta'_m = h(m)$ and then checking the equality $\delta'_m = \delta_m$. In principle, any public-key cryptographic scheme can used for digital signatures. More details on digital signatures can be found in [39]

1.6.2 Multi-Signature

In addition to the originator's signature on the digital document, supervisors are often required to sign office documents for verifying and approving an originator's message. In such cases several persons sign the same document. This is referred to as a multi-signature. Proprietary digital signature schemes cannot resolve the issues related to such multi-signature documents. As a result, the literature contain different multi-signature schemes. Signature schemes originally developed for single signatures, are also extendable to the multi-signature case. However, because of the increase in signature length, they are not satisfactory for use. Itakura and Nakamura [19] proposed a solution based on extended RSA scheme and resolved the problem of signature length. But their schemes needs to predetermine a hierarchical relationship among users. In some offices the hierarchical relationship either does not exist or cannot be predetermined in advance. Okamoto proposed a scheme [24] that overcomes these problems. In this scheme the signature length of a multi-signature in nearly equal to that for a single signature and the order of signing is not restricted. But it introduces the problem of key distribution. All the persons who are communicating among themselves should know one another's public key. This leads to the distribution on n^2 keys, which keeps on increasing exponentially. It puts a lot of processing overhead on the users. A user who is n^{th} on the list of persons reviewing the document then s/he will have to decrypt the document $n-1$ times with the public key of all the previous persons. The verification process is time consuming because a recipient must check the multi-signature by the reverse order of signing. Similarly Harn and Kaisler [18] also proposed a scheme of multi-signature. The issues addressed in the above schemes are mainly related to the length and order of signatures.

1.7 The Problem

From the above discussion, it is evident that, with the development of information technology different directions of office automation have received considerable attention. The technological infrastructure for a paper-less office is almost ready and there is rapid progress in both software as well as hardware aspects. But a vital and central problem, common to almost all offices, is missing from the research agenda on office automation. The problem is the production of Multi-Part Multi-Signature Documents(MPMSD). It is different from the multi-signature problem and the multi-part compound document problem discussed in the literature. In case of multi-signature or group-signature, a group of users can sign a single message. Here, content of the same message is authenticated by multiple people. For example, the minutes of a meeting is signed by all the participants present in the meeting. But in case of multi-part multi-signature each member of the group of signatories contributes a different message which is authenticated by the member by signing the message.

In this section an outline of the problem for the present research work is provided. The issues of the problem to be addressed are also identified and discussed. 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 within an office. 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 dispatched. We can term the process as Document Production Workflow(DPW) [37]. The resultant document of a DPW is termed as a Multi-Part Multi-Signature Document(MPMSD) [36]. 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 other parts in between are the comments of other reviewers, that means, the reactions. Each part of a MPMSD is signed by the corresponding reviewer. The first reviewer is also termed as the originator of the request.

1.7.1 A Scenario

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. 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, 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 University Grants Commission on travel expenditure, and adds her comment, m_C on the amount that may be granted and forwards it to the director, D. D also justifies the previous comments, 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 log-books. This is a case of the travel plan workflow. It is shown in figure 1.1

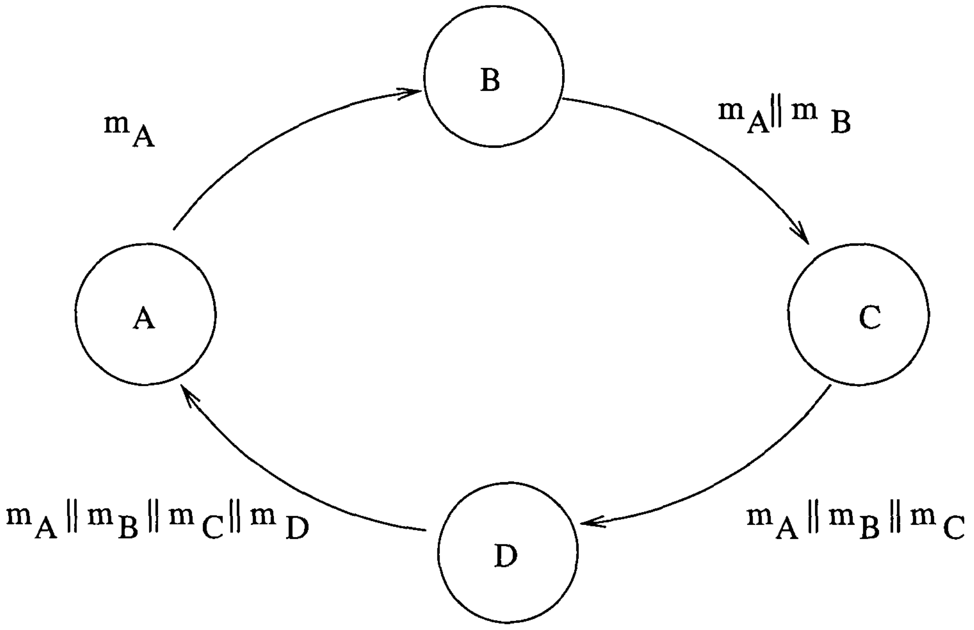

Figure 1.1: Travel Plan DPW

1.7.2 The Components of the Problem

The DPW has three components to study:

- *MPMSD*: It is the major component of a DPW. Since in our framework, an office document is produced as a case of a DPW, therefore, all documents are MPMSDs. It is a generic framework. Different parts belonging to different cases of different DPWs may also be integrated to form a MPMSD subject to satisfaction of certain criteria. For example, office orders, circulars, meeting resolutions produced as parts of cases of different DPWs but containing rules on a certain topic may form a MPMSD. Moreover, a document may have multiple versions produced at different points of time. Such multi-version documents are also MPMSDs. In a paper document system, it is the same paper document that is passed around and the proof that it has come through the proper channel is the series of comments followed by the signatures of the reviewers. In a digital system, there are several issues to be addressed for secure production and storage of MPMSDs. The issues to be addressed are identified and discussed in detail in the following sections. A single part document is a special MPMSD, where the total number of parts in the document is equal to one. Henceforth, in the rest of the discussion, a document means a MPMSD.

- *Context*: Just as a human being can develop *amnesia* and forget past experiences, an office can also experience loss of memory unless there is a proper framework to maintain *organizational memory*[15]. Huge collection of documents in an office is the major constituent of its *organizational memory*. Contemporary offices have only a weak ability to remember and learn from the past. What is missing from organizational memory is the *context* or rationale that lay behind these documents when they were created. 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. In a formal office rules are framed almost on all topics to prevent the possibility of arbitrary decisions. Rules are generally well defined. When rules are either not defined or not well-defined we look for similar cases handled earlier, that is, *precedents*. Here, rules include regulations, office orders, meeting proceedings etc. and the precedents are the already produced cases following concerned rules. Certain decisions require *support documents*. For example, a purchase indent to sanction a purchase. Rules, precedents and support documents constitute a *reference space*. A reviewer navigates through a subspace of the reference space before producing a new document and draws citations wherever necessary to substantiate the rules position, the precedent position etc of the new document. This subspace is called the context of the document. The process of navigation through the context is called the *case examination*.

- *Linking*: Links establish relationships among different documents as well as different parts of a document. As soon as a document is created in an office, the new document may be implicitly linked to many documents. Moreover, a document may also be explicitly linked to many more documents at a future point of time. For example, documents on the same topic or of the same type or created within a certain period are implicitly linked, where as document cited in another document are explicitly linked. Therefore, linking is an important component to be studied in an e-office.

1.8 The Security Issues

There are several security issues related to secure production and storage of digital documents. Some of the issues are general in nature and some are specific to DPW system. The security issues are discussed below.

1.8.1 Principal Authentication Issues

A fundamental concern for a secure office system is the authentication of the principals involved in the system. Authentications of the principals are usually done by using their credentials. A credential is a piece of information that is used to prove the identity of a principal. Passwords, digital certificates, secret keys etc. of the entities are the important credentials. In the office system the principals are the users and the processes. Therefore, the issues are:

1. *User Authentication*: All the office workers (users) of the system need to be authenticated through standard challenge-response protocols during session set up using the credentials of the users.

2. *Process Authentication*: Processes are the entities who speak for users during run-time[21]. A set of processes who speaks for a user may share the credentials of the user. Processes are to be authenticated when they try to access any object.

1.8.2 The Production Security Issues of a Part

For every individual part of a MPMSD, the security issues are as follows: Let A and B be legitimate principals and let A send a signed message m_A to B. The issues are-

1. *Proof of Origin*: It should be verifiable by B or any third party that the message m_A was really signed by A and not forged by an intruder.

2. *Content Integrity*: It should be verifiable by B or any third party that the content of the message m_A was not illegally modified by a intruder. Even the originator of the document, A, is not allowed to modify its

content after it is dispatched to B. The first part of this issue can be taken care of by digital signatures using fixed length message digests generated by one-way hash functions. The second part is the more interesting point to look into in office automation.

3. *Confidentiality*: It is required that the message m_A be accessible for reading only to the authorized principal B, to whom it is addressed, and not to any eavesdropper.

4. *Repudiation of Signing*: The success of a digital signature scheme using public-key cryptography pivotally depends on the secrecy of the secret key. Even if the message m_A is signed by the secret key of A and successfully verified as in issue 1, A can repudiate the signature with the pretext of compromise of the secret key of A and can thus disown the responsibility for m_A.

5. *Repudiation of sending and receiving*: If A or B repudiates the sending or receiving respectively of the message m_A, then it should be verifiable by any third party from the stored evidences of the flow of the message. The evidences may be the proof of sending and the proof of receipt. The evidences should be acceptable as legal and irrefutable proofs. The repudiation may be on the time of sending or of receiving the message. Moreover, A and B may collude to remove the evidences of transmission of the message. Therefore, the records should also be tamper-proof.

6. *Signature Replacement*: If another principal X, with the cooperation of B, tries to replace the digital signature of A on m_A by its own signature and claim the ownership of m_A then such an issue should be resolvable. In digital signature schemes, based on public-key cryptography with

one-way hash functions, this issue is not addressed. In online communication this can be taken care of by a digital envelope which ensures the secured transmission and reception of message m_A and digital signature σ_A on it but in the problem domain with persistent storage, it is not sufficient. Even the recipient B can replace the signature with its own since there is no certification of association of m_A and σ_A. The message digest $\tilde{m}_A$ of m_A included in σ_A can be created by anybody having m_A since the digest function is public.

1.8.3 The Production Security issues of the Whole MPMSD

Apart from the security issues of each individual parts of a MPMSD as mentioned in section 1.8.2, the following special security issues related to a MPMSD as a whole are to be addressed additionally.

1. *Part integrity of a MPMSD* : Apart from the content integrity of each part, we also need the total part integrity of the whole document. The content integrity of all parts individually does not necessarily imply the integrity of the whole document. All parts must remain in order in which they were added to the document. Removal of some parts and reordering of parts should not be allowed.

2. *Reuse of Parts* : Reuse of parts should not be allowed. Suppose, the ordered list of reviewers of a MPMSD is (A, B, C, D) as in the example given in section 1.7.1. If D does not like what C has written on the document (m_C) then D may cooperate with B to have B mark the document directly to D, bypassing C. B can do this by using the document $m_A || m_B$ passed to it by A and reusing it.

1.8.4 The Storage Security Issues

1. *Authorization Flow*: During the production of MPMSDs, the authorization also flows synchronously with the document flow. Only the latest reviewer of a document can add a part to the document and only she can read the previous parts but cannot modify or reorder the previous parts. After signing and forwarding the document to the next reviewer of the document the current reviewer loses the privileges of being the latest reviewer. The author of a part is not allowed to modify her previous parts, if any, even if she is marked again as the latest reviewer. Moreover, a user has privileges as long as he/she is assigned to the role of a reviewer of the DPW.

2. *Authorization Constraints*: During case examination, a reviewer accesses the documents belonging to the context. But the accesses are subject to some authorization constraints. A reviewer may be allowed to read the precedent cases but may not be allowed to read all the parts of it. For example, the originator, that means, A in the scenario given may be allowed to see only the last part, the office order m_D and her application, m_A, but not the intermediate comments. Some reviewers may not be allowed to see all the comments given by the higher authorities. Similarly, all rules and other support documents may not be accessible to all reviewers. It depends on the security policy of the office concerned. But the system should have provisions for such authorization mechanism.

3. *Secure storage of session keys*: Session keys play an important role in our system both for digital signatures and secure transport of MPMSDs. Generation of a session key should be unique for a particular user, be-

cause a session key speaks for a user both during the session and in future for persistent documents. Signed copies of the session keys should be stored securely.

4. *Secure storage of document flow records*: The records of the flow of documents along with the evidences of non-repudiation are to be protected from unauthorized access. The records need to be well structured to enable efficient tracing of a document with its latest state, that is, at a particular time where the document is lying and at what state. The tracing of documents is to be allowed only to authorized users.

5. *Security of program codes*: The program codes designed to implement the system are to be stored securely in storage. They are to be authenticated using the credentials during loading.

6. *System administrator threat*: The system is on top of the OS of a host system. Even though the operating system of the host system is assumed to be trusted the administrator of the host system should not be allowed to access the objects of the system without proper authorizations.

7. *Validation of Old Documents* : If the principal A feels that her secret key has been compromised and the key is consequently changed then all the documents signed by A using the earlier key will be invalid. Now, the issue is how to validate such old documents and how to disallow the use of the old key of A.

8. *Archival of Old Documents*: The documents no longer active and not referred frequently are to be archived as normally done in the record rooms of an office. The archived documents will have life tags attached

signifying the permanence of the information stored in the document. After the expiry of the life, the documents may be deleted from the system. Only archived documents may be deleted from an office.

1.9 The Management Issues

Apart from the security issues there are important management issues to be addressed.

1.9.1 The Context Management Issues

1. *DPW Context*: With every DPW, a default initial context is attached. A reviewer can add more items to it during case examination, but cannot remove any item from the existing context. As a result, the context may grow as the document flows from the current reviewer to the next. Moreover, during case examination, a new relevant rule or a new version of the existing rule, a new case or a support document may come up, which should be automatically added to the concerned contexts. Therefore, a DPW context consists of all the documents relevant to the DPW in general.

2. *Drawing Citations*: During composing a part, a reviewer should be able to draw citations directly from the context, so that a hyper linked address of the cited document is automatically included in the comment.

3. *Case Context*: The DPW context provides the documents relevant to all the cases of the DPW. The DPW context attached to a DPW changes with time. The context of a case is the context specific to the case. It consists of the DPW context and some more documents relevant to the

specific case. Moreover, a case is a MPMSD. Different parts of the case may be produced at different points of time. Therefore, the context of a part of a case consists of the state of the DPW context and the other documents included in the case context at the time of production of the part. During case examination, when a reviewer peruses a part of a precedent case, he/she should be able to retrieve the state of the case context at the time of creation of the part of the case. The state of the case context is defined by which rules and which versions of the rules, which precedents and support documents were available at the time of production of the part.

1.9.2 Storage Management Issues

Apart from the security issues of stored documents there are a few more management issues to be addressed.

1. *Organization of Documents*: An office has a huge volume of documents. Tracing a particular document and then retrieving it for perusal in an efficient way is a major issue in any office. Therefore organization of documents is a very important aspect. In a paper-based system office documents are organized in folders and folders in cabinets etc.. Within a folder documents are organized either as a stack or as a queue. As a result we get a linear organizational structure. Related documents are stored in the same folder. But, as the relations among the office documents are non-linear and complex in nature we find unnecessary duplication of documents and inefficient retrieval of documents. For efficiency, the organizational issue is to be addressed.

2. *Invariant Document Address*: During the life-cycle of an office document, it has certain degrees of mobility. The mobility may be attributed to the documents due to document flow, archival activities and system management. Therefore, address of the document may change from time to time. Since many documents may be hyper linked to a particular document, change of all such hyperlinks as soon as the location of the document changes, will be an inefficient proposition, if not impossible. Therefore, an invariant document address is required for mobile office documents.

3. *Performance*: In course of time, the volume of office documents will be large which may lead to performance degradation of document storage and retrieval. Therefore, measures are to be taken for enhancing efficiency.

4. *Reliability*: The reliability of the storage system is a pivotal issue. The entire system depends on it.

5. *Reverse Linking*: An explicit link comprises of a conjugate pair of directed links: one forward and one reverse. As soon as a forward link is established, the reverse link is to be established automatically. For example, an office document may be cited in many different documents at different points of time. If we want to study the effects or the reactions on the office order within the office then such reverse linking is a fundamental requirement.

1.10 Scopes and Goals

Almost all the work done in different directions of office automation discussed above have failed to address the DPW problem. With the advent of workflow solutions, similar problems are coming to focus. But the genesis of workflow automation is on the automatic routing of documents and the automatic execution of the tasks based of the rules and roles defined in the workflow design. It is rigid in nature. As a result we see success of workflow systems in practical application areas like manufacturing, where the flow of work is almost static. But what is required in a real life office today is a flexible, co-operative workflow tool which assists the office workers in reviewing the documents by providing facilities for secure production, storage, case examination and flexible routing. Our study aims towards incorporation of such a tool in future office automation software. A similar problem was studied in the POLITeam project [27] . It addressed the problem of multiple versions during the production of a speech in a German ministry. We found that this multi-version speech production workflow is a special case of our more generic framework of MPMSD where each part is a version of the speech. Single-part documents discussed in the literature for office automation are special cases of multi-part documents. Secure production and storage of persistent multi-part document constitute the major part of office work. But almost no effort has been made to solve this problem.

As the above discussion shows, digital signature schemes developed for general digital documents addresses the security issues of single-part documents but do not address the security issues of multi-part documents. The multi-signature schemes developed till date also do not address even the basic issues of MPMSD. Most successful, widely used, and a robust groupware product Lotus Notes/Domino also does not address the issues of MPMSD.

Therefore, there is sufficient scope for further research in this field.

The main goal of our research is to identify the issues of persistent multi-part digital documents in an office environment during production and storage and provide solutions to address the issues. The output of our research may be the input to future commercial software for paper-less office. We provide a design framework for multi-part document system and protocols to address the issues of multi-part documents using a trusted third party, called an arbiter. We also provide a storage model for automatic generation of context of a workflow. The present work does not provide a complete solution to a paper-less office. The study is limited only to the problem of persistent multi-part office documents, their production and storage within a single office and under a single arbiter. Moreover, by document in this work, we mean, text only documents, excluding multimedia documents and compound documents.

In the present chapter, we reviewed the directions of office automation, outlined the problem of DPW and identified the different issues of the problem. The rest of the thesis is organized as follows. In Chapter 2 we discuss a production and storage framework and a conceptual architecture for DPW. In Chapter 3 a model, named Page Cube, for storage and retrieval of documents is presented. In Chapter 4, we discuss the security framework and propose a protocol for secure production of MPMSD using a neutral arbiter as an in-line TTP. In Chapter 5, we discuss production of contexts. In Chapter 6, we discuss the aspects of authorization of pages in a page cube. Some implementation issues in relational as well as in XML database models are discussed in Chapter 7. In Chapter 8, we try to answer the question, where does our research work stand in the space populated by the current academic as well as commercial solutions for office automation, like the POLITeam project, Lotus Notes etc. Some conclusive remarks and scope for further work are presented in Chapter 9.

Chapter 2

The Production and Storage Frameworks and an Architecture

In this Chapter, frameworks for production and storage of digital documents in an office are studied. Based on the frameworks, a conceptual architecture is also designed. The problem of production and storage of digital documents in an e-office can be studied under the following frameworks. These are not formal, but design frameworks.

2.1 The Production Framework

Document production in an office can be studied under a generic framework comprising of three concepts: *Document Production Workflow (DPW)*, *Context* and *Multi-Part Multi-Signature Document (MPMSD)*.

An office document is an output of an *office task*. The office task, concerned with a DPW is termed, in general, as *review*. The review task comprises of a set of operations: *case examination, composition, signing, filing, linking and flowing.* A review task may have specific names at different positions of the workflow. For example, in the travel plan workflow discussed in Chapter 1, the different review tasks may be termed as travel request, departmental approval, financial approval and final approval.

Definition 2.1 *An* **office 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*,* $T_{IN_i} \subseteq TO$ *is the set of object types allowed as inputs,* $T_{OUT_i} \subseteq TO$ *is the set of object types expected as outputs.* TO *is the finite set of object types.*

An office work comprises of an ordered set of tasks. Various tasks are usually carried out by several office workers in accordance with the organizational rules relevant to the office concerned. The output object of one task may be the input of the next task. As a result, work objects flow from one task to another. Therefore, office work can be ideally represented by a workflow model. When the work object is a document, we term it Document Production Workflow (DPW) and the output document thus formed from a DPW is termed as a *case.*

Definition 2.2 *A* **Document Production Workflow** *can be represented as a directed graph* $W(T, E)$*, whose vertices* T *represent the set of tasks,* $T = \{tw_1, tw_2, tw_3, \cdots, tw_n\}$ *in the workflow and the edges* $E = \{(tw_i, tw_j) | tw_i, tw_j \in T\}$ *represent the stages of the flow of the case under review. Both the vertices and the edges are labelled. In the edge* $tw_i \rightarrow^k tw_j$*,* k *signifies that the case under review is in the* k^{th} *stage. This signifies that the ordinal number of the latest comment, added to the flowing case so far, is* k*, assuming the ordinal number of the original request as* 1*. The label on a vertex signifies the reviewer associated with the task.*

All documents in an office constitute a document space and a subspace of it constitute a reference space. The documents in the reference space are referred or cited while producing a new document or a part of it. Only a subset of the reference space may be relevant to a particular DPW. This relevant set of documents constitutes the context. Thus every DPW has a context associated with it. The context comprises of the rules, precedents and other support documents relevant to the DPW.

Definition 2.3 *A* **Document Space** $\mathcal{D}$ *is the universal set of uniquely identifiable documents in an office.*

Definition 2.4 *A* **Reference Space** $\mathcal{R} \subseteq \mathcal{D}$ *is a collection of documents of types rules(R), precedents(P) and other support documents(S), which can be consulted and referred to while generating other documents. Rules are the the documents containing formal guiding principles, policies etc. on all topics of concern in an office. Precedents are the cases handled earlier. Support documents are the documents which are neither rules nor precedents but are consulted or referred to while generating other documents.*

Definition 2.5 *A* **DPW Context** $\mathcal{C}_{\mathcal{W}} \subseteq \mathcal{R}$*, of a DPW* W*, is a collection of documents relevant to* W*. The relevance is defined as a predicate based on the different attributes of the documents in* $\mathcal{R}$*.*

During composition of a document, a reviewer of W navigates through the context, peruses documents belonging to C_W and draws relevant citations to the document under composition, from the context, in order to provide rules position, precedent position and other supporting documents to complement the position of the document. This process is termed as *case examination.*

Definition 2.6 Case Examination *is the process of navigation and perusal of the documents belonging to the context C_W of a DPW W done by a reviewer of W as an operation of the task $tw_i \in T$ of $W(T, E)$.*

A reviewer of W can add new documents from $\mathcal{R}$ to $\mathcal{C}_\mathcal{W}$ but cannot remove any document from $\mathcal{C}_\mathcal{W}$. Moreover, as soon as a document is added to $\mathcal{D}$, if it satisfies any relevance predicate of C_W then the document will automatically be included in C_W. Only the designer of the DPW, who is authorized to modify the DPW can redefine the predicates and thereby remove documents from the context.

Once the document is generated, it is registered in a proper way and related documents are linked for immediate pick up. The document then flows from its point of origin to the target.

If we consider the output of an office task as an elementary document, then the output of a DPW is a composite document, composed of multiple elementary documents, each containing comments of the corresponding reviewer. Such a composite document is termed as a Multi-Part Multi-Signature Document(MPMSD), where each part is an elementary document.

Definition 2.7 *A* **Multi-Part Multi-Signature Document (MPMSD)**, *D_W, produced in a DPW W, is an n-tuple, $n \geq 1$, such that*
$D_W = (d_1, d_2, d_3, \cdots, d_n)$.
Each part d_i in turn is defined as a 4-tuple $(m_i, c_i, \sigma_i, s_i)$, where m_i is the comment of the reviewer s_i, c_i is the context of m_i, based on which the comment m_i is produced, and σ_i is the signature of s_i.

A case is a MPMSD. Since, the parts of a case are produced at different points of time, the context of the DPW may be at different states at different points of time. As a result, the context of creation of any two parts of the same case may not be the same. Moreover, during case examination, a reviewer may cite another document, not in C_W, as a support for his comments. This document is specific to the case and may not be relevant to other cases of the DPW. Therefore, this document need not be included in the context C_W of the DPW W but it needs to be included in the context of the case.

Definition 2.8 *A case context C_{D_W} of a case D_W consists of the context C_W of the DPW W and a set documents, δ_{D_W}, specific to D_W, added to C_{D_W} at different points of time by the reviewers of D_W.*

Therefore, the context c_i of a comment m_i of a part d_i of a case D_W is defined by the state of C_{D_W} at time t_i, where t_i is the time of production of d_i. The state of C_{D_W} in turn is defined by the states of C_W and δ_{D_W} at t_i.

For example, let us consider the travel plan workflow discussed in Chapter 1. The workflow has a context comprising of documents containing rules on travel permission, on leave to be granted during travel, on TA/DA and precedent cases of travels granted or rejected. It also contains documents on budget provisions etc. as support documents. Each of the documents in the context is a MPMSD. An employee A composes her application citing some of these documents. A also wants to cite her leave account report on the date of application in support of her application. Since this report is specific to her case only, so she adds the report in the context of the case and submits an application m_A to B with a request to allow her travel to an organization. This is the first part part of a new case. Before forwarding the case to C, B wants to peruse her forwarding note on a similar case forwarded earlier. When B retrieves the concerned part $d_2 = (m_2, c_2, \sigma_2, s_2)$ from the precedent case, the context c_2 prevailing at the time of creation of the forwarding note m_2 is regenerated. c_2 is defined by all the documents and their parts available to the context of the case, of which d_2 is a part. B then appends her forwarding note m_B to the application and sends the same to C. C finds that a resolution taken in a recently concluded Finance Committee Meeting is relevant not only to the case in hand but also to all future cases of the workflow. Therefore, she registers the document containing proceedings of the meeting, of which she is the convenor, in the document space of the office first as a rule, which automatically qualifies for a member of reference space. C then adds it to the context of the workflow. Now, the comment m_C is created, citing the newly added document from the context, and is forwarded to the director D and D approves the case.

2.2 The Storage Framework

The main objective of storage of information in an office is to keep track of the *history* of *who* did *what, when, why* and *how.* Thus storage in an office serve as the organizational memory, where the documents are the neurons. Therefore, the central issue is to store the documents in such a way that they can be identified, located and retrieved in an efficient way. Moreover, from a document thus retrieved, all the related documents should be reachable in a simple and straight forward way. The framework of storage and retrieval of office documents comprises of the following concepts: *life-cycle, representation, identification, organization, relationship, efficiency* and *reliability* of office documents.

2.2.1 Life-Cycle

Normally, an office document has a long life time and during its lifetime it passes through different states. The life-cycle of a MPMSD can be represented by a state transition diagram. A state transition diagram is a digraph

$LC(S, O)$, where S is a finite set of states represented as vertices and O is the finite set of operations on MPMSDs represented as labels on the arcs.
$S = \{$born, active, reference, archived, expired$\}$
$O = \{$registration, addpart, close, reopen, archive, de-archive, burn$\}$
The states and the arcs of $LC(S, O)$ are labelled and the labels are the corresponding elements of S or O. On execution of an operation on a MPMSD, a transition occurs from a state to the next state of the life-cycle. The life-cycle is shown in figure 2.1. When a new MPMSD is created, it is in the

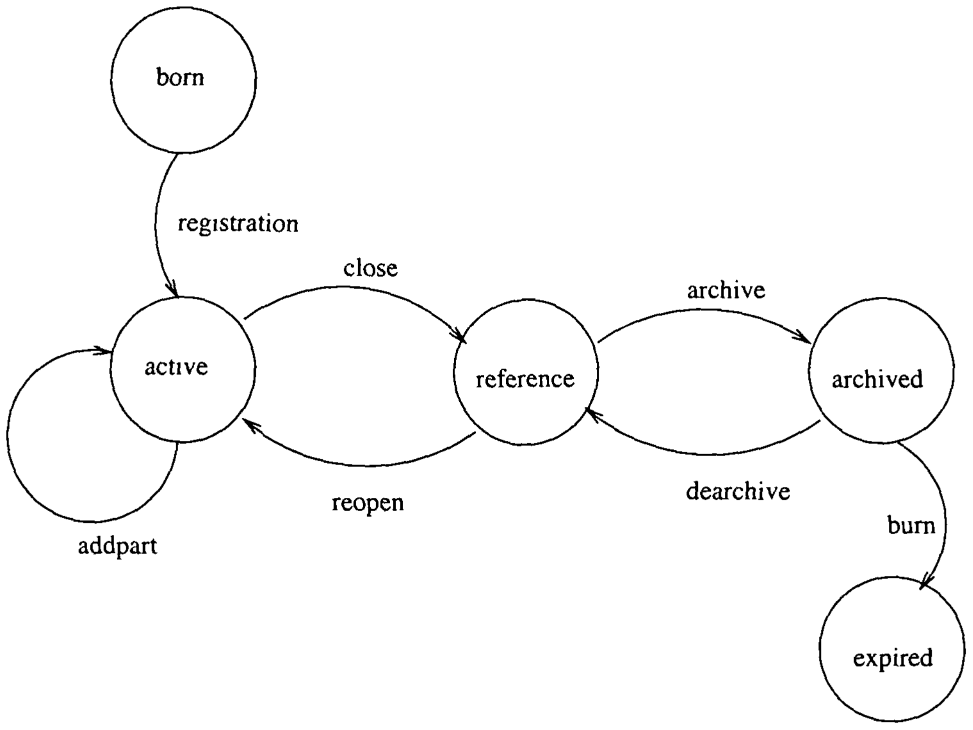

Figure 2.1: Life Cycle of a MPMSD

born state. On registration it transits to the active state. Registration of a new document means inclusion of a document in the document space $\mathcal{D}$ with a unique document identifier. Multiple parts can be added to a MPMSD in the active state. On execution of the addpart operation, the MPMSD remains in the same state. The addpart operation links a part to the MPMSD. If the part is a new one, the addpart operation also includes registration of the new part and it precedes the linking. On closing a MPMSD in the active state it transits to the reference state. Parts cannot be added to the MPMSD in the reference state. On reopening a MPMSD in the reference state it transits back to the active state again. On execution of an archive operation on a MPMSD in the reference state it transits to the archived state. Archival may include compression of the document. The reverse transition occurs on the operation de-archive. On execution of the burn operation, an archived document transits to the expired state, which is the final, and no return state. The burn operation physically deletes the document from the system. An archived document may be associated with a life-tag. A life-tag signifies the life of an archived document based on the values: permanent, semi-permanent, temporary, immediate etc. of the information contained in the document. On the expiry of the life in the tag the document will be automatically burned. Therefore, the de-archive operation is applicable only before the expiry of the period mentioned in the life-tag of an archived document. The main characteristic that differentiates these states is the accessibility. Archive objects have no access privilege for the reviewers. The reference objects are read only and the active objects may have privileges like read, write, modify etc.

2.2.2 Representation

An office document has three aspects: *profile, content* and a *presentation.* The profile represents a document.

Profile

A profile is the *bio-data* of the office document. It contains meta-data of the document which provides a detailed 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 the vector space model [16]. We excluded keywords from the discussion of our model but it can be easily incorporated. Unfortunately, keywords alone cannot adequately capture the office 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. The attributes are used in designing multi-dimensional Page Cube model, discussed in Chapter 3.

- *storage record*: This record consists of storage related attributes like address, size, authorization, state etc. of a document.

- *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.

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 not 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.

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 hierarchical 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 de-facto layout framework for digital pages. Thus the layout framework provides the *get up* of a document. For storage and retrieval of office documents, the focus of our discussion is on the profiles of the documents. Therefore, content encoding and presentation aspects of a document are excluded from the rest of our discussion.

2.2.3 Organization

There are different states of office documents as described in section 2.2.1. For persistent documents we are concerned with the active, reference and archived states. The documents can be organized in such a way that all documents in a particular state will be stored in the same data-storage. Accordingly, we can have three types of storages: *active storage, reference* *storage* and *archive storage*. A document may be stored in the respective category of storage based on the state in which it belongs. Categorization of storages provides a level of access control. For example, any attempt to access an archived document directly by a reviewer other than some authorized users like the storage manager will be denied straight away.

State transition of a document may induce migration of the document as well as other related objects like digital certificates etc from one storage to the other. The migration of objects from one storage to the other are recorded in the *migration table* maintained in the respective storages.

2.2.4 Identification

A document may be represented by the attributes of the profile but for identification of a document during storage and retrieval a unique address is required. In a paper-based system such a unique string is generated manually combining some of the attributes of the profile, which serves as the unique document identifier. In an e-office an equivalent identifier is to be generated automatically. In our framework, a document may migrate from one storage to the other as soon as it changes its state and accordingly the physical address of the document may change from time to time. But at a particular time a document will have one and only one physical address. Therefore, each document in the document space will have a pair of addresses: a time varying physical address and an invariant logical address.

2.2.5 Relationship

As soon as a document is produced in an office, the document is implicitly related with a set of documents belonging to the document space $\mathcal{D}$. The implicit relations are set up by virtue of production of the documents in the office concerned. For example, a newly produced document of a particular type on a particular topic will be closely related to other documents of the same type and/or on the same topic. Again, the types are hierarchically linked to each other. Thus, the documents are related further as siblings, ancestors and successors etc. Similar is the case with topics. On the other hand, a document may be explicitly related with other documents. For example, when a document is cited as reference in another document, then these two documents are explicitly related. In our framework, the relations among documents are set up with links. In DPW, links are of two types: *implicit link* and *explicit link.*

Definition 2.9 *Implicit links are the the links which are set up among documents implicitly based on the attributes of the profiles of the documents.*

For example, key-word links, type links, topic links etc. are the implicit links.

Definition 2.10 *Explicit links are the links which are set up explicitly among documents by the reviewers or by other entities of the system*

For example, citation links, part link etc. are explicit links. The links in our framework are bidirectional. Every link is a conjugate pair of directed hyperlinks: one forward and one reverse.

2.3 The DPW Architecture

A software architecture can be viewed as a style or a method of design and construction or strategic policies and patterns that shape a system. It provides a common definition in abstraction for different components involved in a system [23]. In this section, we present a conceptual architecture for production and storage of digital office documents. A preliminary version of the architecture was published in [36].

A common method in software architecture design is based on the multi-layering principle. Layers are popularly termed as tiers in software architectures. Each tier is an abstract representation of a perspective of the application. In the present architecture, different tiers, and different components in each tier, are identified and discussed. Common business application architecture of today comprises of three tiers, based on three perspectives: *corporate data* representation, *business logic* to manipulate the data and *presentation* of input output data or information. The architecture we propose here is also a three tier architecture consisting of the following tiers: *view*, *logic* and *data storage.* The view consists of user interfaces, the logic consists of processes to manipulate data and data storage stores data. These three tiers are again grouped into *Client* and *Arbiter* with split logic. The logic is split into *client logic* and *arbiter logic*, because the data manipulation will be done by the client and the arbiter cooperatively. The client contains the view and the client logic and the arbiter contains the arbiter logic and data storage. The client and the arbiter will communicate over a network using some protocols. This architecture is based on a centralized arbitration mechanism, where all the documents must be routed through the arbiter. The architecture is shown in figure 2.2.

The components of the architecture are described in the following sections.

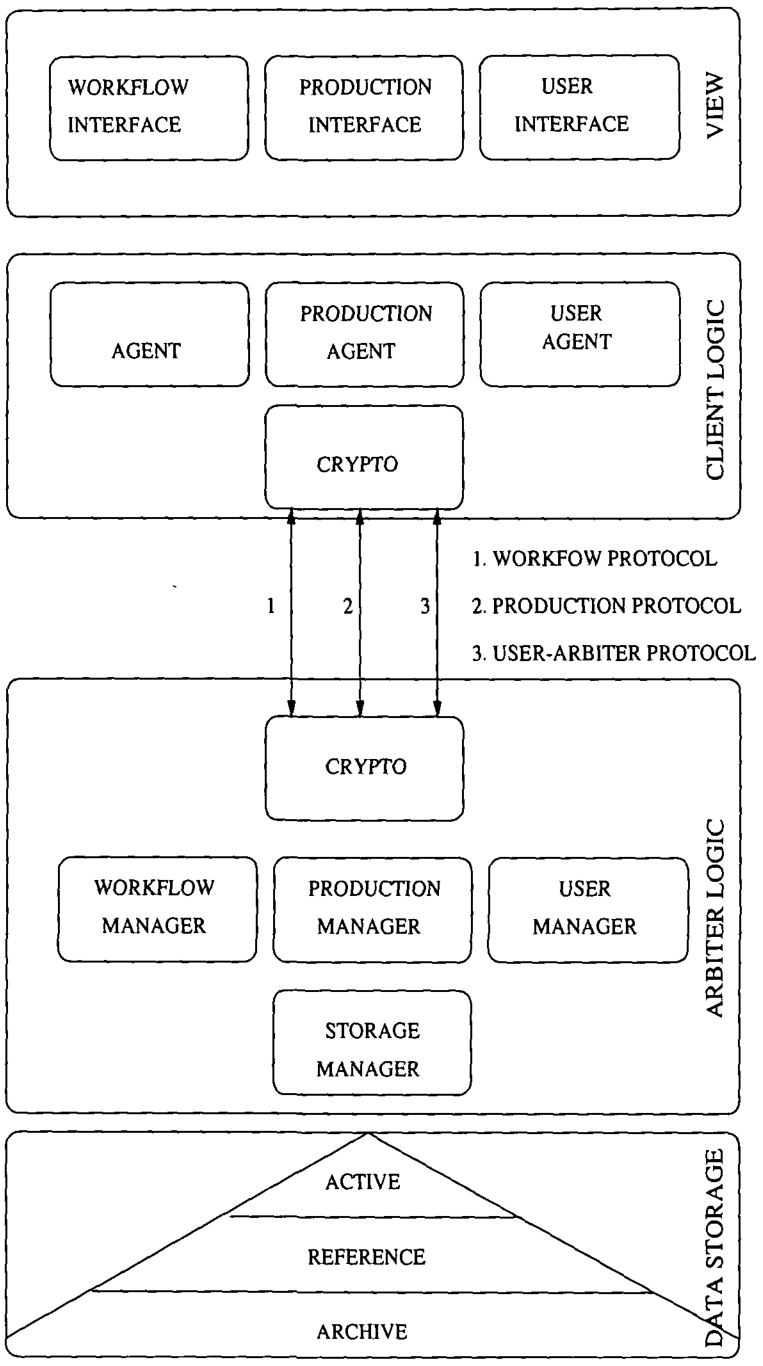

Figure 2.2: The Conceptual Architecture for DPW

2.4 The View

This tier gives us basically the Graphical User Interfaces (GUIs) of the system. The physical metaphor conceived here, for this architecture, is the office desktop. View comprises of three interfaces: Production Interface, User Interface, and Workflow Interface. A user interacts with the system through these interfaces. The interfaces are discussed in brief below:

2.4.1 The Production Interface

Through this interface a reviewer interacts with the production agent of the client logic module to review a case received. The reviewer examines the case in the context of existing rules, precedents and other support documents, composes his own comments, cites references from the context into the comment to establish the rules position, precedent position etc. of the comment, and forwards the case to the next reviewer. The main elements of this interface are:

- **Workspace**: It is the area where a document under creation or a document retrieved from the arbiter, will be displayed.
- **Inlog** and **Outlog**: These are the indices of documents incoming to and outgoing from this desktop respectively. The indices include the time of receiving / sending and the user_ids of from whom received or to whom sent etc. Thus the Inlog and the Outlog of a reviewer keeps the record of flow of documents.
- **Context**: It is the dynamic context attached to the workflow. It is basically a document which contains a list of hyperlinks to the documents referenced: rules, precedents and support documents. It also contains

templates based on the predicates defined on the profiles of documents to enable the reviewer to search for documents to be included in the context, if required.

2.4.2 The Workflow Interface

Through this interface a user interacts with the workflow agent of the client logic module. The interaction may be different for different roles. A reviewer interacts to find out the status of an active case of the workflow. The status includes the reviewer with whom the case is lying and for how long, the time taken to complete a particular task etc. A work manager inherits a reviewer and additionally can create a new DPW and can modify an existing workflow in different aspects. For example, a manager can change the route of the DPW, can temporarily reassign a task to another office worker of the same role in case one in the review line is absent, can create and modify the context template and the default context as need arises. The main elements of this interface are:

- **DPW design**: Through this a work manager interacts with the workflow agent to create a new DPW or to modify an existing DPW.
- **Case status**: Through this a user interacts with the workflow agent to find out the status of a case.
- **Load balancing**: Through this a work manager interacts with the workflow agent to find out the load of an office worker. The load may be calculated based on the number of DPWs the user is participating as a reviewer, the number of cases flowing into and the number of cases flowing out from the desk of the user, average time of completion of the tasks for the cases handled etc.

2.4.3 The User Interface

This is a standard interface through which a user interacts with the user agent for mutual authentication of the user and the arbiter and also to set up a signed session key.

2.5 The Client Logic

This module is populated by client side processes. The processes in this module are termed as agents. Main agents are: *user agent, production agent, workflow agent* and *crypto*

2.5.1 Crypto

This component is responsible for all security activities in the client side. The responsibilities include: encryption and decryption of messages, verification of origin, confidentiality, content integrity of the messages.

2.5.2 Production Agent

This agent is responsible for client activities during production of digital documents. It interacts with the production manager of the arbiter using the production protocol, discussed later. The responsibilities of the production agent include:

- create original request documents using a template, submit it to the arbiter via *crypto.*
- receive the document sent by the arbiter and verified by the crypto and forward it to the view module for display.

- get the context of the DPW from the production manager,
- get the reference document selected out of the context by the user from the production manager,
- update the context by inclusion of reference items in the context during review
- submit the updated context, basically the new inclusions, to the reference manager
- draw citations from the context to the comments and thus form a part and submit it to the arbiter via crypto.
- receive the evidence of submission from the arbiter and update the inlog and outlog indices.

2.5.3 User Agent

This agent is responsible for client activities during user authentication and session setup. It interacts with the user manager of the arbiter logic using the authentication protocol described later. The responsibilities of the agent include:

- request the user manager for registration of a new user
- speak for the user during mutual authentication of the user and the arbiter
- collect session key(s) from the user manager during a login session
- return signed session key(s) to the user manager via crypto.

2.5.4 Workflow Agent

This agent is responsible for client activities regarding design and management of document production workflows. The responsibilities include:

- interact with the authorized user through the workflow interface to get the required parameters to create a new DPW or to modify an existing DPW. Modification includes reassignment of tasks, change of rules etc.
- get the request for workflow status from the user and display the status info received from the arbiter.

2.6 The Arbiter Logic

The arbiter is the central hub of of the architecture. It certifies subjects and objects, authenticates subjects and objects, manages storage and retrieval of objects, manages authorizations of subjects on objects, handles time-stamping etc. Since it is the arbiter, the resolutions of disputes based on evidences stored in the data storage of the arbiter will be final and binding. The services of the arbiter are provided by the following components called managers.

2.6.1 Crypto

It performs the same function as the crypto component of the client logic in the arbiter side.

2.6.2 Production Manager

This is an important component of the arbiter. The services provided by this manager include:

- forwarding a document to the next reviewer (client)
- addition of the part submitted by the client to the MPMSD under production,
- sending evidence of submission (NRS) of parts to the client
- recording document flow in the log books
- time-stamping the evidences of occurrences of events in the communication (sending, receiving, authoring etc.)
- authorization flow management (access control) during production
- rendering context and the reference documents to a client when requested for
- communication with other components

2.6.3 User Manager

This component provides the user management services. The services include:

- registration of a new user,
- de-registration of users (employees transferred, terminated, retired, suspended etc)
- generation of conjugate pair of private-public keys per user
- issuance of credentials (digital certificates) to users
- revocation of digital certificates

- maintenance of multi-version digital certificates
- generation and collection of signed session keys from users during a login session
- communication with other components

2.6.4 Storage Manager

The services of the component include:

- encryption of documents with a master key known only to the storage manager while storing in the data storage and decryption after retrieving.
- storing and retrieving objects from different storages of the data storage.
- migrating objects from one storage type to the other as soon as criteria are satisfied.
- maintenance of migration tables in the storages
- archiving and removal of reference objects
- deletion of archived objects after the expiry of the life of the objects

2.7 The Data Storage

This tier is basically a repository of office objects. There are three storage types - active, reference and archive.

2.7.1 Active Storage

Here, all the active objects are stored. Active objects are basically operational data. MPMSDs which are under production are the main objects stored here. Apart from MPMSDs, workflow documents, context documents, user credentials, log books, session keys, program codes, program credentials, migration tables are also active.

2.7.2 Reference Storage

Here all the reference documents are stored. An active MPMSD becomes a reference document as soon as it is closed and it is migrated to the reference storage. Other reference documents are: older versions of workflow documents, and program codes, rules, precedents, revoked user credentials, log book entries related to the flow of the closed MPMSDs and the signed session keys of closed MPMSDs that are not associated with any other active document. The objects stored here are read only to authorized users.

2.7.3 Archive

The reference objects which are not accessed for a long time are archived and are migrated to the archive storage. The objects stored in the archive cannot be read even. The objects are stored in compressed form.

2.8 The Protocols

There are three main protocols by which client agents interact with the corresponding managers in the arbiter. The protocols are:

- *Workflow Protocol:*This is a protocol used by the workflow agent and the workflow manager during interaction. The interaction is mainly during design, modification and maintenance of DPWs.
- *Production Protocol:*This is the major protocol. It is used by the production agent and the production manager during production of MPMSDs. The protocol will be discussed in detail in Chapter 4.
- *User-Arbiter Protocol:* This is a standard protocol used for authentication of entities. Here the user and the arbiter will be authenticated through this protocol. The protocol provides peer authentication of the user and the arbiter.

There are standard solution for workflow design. Even graphical workflow design tools are available. Example products are Lotus Notes, Office.IQ, Oracle Workflow, WorkMAN, Visual WorkFlow, FlowMark etc. In the present work we assume that standard workflow design tools , suitable for DPW are available. Therefore, no further discussion on the components of workflow design, like workflow interface, workflow agent, workflow manager and workflow protocol has been done in this work. There are standard solutions for user authentication, like Smart Card technologies. Moreover, in depth theoretical study in user authentication, including peer authentication and secure protocols, are done in [10, 21]. Therefore, we also assume existing user authentication components are sufficient for entity authentication. Only additional concept which needs to be added to peer authentication is the establishment of a signed session key, which is a prerequisite for production protocol to start. It is discussed in Chapter 4.

2.9 Discussion

In this chapter production and storage framework for DPW is presented. The frameworks are design frameworks. The objective of the frameworks is to provide a clear perimeter of the production and storage perspective of the DPW problem. A conceptual architecture is also provided in the present chapter. While the frameworks provide vertical perspectives, the architecture provides a horizontal broad perspective across all the vertical perspectives. The architecture is based on a central arbiter. The need for central arbitration comes from security perspective of the problem, which is the subject matter of Chapter 4.

Chapter 3

The Page Cube Model

In this chapter, we discuss a model for storage and retrieval of documents in an e-office during document production workflow with the context as the main binding element. This is a conceptual multi-dimensional model. The notion of a dimension provides a lot of semantic information, especially about the hierarchical relationship among its elements. 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. A page has a *profile*, which describes the page and is defined by a set of attribute-value pairs. *Registration* of a page means adding and recording a new page to the page cube and assigning a unique page identifier, *pid*, to the new page. PC has two components: *page space* and *page graphs.* A preliminary version of the model appeared in [38].

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 an attribute. An attribute may have attributes and these in turn may have further attributes. Thus, the 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 main dimensions include the following but are not limited to:

- **Time**: It is a hierarchical dimension. The hierarchy is *year.month.day.hour.min.sec.* The time dimension provides the time of creation of a page. It is a hierarchy with fixed depth.

- **Topic**: A page may be in one or more topics. A topic may have subtopics and a subtopic may be further classified. Thus it forms a topic hierarchy. Topic is a growing hierarchy.

- **Type**: A page may be of a type. A type may have subtypes and thus type forms a growing hierarchy. In time, a new type may be created under an existing leaf type. Generic to specific and to more specific types of pages in an office forms this hierarchy. For example, let the most general type of a page be *note.* A *note* may have subtypes *office order, report, comment* etc. Again, *office order* may have subtypes *appointment letter, termination letter, sanction order, circular, notice* etc. Similarly, *report* may have subtypes *inquiry, meeting minutes, field report* etc. and comment in turn may have subtypes *advice, remark, suggestion, ascent, descent* etc.

- **Category**: A page may belong to one of the three categories: *context, document* or *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 of the parts. The pages of category *documents* are the MPMSDs. A page of category *context* contains a set of links to pages of category *document.* A DPW will have a context page associated with it.

- **Class**: Pages may be classified based on the themes of the content of the pages. The classes are *rule, case, support, result-set* and *query.* 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. 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 the BoM meeting workflow, while at the same time, it is a rule of concerned topics. The result of a query will also be a page containing links to the pages satisfying the query and such a page will be of class *result-set.* Finally, the queries are also stored in the page cube as pages, therefore *query* is 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 and subdomains.

- **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 be in one of the four states: *active, reference, archived* or *burned.*

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.

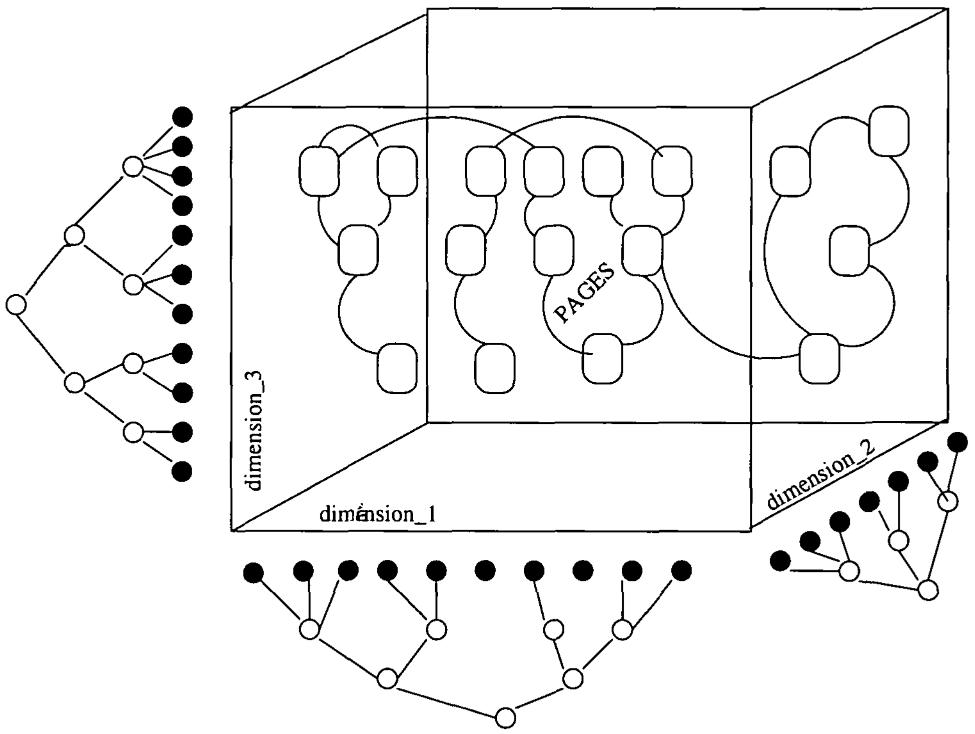

Figure 3.1: Page Cube

3.2 The Page Graphs

The pages of a PC are linked to a given page either implicitly or explicitly. Implicitly linked pages are those pages which satisfy a *predicate* defined over the dimensional values of the pages. 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 hyperlinks 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. Each dimension will have one DG. Thus the page graph component of PC is a set of DGs.

Let P be a PC, G the set of all graphs in P, V the set of all pages in P and E the set of all the directed edges of the graphs in G. In a dimension graph, pages are the vertices.
Let $G_d(V_d, E_d)$ be a digraph representing a dimension graph of dimension d, where $V_d \subset V$, $E \subset V$.
The vertices and the edges of $G_d(V_d, E_d)$ can be further classified into different kinds, based on the value of the dimension d.
Let T_d^V and T_d^E be the set of all different kinds of vertices and edges of $G_d(V_d, E_d)$. Then V_d^α is a set of vertices of dimension d and of kind α. Similarly E_d^β is the set of edges of graph $G_d(V_d, E_d)$, of kind β. Again the edges may be classified based on the direction *forward* or *reverse*.
Accordingly, $E_d^\beta = E_d^{\beta+} \cup E_d^{\beta-}$, $E_d^{\beta+} \subset E_d^\beta$ is a set of forward edges and $E_d^{\beta-} \subset E_d^\beta$ is a set of reverse edges. Moreover, the edges in E_d may be weighted, making $G_d(V_d, E_d)$ a *weighted graph.* The weight may be defined differently in different DGs. Linking of different kinds of pages by different kinds of edges in $G_d(V_d, E_d)$ is subject to the satisfaction of a set of constraints C_d. The act of linking two pages by creating a pair of conjugate edges is termed as *plugging.*

3.2.1 Pages

Pages are the main entities to deal with in a PC. Therefore, 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 PC. To define a page, we first define a few sets as follows:

Let S be a countably infinite set of strings,

$A \subset S$ be a set of variables called attributes,

$C \subset S$ be a set of allowed values for the attributes in A,

From the preceding discussion, we define an edge as follows:

Definition 3.1 *An edge $e \in E$ is a $6-tuple$, (source, target, graph, kind, direction, weight), where source is the source page, $target$ is the target page, graph $\in G$ is the concerned dimensional graph, kind is the kind of the edge, direction is the the direction of the edge: either $forward(+)$ or $backward(-)\}$ and $weight$ is the weght of the edge.*

With these definitions let us now define a page as follows:

Definition 3.2 *A page $p \in V$ 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, $L_+ \subset E$ is the set of forward edges, $L_- \subset E$ is the set of reverse edges, where for every edge $e \in L_+ \cup L_-$, $e.source = p$.*

$e.source = p$ in the definition 3.2 is a format of writing the *value* of the *source* attribute of tuple e is p

Definition 3.3 *A profile B is defined as a 2-tuple, an attribute and a list of values, $B = \{(a, \{v_1, v_2, \cdots, v_n\}) | a \in A, v_i \in C, i = 1, 2, 3, \cdots, n\}$*

A page is a point in the multi-dimensional page space of a PC. Its co-ordinates are defined in the profile of the page. The profile of a page contains the values of the attributes of the page. The attributes of a page may be dimensional and non-dimensional attributes. A dimensional attribute contains the values of the page for a dimension of the PC. A non-dimensional attribute contains the values other than the dimensional values. For example, $type, topic, category$ etc. are the dimensional, whereas *pageId*, *signature*, *size* etc. are the non-dimensional attributes of a page. The kind of a vertex (which is a page) of a DG is stored in a page as a dimensional attribute. Similarly graph, kind and direction of an edge of a DG are stored as attributes of an edge.

A page of a particular type may be on more than one 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 page. Otherwise, the n-tuple coordinate would have been an ideal logical address of a page. Without any loss of generality, we can use the time of creation of a page as the unique pageId, since in our scheme the timestamp will be given by a central arbiter [37].

3.2.2 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 the dimensional graphs for *category* is mandatory. Therefore, we shall discuss the category graphs 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* kind of edges. Moreover, a part itself may consist of multiple parts due to revision of the part from time to time, as discussed in detail in section 3.2.3. Therefore, a part may be linked by *splitPart* or *revisePart* kinds of edges. 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 *docPart* kind of edges. 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 *conDoc* kind of edges. It certain cases, a context may be plugged to another context by *conCon* kind of edges. For example, in addition to the documents specific to a case, the context of a case includes the context of the DPW. So, the context page of a case plugs the context page of the corresponding DPW.

Let $G_{category}(V_{category}, E_{category})$ be the dimensional graph for the dimension *category*. Let $T^V_{category} = \{part, document, context\}$ be the set of kinds of vertices, $T^E_{category} = \{cite, splitPart, revisePart, docPart, conDoc, conCon\}$ the set of kinds of edges,
$V_{category} = \{V_{part}, V_{document}, V_{context}\}$, where V_{part} is a set of vertices of kind *part*, $V_{document}$ is a set of vertices of kind *document* and $V_{context}$ is a set of vertices of kind *context*.
$E_{category} = \{E_{cite}, E_{splitPart}, E_{revisePart}, E_{docPart}, E_{conDoc}\}$, where E_{cite} is a set of edges of kind *cite*, $E_{splitPart}$ is a set of edges of kind *splitPart*, $E_{revisePart}$ is a set of edges of kind *revisePart*, $E_{docPart}$ is a set of edges of kind *docPart*, E_{conDoc} is a set of edges of kind *conDoc* and E_{conCon} is a set of edges of kind *conCon*.
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 2-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 an edge remains active from the moment it is plugged till it is unplugged, that is, during the period defined by t_+ and t_-. Therefore, edges in $G_{category}(V, E)$ are persistent in nature. Unplugging does not delete the conjugate pair of edges, it only modifies their weights. The temporal weights of edges keep the history of the plugging of pages and are used in the retrieval of a stage of a page at a particular time. It will be discussed later. Since there is a possibility of plugging as well as unplug-

ging more than once between the same pair of vertices, parallel edges with different weights may exist.

The graph $G_{category}(V_{category}, E_{category})$ is constructed by plugging the pages 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:
  (i) $p_i \in V_{part}$ and $p_j \in V_{part} \cup V_{document}$
  (ii) $p_i, p_j \in P$.
  The significance of the first restrictions is that a page of category *part* can cite another page of category either *part* or *document.* According to the second restriction, actual plugging takes place only after registration of the new page in the page cube.

- **DocumentPart Constraints**: A page p_j is plugged to page p_i iff the following conformability conditions are satisfied:
  (i) $p_i \in V_{document}$ and $p_j \in V_{part}$
  (ii) p_i is in active state.
  (iii) $p_i, p_j \in P$.
  A page of category *part* can be plugged to a page of category *document* in active state only. 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:
  (i) $p_i \in V_{context}$ and $p_j \in V_{document}$
  (ii) p_i is in active state
  (iii) $p_i, p_j \in P$

- **PartPart Constraints**: A page p_j is plugged to page p_i iff the following conformability conditions are satisfied:

  (i) $p_i, p_j \in V_{part}$

  (ii) p_i is in active state

  (iii) $p_i, p_j \in P$

- **ContextContext Constraints**: A page p_j is plugged to page p_i iff the following conformability conditions are satisfied:

  (i) $p_i, p_j \in V_{context}$

  (ii) p_i is in active state

  (iii) $p_i, p_j \in P$

3.2.3 Revision of a Page

Revision of a page is applicable only to the pages of category *part*. For certain classes of pages of category *part*, splitting of a part may be necessary. For example, let us consider a page of category *part* and class *rule*, containing rules on some topics. These rules may be revised from time to time. Due to revision only a certain portion of the content of the page may be changed in the revised version and the remaining portions of the content is intact. There are two ways to take care of such time varying parts. In the first approach, a new version of the part is created in a new page. This page is plugged to the document page and the page containing the old version is unplugged. The disadvantage of this approach is that for a minor revision in the content of the page, which is very common in an office, a major unrevised portion of the content is replicated in the new version of the page. In the second approach, an authorized user selects a page to be revised and marks a portion to be replaced by a new portion. The page is split into a number of portions and placed in separate newly created pages. One of these pages contains the revised portion, while the other pages (there can be 0 to 2 more such pages depending on whether the revised portion covers the entire original page, is in the middle of the page, or is at one end of the page), contain unrevised portions. All these pages are plugged to the original page and become its children, so to speak. The original page now has no content in it. Instead, its contents is to be recovered from the contents in its child pages based on the given time of recovery. A page of category *part* which has child pages may be considered to be a *compound part.* We shall refer to the parts in in these child pages as *portions* to ease the discussion. The algorithm to revise a part is given below as algorithm *RevisePart.* One aspect that needs to to be pointed out is that, as a result of this scheme, there is a need to distinguish between the time of creation of a page and the time of creation of the content of a page. The former is represented by the pageId, while the later is stored as the dimensional attribute *time* in the profile of a page. A newly created child page may have in it very old content.

ALGORITHM RevisePart(*pid, begin, end, new_portion*)

Algorithm to revise a page *p* of category *part*

INPUT :

pid : pageId of *p* to be revised

begin : beginning position of the portion in *p*

end : ending position of the portion in *p*

new_portion : content of new revised portion

OUTPUT :

status: 1 if successfully revised, 0 otherwise

ASSUMPTIONS :

status = 0 initially, when the algorithm is called

NOTATIONS :

t_p : time of creation of the content of p

null : a constant with value 0

now is also a timestamp, which represents the present time

timeStamp() : a function which returns the present time

endOfPage : end position of a page, taking beginning position as 0

```
BEGIN

Begin Case
step 1: case 1: $begin = 0$ and $end = endOfPage$
                // full page is to be replaced
step 1.2:   Register a new page with pageId $pid_{old}$ and category $part$
            Copy the content of $pid$ to $pid_{old}$;
            Set $X$ of $pid$ to $null$ //content of $pid$ is removed
            Set the time of creation of content of $pid_{old}$ to $t_p$
step 1.3:   Register a new page of category $part$ with pageId $pid_{new}$
            containing $new\_portion$
step 1.4:   Set $now$ = timeStamp();
            Plug $pid$ to $pid_{old}$ with a pair of edges of kind $revisePart$,
            where $plugtime = t_p$ and then unplug it with $unplugtime = now$;
            Plug $pid$ to $pid_{new}$ with a pair of edges of kind $revisePart$,
            where $plugtime = now$ and $unplugtime = null$;
            Set the time of creation of content of $pid_{new} = now$
step 1.5:   return($status = 1$)
step 2: case 2: $begin = 0$ and $end < endOfPage$
                //top portion of the page is to be replaced
```

```
step 2.1:  Split $pid$ into two portions $p_1$ and $p_2$ at position $end$
           Register $p_1$ and $p_2$, as new pages of category $part$
           with pageIds $pid_1$ and $pid_2$ respectively
step 2.2   Copy the portion from $begin$ to $end$ to $pid_1$ and
               from $end + 1$ to $endOfPage$ to $pid_2$
           Set $X$ of $pid$ to $null$ //content of $pid$ is removed
           Set the time of creation of content of $pid_1$ and $pid_2$ to $t_p$
step 2.3:  Plug $pid$ to $pid_1$ with a pair of edges of kind $splitPart$,
           where $plugtime = t_p$ and $unplugtime = null$
step 2.4:  Plug $pid$ to $pid_2$ with a pair of edges of kind $splitPart$,
           where $plugtime = t_p$ and $unplugtime = null$
step 2.5:  RevisePart($pid_1, begin = 0, end = endOfPage, new\_portion$)
step 3: case 3: $begin > 0$ and $end = endOfPage$
               //bottom portion of the page is to be replaced
step 3.1:  split $pid$ into two portions $p_1$ and $p_2$ at position $begin$
           Register $p_1$ and $p_2$, as new pages of category $part$
           with pageIds $pid_1$ and $pid_2$ respectively
step 3.2   Copy the portion from 0 to $begin - 1$ to $pid_1$ and
               from $begin$ to $endOfPage$ to $pid_2$
           Set $X$ of $pid$ to $null$ //content of $pid$ is removed
           Set the time of creation of content of $pid_1$ and $pid_2$ to $t_p$
step 3.3:  Plug $pid$ to $pid_1$ with a pair of edges of kind $splitPart$,
           where $plugtime = t_p$ and $unplugtime = null$
step 3.4:  Plug $pid$ to $pid_2$ with a pair of edges of kind $splitPart$,
           where $plugtime = t_p$ and $unplugtime = null$
step 3.5:  RevisePart($pid_2, begin = 0, end = endOfPage, new\_portion$)
step 4: case 4: $begin > 0$ and $end < endOfPage$
```

```
            //middle portion of the page is to be replaced
step 4.1    Split $pid$ into three portions $p_1$, $p_2$ and $p_3$ at positions $begin$ and $end$
            Register $p_1$, $p_2$ and $p_3$, as new pages of category $part$
            with pageIds $pid_1$, $pid_2$ and $pid_3$ respectively
step 4.2    Copy the portion from 0 to $begin - 1$ to $pid_1$,
                from $begin$ to $end$ to $pid_2$ and from $end + 1$ to $endOfPage$ to $pid_3$
            Set $X$ of $pid$ to $null$ //content of $pid$ is removed
            Set the time of creation of content of $pid_1$, $pid_2$ and $pid_3$ to $t_p$
step 4.3    Plug $pid$ to $pid_1$ with a pair of edges of kind $splitPart$,
            where $plugtime = t_p$ and $unplugtime = null$
step 4.4    Plug $pid$ to $pid_2$ with a pair of edges of kind $splitPart$,
            where $plugtime = t_p$ and $unplugtime = null$
step 4.5    Plug $pid$ to $pid_3$ with a pair of edges of kind $splitPart$,
            where $plugtime = t_p$ and $unplugtime = null$
step 4.6    RevisePart($pid_2, begin = 0, end = endOfPage, new\_portion$)
End Case

END
```

All the four possible cases of revision of a part are taken care of in the algorithm *RevisePart*(). In case 1 (step 1), entire part is revised by a new part. In case 2 (step 2), case 3 (step 3) and case 4 (step 4) revision of top, bottom and middle portions of a part are taken care of. A page is created first, and then may be plugged to one or more pages. Therefore the time of creation of a page is always less than or equal to the time of plugging the page to other pages. When a page is split to multiple portions, the portions of a page collectively contain the content of the split page. Therefore, the time of creation of the contents of the portions are equal to the time of creation of the original part. When a state of the page at a particular time is retrieved, the portions of the page not revised are to be included intact. The portions are to be retrieved with respect to the time of plugging and unplugging. Therefore, if the plug time of the portions, that are unrevised and the old versions of the revised portions, are set to the t_p, the time of creation of the content of the part to be revised, then the retrieval will be correct. The retrieval of a state of a page at a particular time is discussed in the next section.

3.2.4 Retrieval of a Page

Let us first discuss how to retrieve a page of category *part*, which may have multiple portions, and some portions may be revised. We discuss below an algorithm *GetPart*() to retrieve the state of a page of category *part* at time t. The algorithm returns a list of pageIds, and the concatenation of the contents of these pageIds gives the state of the part at time t. Since a portion is registered as a page with a unique pageId and the portions are registered in order, therefore pageIds of children nodes of a parent node gives the order. A portion may have further portions as children. A part may have three types of edges: *cite, splitPart* and *revisePart.* A leaf node is identified as a node having no *splitpart* as well as *revisePart* kinds of edges. An internal node will have either *splitPart* or *revisepart* kind of edges. The leaf nodes which satisfy the temporal conditions on time t, given in the algorithm *GetPart()*, constitute the state of a part at time t from the tree. When $t = now$, we get the current state of a part.

ALGORITHM GetPart(pid, t)

Algorithm to retrieve the state p^t of a page p of category *part*, at time t

INPUT :

pid : pageId of part p to be retrieved

t : time which defines the state of *pid*

OUTPUT :

$pidList^t$: list of pageids, in order, which constitute the part *pid* at time t

ASSUMPTIONS :

$pidList^t$ is intially *null*

NOTATIONS :

$pid_1 || pid_2$: concatenation of pid_1 and pid_2

null : a constant with value 0

now is a timestamp, which signifies the present time

element.attribute = *value* : the structure is a short form of writing the value of the attribute of an element

BEGIN

```
step 1:   Retrieve page $p$ with pageId = $pid$
step 2:   From $p$, form a set of forward edges of kind $splitPart$
              $L^s_+ = \{e^s | e^s.source = pid$ and $e^s.kind = splitPart$
              and $e^s.dir = +$ and $e^s.t_+ <= t$ and $(e^s.t_- > t$ or $e^s.t_- = null)\}$
step 2.1: Form $pidList^s$ of pageIds of pages of category $part$ from $L^s_+$
              $pidList^s = \{p_s | \ e^s.target = p_s$, where $e^s \in L^s_+\}$
step 2.2: if $pidList^s$ is not $null$ then sort pageIds in $pidList^s$
              in ascending order of pageIds
step 3:       From $p$, form a set of forward edges of kind $revisePart$
              $L^r_+ = \{e^r | e^r.source = pid$ and $e^r.kind = revisePart$
              and $e^r.dir = +$ and $e^r.t_+ <= t$ and $(e^r.t_- > t$ or $e^r.t_- = null)\}$
step 3.1:     Form $pidList^r$ of pageIds pages of category $part$ from $L^r_+$
              $pidList^r = \{p_r | \ e^r.target = p_r$ and $e^r \in L^r_+\}$
step 3.2:     if $pidList^r$ is not $null$ then sort pageIds in $pidList^r$
              in ascending order of pageIds
```

```
step 4:    if $pidList^s = null$ and $pidList^r = null$ then
               $pidList^t = pidlist^t || pid$
step 4.1:  else if $pidList^s \neq null$ then
           for each $p_s \in pidList^s$
               $pidList^t = pidList^t ||$ GetPart$(p_s, t)$
step 4.2:  else if $pidList^r \neq null$ then
    ·      for each $p_r \in pidList^r$
               $pidList^t = pidList^t ||$ GetPart$(p_r, t)$
           endif
step 5 :   return$(pidList^t)$
```

END

The algorithm *GetPart*() gives the state of a part *pid* at time t. Once we take care of pages of category *part* retrieval of a page of category *document* is a simple one, similar to *GetPart*(). The algorithm *GetDocument*() is given below:

ALGORITHM GetDocument(pid, t)

Algorithm to retrieve the state p^t of a page p of category *document*, at time t

INPUT :

pid : pageId of document p to be retrieved

t : time which defines the state of *pid*

OUTPUT :

$pidList^t$: list of pageids, in order, which constitute the document *pid* at time t

ASSUMPTIONS :

$pidList^t$ is intially *null*

NOTATIONS :

$pid_1 || pid_2$: concatenation of pid_1 and pid_2

null : a constant with value 0

now is also a timestamp, which signifies the present time

element.attribute = *value* : the structure is a short form of writing the value of the attribute of an element

```
BEGIN

step 1: Retrieve page $p$ with pageId = $pid$
step 2: From $p$, form a set of forward edges of kind $docPart$
            $L^s_+ = \{e^s | e^s.source = pid$ and $e^s.kind = dosPart$
            and $e^s.dir = +$ and $e^s.t_+ <= t$ and $(e^s.t_- > t$ or $e^s.t_- = null) \}$
step 2.1: Form $pidList^s$ of pageIds of pages of category $part$ from $L^s_+$
            $pidList^s = \{p_s | \ e^s.target = p_s$, where $e^s \in L^s_+\}$
step 2.2: if $pidList^s \neq null$ then
            sort pageIds in $pidList^s$ in ascending order of pageIds
            for each $p_s \in pidList^s$
                $pidList^t = pidList^t ||$ GetPart$(p_s, t)$
            endif
step 3 :    return($pidList^t$)

END
```

Retrieval of a page of category *context* is similar to *GetDocument*(). Since, production of context is the subject matter of Chapter 4, it is discussed in detail there.

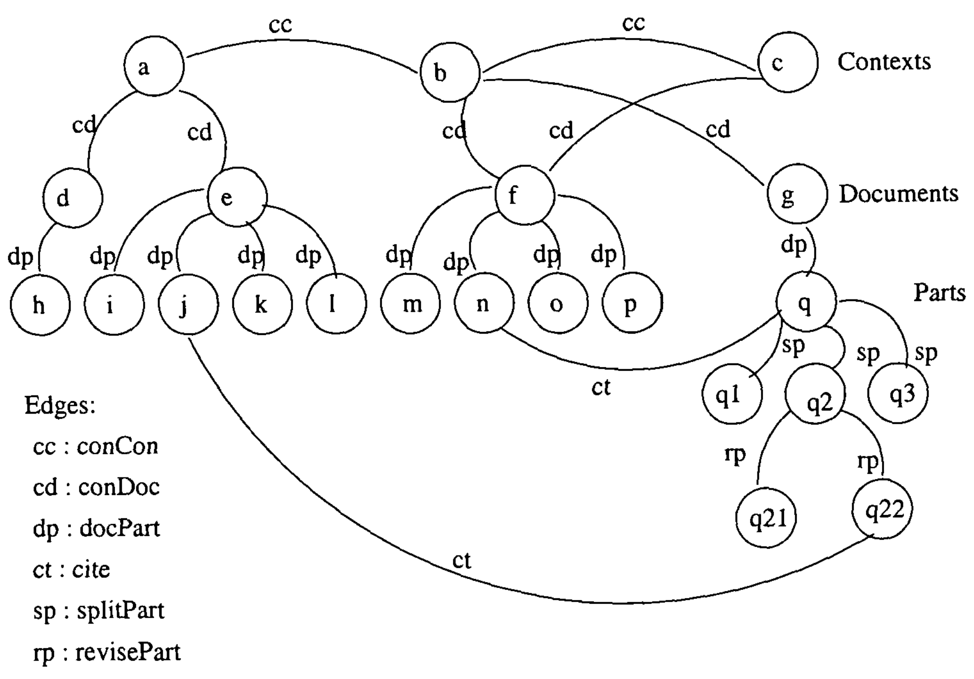

Figure 3.2: Portion of Category Graph for Travel Plan Workflow

3.2.5 An Example

Let us consider again the travel plan workflow discussed in Chapter 1 and 2. Portion of a category graph for such a workflow is shown in figure 3.2. The node e is the document that is under examination. This document has four parts, i, j, k and l corresponding to employee A's application, reviewers B, C and D's comments respectively. The document e has a case context and this is node a. Node a points to document d, which is being used by A as it is her leave account. The context a also points to the DPW context b. The context b includes document f, which is an earlier case acting as a precedent for this DPW and it has a case context c. The context b also includes document g, which is the set of leave rules. This rules were originally in node q, but a portion got revised. As a result, nodes $q1$, $q2$ and $q3$ were created. $q1$ and $q2$ contains the first and the last part of the rules while $q2$ points to $q21$ and

$q22$. $q21$ contains the pre-revised portion and $q22$ the revised portion of the rules. $q21$ was unplugged when $q22$ was plugged. The rules corrently are therefore contained in $q1, q22$ and $q3$. A link emanating from node $\imath$ to $q22$ is a citation of the rules in $q22$ by applicant A. There is another citation link from node n to q shown. Here the entire set of rules are being cited.

3.3 Query Languages

A query is an expression denoting a set of pages described by a formula ϕ of the form $\{p|\phi(p)\}$. A query language provides a user with a means of expressing questions in the form that can be handled by the model enabling the model to answer 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 languages are similar to Relational Algebra and SQL respectively.

3.3.1 Path Expression

A Path Expression(PE) defines a path from one node in the graph to another in terms of intermediate node and edge labels. PEs in graphs are used in navigation oriented queries. 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. Details of use of path expressions in document databases is given in [14]. We follow the simplified PE discussed in [35]. The standard wildcat

character "*" is used in a path expression to signify all successors of a node in the dimension hierarchy as well as in the dimensional graph. The standard "." operator, commonly used to denote attributes of a relation in the relational model can now be cascaded to express a listed path. In addition a ".." operator is introduced, which is used to construct an abbreviated path from a listed path. For example, a fully listed path expression is $p_1.p_2.p_3.p_4.p_6.p_7$, where $p_i, i = 1, 7$ is a pageId. An abbreviated path $p_1..p_7$ means that there is a path between p_1 and p_7, but the actual path itself is not of significance. Formally, the above expression evaluates to :
$\exists x PATH(x) \cap p_1.x.p_7$
".." and "*" takes care of the don't care conditions in a PE. Details of PEs is available in[35].

3.3.2 Page Algebra

Page Algebra(PA) is an operator based query language for querying pages from a PC. It is an extension of relational algebra. PA is defined in terms of a special set operators that map one or more sets of pages to a new set of pages. 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 edges to the pages belonging to E_2 and the pages belonging to E_2 with backward edges 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 forward (+) and backward (-) edges. For example, for the category graph, with graph id g has context to document(cd), document to part(dp), part to part(pp) kinds of edges. Accordingly, the forward and backward edges may be qualified. If E_1 contains pages of category document and E_2 contains parts, then the qualified edges due to $E_1 \times E_2$ will be

$p_1 = (p_1, \{(p_3, g, (t_+,), dp, +), (p_4, g, (t_+,), dp, +)\})$
$p_2 = (p_2, \{(p_3, g, (t_+,), dp, +), (p_4, g, (t_+,), dp, +)\})$
$p_3 = (p_3, \{(p_1, g, (t_+,), dp, -), (p_2, g, (t_+,), dp, -)\})$
$p_4 = (p_4, \{(p_1, g, (t_+,), dp, -), (p_2, g, (t_+,), dp, -)\})$

- **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 deactivating forward edges to the pages belonging to E_2 and also reproduces the pages belonging to E_2 deactivating the backward edges to pages belonging to E_1. Unplug does not necessarily remove the edges physically. It simply modifies the weights of the edges by incorporating the time of unplugging (t_-). After unplugging the pages of E_1 and E_2 change to the following:
  $p_1 = (p_1, \{(p_3, g, (t_+, t_-), dp, +), (p_4, g, (t_+, t_-), dp, +)\})$
  $p_2 = (p_2, \{(p_3, g, (t_+, t_-), dp, +), (p_4, g, (t_+, t_-), dp, +)\})$
  $p_3 = (p_3, \{(p_1, g, (t_+, t_-), dp, -), (p_2, g, (t_+, t_-), dp, -)\})$
  $p_4 = (p_4, \{(p_1, g, (t_+, t_-), dp, -), (p_2, g, (t_+, t_-), dp, -)\})$

- **Path Selection** ($\circ$): This is basically a navigational operation, where $\circ \in \{.,..\}$. Given a PA expression E and a PE P, $E \circ P$ returns the set of pages obtained after traversing the path P from each of the pages in E [35].

- **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

  $E_1 - E_2 = \{p | p \in E_1 \wedge p \notin E_2\}$

Examples

Let us look at some examples to illustrate the different operators of PA.

1. 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

   $\sigma_{category="context" \wedge dpw="w"}(P)$

$\times$

$\sigma_{category="document" \wedge class="rules" \wedge topic="leave.casual.special"}(P)$

2. Find from the page cube P, all the documents containing rules on special casual leave and exclude them from the context of the dpw w.

   $\sigma_{category="context" \wedge dpw="w"}(P)$

   $\div$

   $\sigma_{category="document" \wedge class="rules" \wedge topic="leave.casual.special"}(P)$

3. Find from page cube P, all the parts included in the context of the dpw w and signed by user u.

   $(\sigma_{category="context" \wedge dpw="w"}(P)) \circ (\sigma_{category="part" \wedge user="u"}(P))$

   This expression returns all parts signed by u if there exists a path from the context of w to the part.

4. Find from the page cube P all office orders on leave which are neither sanction orders nor the circulars.

   $(\sigma_{type="..office\_order.*" \wedge topic="..leave.*"}(P))$

   $-$

   $(\sigma_{type="..office\_order.sanction.*" \wedge topic="..leave.*"}(P)$

   $\cup$

   $\sigma_{type="..office\_order.notice.*" \wedge topic="..leave.*"}(P))$

3.3.3 Page Structured Query Language

Here we present a language, called PSQL, for interactively processing queries on pages . PSQL is an extended version of SQL. The primary motivation behind such a language is to provide users of database 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 accommodate 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 hyperlinks(edges) to the constituent pages.

Examples

Let us consider some queries expressed in PSQL.

1. Find all the precedents of the workflow w from the page cube P.

```
SELECT page q
FROM   cube P
WHERE q.profile.dimension.category = document
 and  q.profile.dimension.class = case
 and  q.profile.dimension.dpw = w
 and  q.profile.dimension.state = reference
```

2. Find all the parts of the precedents of the workflow w from the page cube P.

```
SELECT page q
FROM  cube P, P.graph.category g
WHERE q.profile.category = part
```

```
and  g.dpw = w
and  g.edge.target= q
and  g.edge.kind = docpart
and  g.edge.dir = +
and  g.edge.source IN
  (SELECT page p
   FROM  cube P
   WHERE  p.profile.dimension.category=document
    and  p.profile.dimension.class=case
    and  p.profile.dimension.dpw=w
    and  p.profile.dimension.state=reference)
```

3. Find all documents from a page cube P containing rules on casual leave or on travel abroad produced after August, 1990.

```
SELECT page q
FROM cube P
WHERE q.profile.dimension.category = document
  and q.profile.dimension.class = rules
  and q.profile.dimension.topic
                IN {leave.casual.*, travel..abroad.*}
  and q.profile.dimension.time BETWEEN {1990.08.*, now}
ORDER BY dimension.topic, dimension.time
```

The query produces a resultant page from a page cube P containing links to all the pages containing rules on casual leave, or on travel.abroad, or on any subtopics, produced between August 1990 and *now*. The value of the special variable *now* is defined by the arbiter with the current time stamp. The values included in the IN list are generally values of dimensional attributes of cube P. 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.

4. Find all documents from a page cube P containing rules on special casual leave and include them in the context of the workflow w.

```
(SELECT page q
 FROM cube P
 WHERE q.profile.dimension.category = context
  and  q.profile.dimension.dpw = w)
 PLUG (SELECT p
       FROM P
       WHERE p.profile.dimension.category = document
        and p.profile.dimension.class = rule
        and p.profile.dimension.topic = leave.casual.special )
```

5. Find all parts of the dpw w from a page cube P where the page with pageId "2000.08.12.13.25.31" is cited.

```
SELECT page p
FROM cube P, P.graph.category g
WHERE p.category = part
 and g.dpw = w
and g.edge.source = 2000.08.12.13.25.31
and g.edge.target = p
and g.edge.kind = cite
and g.edge.dir = -
```

3.3.4 PSQL Grammar

In this section the core of the Context-Free Grammar of PSQL in BNF notations is given. Since PSQL is a an extension of SQL, therefore, some of the productions are from standard SQL, some are from Active Rule [41]. Some productions are similar to DSQL [35] and Lorel[31]. Others are specific to PSQL.

```
<query-exp>     ::= <query_term> <query-exp> | <query_term>
<query_term>    ::= <rule_spec> | <query_spec>
<rule_spec>     ::= WHEN <event> [IF <cond_exp> THEN] <query_spec>
<event>         ::= request | response
<query_spec>    ::= SELECT [ALL | DISTINCT] <query_body>
                    [{PLUG | UNPLUG | UNION | INTERSECTION |
                      DIFFERENCE} <query_spec>]
<if_clause>     ::= IF <cond_exp> THEN {<if_clause>|<query_spec>}
<query_body>    ::= <from_clause> [<where_clasuse>] [<order_clause>]
<order_clause> ::= ORDER BY (path_list)
<from_clause>   ::= FROM <path_exp>  <identifier>]
                                  [, <path_exp>  <identifier>]*
<where_clause>  ::= WHERE <cond_exp>
<cond_exp>      ::= <cond_term> | <cond_exp> OR <cond_term>
<cond_term>     ::= <cond_factor> | <cond_term> AND <cond_factor>
<cond_factor>   ::= [NOT] <cond_test>
```

```
<cond_test>       ::= <cond_primary> [ IS [ NOT ] { TRUE | FALSE}]
<cond_primary>    ::= <simple_cond> | (cond_exp)
<simple_cond>     ::=  <comp_cond> | <between_cond> | <in_cond>
<comp_cond>       ::= <path_exp> = <path_exp>
<between_cond>    ::= <path_exp> [NOT] BETWEEN (<range_exp>)
<in-cond>         ::= <path_exp> [NOT] IN (<path_list>)
<range_exp>       ::= <path_exp> - <path_exp>
<path_exp>        ::= <path>[.|.. <path>]*
<path_list>       ::= <path>[,<path>]*
<path>            ::= <identifier>[.<identifier>]*
<identifier>      ::= <identifier><character>|<character>
<character>       ::= <letter>|<digit>| |_|-|+|<|>|=|!|*|.
<letter>          ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|
                      U|V|W|X|Y|Z
                      |a|b|c|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|
                      s|t|u|v|w|x|y|z
<digit>           ::= 0|1|2|3|4|5|6|7|8|9
```

3.4 Closure

Page Cube is a closed model. Simply put, this means that input to a query are pages belonging to PC and the result-set, that is, the output of a query, is also a page having hyperlinks to the pages satisfying the query. Closure is prominent in a relational database model, as tables are both input to queries and output from queries. In addition using the QBE query language, queries are formulated also using tables. Similarly, in page cube, queries can also be pages. The advantages of a closed model are discussed in [35]

3.5 Discussion

In this Chapter a model, called Page Cube, for storage and retrieval of documents in an office environment is presented. The model is a multi-dimensional model. Multi-dimensional modelling is a new paradigm of modelling data. Data Cube model of OLAP is the most common application of multi-dimensional modelling that influenced the data warehouse architecture. In data cube, numeric measures, like sales, inventory, population etc., are the main subject of the analysis. Unlike ER modelling, which describes entities and relationships, dimension modelling deals with numeric measures and the dimensions. It is for the first time, in the Page Cube model, dimension modelling is extended in modelling office documents. The Page Cube deals with dimensions and pages and their inter-connectivity. The genesis of Page Cube model is the formation of page graphs with temporal labelling scheme of the edges. Query languages like Relational Algebra and SQL are very rich in constructs. But for querying pages from a Page Cube only a small and simple set of constructs are necessary. Moreover, some constructs are specific to Page Cube. Therefore, two query languages, in the line of RA and SQL, namely PA and PSQL are proposed to query Page Cubes. The model can be extended by incorporating document to document linking by corresponding kinds of edges if such a need arises in a particular office.

Chapter 4

Security

An office document handling system requires security during production, storage and transport of documents. Therefore, security is an important aspect of the DPW problem. In the present chapter we discuss the security of DPW under a security framework and we propose a secure protocol for production of cases of a DPW.

4.1 The Security Framework

The security of a DPW can be studied in a framework formed by the following standard fundamental security concepts: *authentication, access control, authorization, authorization flow, integrity, confidentiality* and *non-repudiation.* All these concepts are well known. We are giving a review of these basic concepts in this framework.

Only a set of legitimate users, called the office workers are allowed to handle documents in a particular office. Therefore, authentication of users is a very important security requirement in an office system.

Definition 4.1 *The authentication process is the aggregation of two pro-*

cesses: identification and verification. Identification is the process whereby an entity claims a certain identity, while verification is the process whereby the claim is checked. When two communicating entities authenticate each other simultaneously it is referred to as mutual or peer authentication.

There are three primary methods of authentication of an entity:

K. *authentication by knowledge*: An entity is authenticated by the verification of some secret knowledge, for example, authentication by the knowledge of a password.

O. *authentication by ownership*: An entity is authenticated by the possession of some devices. Possession of an identity card is an example of this method.

C. *authentication by characteristic*: An entity can be authenticated by the characteristics unique to the entity. Characteristics like finger print, retinal pattern, DNA patterns etc in case of human being and message digests in case of digital messages can be used as characteristics for authentication.

Let $A = \{K, O, C\}$ be the set of primary methods of authentication. All the authentication protocols availabble today use some combinations of these three methods. The set of all possible combination is the power set of S excluding the null set. The traditional password based method, $\{K\} \subset A$, is vulnerable because it is easy to grab the password from the network [39]. A substantially better method is the smart card method, which is used in electronic commerce etc.. Basically it uses the ownership of a card which contains a secret-key, which in turn is always encrypted by a password and stored in a chip of the card along with other information. It is the combination $\{K, O\} \subset A$. This method is also not perfect, because loss of the card and the leakage of the password may lead to forgery. Perfect authentication is an open problem and if a solution for that evolves, it will obviously be of the combination $\{K, O, C\} \subseteq A$.

Access control and authorization are generally defined in terms of subjects and objects. Subjects are sometimes called the principals.

Definition 4.2 *A* **subject** *is an active entity of the system, which accesses objects. These accesses must be controlled to ensure that they match the security requirements.*

Definition 4.3 *An* **object** *is a passive entity of the system, which contain information to be protected from unauthorized accesses.*

If a user is properly authenticated in an office, it does not necessarily mean that the user can access all the objects in the office. Some types of objects may be accessible to only office workers of a certain role. Moreover, a worker of a role may not be authorized to access a particular object even though the role(s) he belongs to may have authorization to access that particular type of objects. How to ensure that access to information resources and hardware resources is available only to the authorized user is the subject matter of access control.

Definition 4.4 *Access control is the act of determining whether a subject is allowed to perform a given operation on an object.*

Authorization is typically determined based on three access control policies: Mandatory Access Control (MAC), Discretionary Access Control (DAC) and Role-Based Access Control(RBAC).

- **MAC** : Under MAC, a security level is assigned to each subject reflecting the degree to which it is trusted to not disclose sensitive information and to each object in accordance with the sensitivity of the information that it contains. The set of security levels is partially ordered in a lattice-structured hierarchy so that each level dominates itself and the ones below it. If finer granularity is required, a set of categories can also be assigned to each subject specifying the areas in which it operates and to each object describing the areas to which the contained information pertains. Detail of MAC is available in [32].

- **DAC** : The main idea behind DAC is that of enforcing the rules specified by an access matrix describing the modes in which each subject is allowed to access each object. Since the access matrix is usually sparse, typical implementations adopt either a column-wise representation called Access Control Lists(ACL) or a row-wise representation called Capability Lists(CL). Detail of DAC is available in [32].

- **RBAC** : In RBAC, accesses by a user to an object is determined on the basis of the role(s) that the user plays at the time of access. A role is a representation of a set of responsibilities associated with a particular activity. There are two kinds of authorizations in RBAC:

  - The ACL for each object indicates the modes of access permitted for each role.
  - Each user is authorized to adopt a certain predetermined set of roles.

  If a user's responsibility changes then new roles can be assigned. It is not necessary to go through the ACLs of all the objects one by one to remove entries pertaining to the user in her previous capacity and add entries pertaining to the new role(s).

Definition 4.5 Authorization *is defined as a 4-tuple* $A = \{s, o, pr, [t_+, t_-]\}$, *where subject* s *is granted access on object* o *with privilege* pr *at time* t_+ *and is revoked at time* t_-.

The time interval $[t_+, t_-]$ signifies that the subject s has privilege pr to access object o only during this time interval. In a workflow the authorization is non-monotonic in nature. A suitable authorization model for workflows must ensure that authorization is granted only when the task starts and revoked as soon as the task finishes. Otherwise a subject may possess authorization for time periods longer that required, which may compromise security. Therefore, authorization flow need to be tightly coupled to the workflow in order to ensure subjects possess authorization only when required.

Apart from authorization, the system must ensure to the users that the content of an object has not been tampered during transmission or while being stored. When an office object is stored or transported over a network from one point to the other, the confidentiality of the information contained in the object may need to be maintained. That means the information contained in the object be accessible only to the authorized subject. The type of access includes printing, displaying and revealing the existence of an object. In common practice, confidentiality of digital information is maintained by cryptographic encryption.

Definition 4.6 *Confidentiality of an object is concerned with the protection of disclosure of information to unauthorized users.*

In the production process of a document three main events are involved *authoring* a document, *sending* a document to the next reviewer, and *receiving* a document from the previous reviewer of the document. These events have binary possibilities: whether an entity has authored or has not authored a document, whether a document has been sent or has not been sent, whether a document has been received or has not been received, whether a document has been sent / received at a given time or has not been sent /received at that time. If two possibilities of an event cannot be distinguished, a party related to the event could make the denials.

Definition 4.7 *Repudiation is defined as denial by one of the entities involved in a communication of having participated in all or part of the communication. Non-repudiation is concerned with preventing such a denial*

Digital signatures address the non-repudiation of authorship of a document, whereas non-repudiation of sending and receiving are addressed by non-repudiation protocols using evidence of sending (NRS) and of receiving (NRR) with a certified time stamp. It is shown in [44] that to address the non-repudiation of sending or of receiving a document at a given time, involvement of an *trusted third party (TTP)* is mandatory. The TTP must be in-line. In the protocol for production of MPMSD also an in-line is mandatory. This will be discussed in 4.2.

Definition 4.8 *A* **trusted third party (TTP)** *is a security authority or its agent, trusted by other entities with respect to security related activities [22].*

From a communication viewpoint, three categories of TTPs can be distinguished based on the relative location to, and interaction with, the communicating parties. The three categories are in-line, on-line and off-line.

Definition 4.9 *A TTP is in-line if it is the intermediary, serving as the real-time means of communication between two entities A and B.*

Definition 4.10 *A TTP is on-line if it is involved in real-time during each protocol instance (communicating with A or B or both), but A and B communicate directly rather than through the TTP.*

Definition 4.11 *An TTP is off-line if it is not involved in the protocol in real-time, but prepares information a priori, which is available to A or B or both is and used during protocol execution.*

Definition 4.12 *A security policy is a set of rules that define the security* **subjects**, *security* **objects** *and relationship among them. Security policies define the principles on which access is granted or denied (without limiting legitimate access).*

The definition of security policies lead to the explicit formulation of security strategies, thus giving security its rightful relevance instead of the fragmentary and approximate consideration it is often given [32]. In the perspective of security we can consider an office as a single trust domain.

Definition 4.13 *A trust domain is a logical administrative structure within which a single, consistent security policy holds. Put another way a trust domain is a collection of both subjects and objects governed by single administration and a single security policy.*

The scope of our discussion is limited to a single trust domain. Inter domain security is beyond the scope of this discussion.

There are standard user authentication schemes, like Smart cards. During run-time a process represents the user. Process authentication is discussed in [21]. Standard digital signature schemes address the fundamental issues, like proof of origin, content integrity (sec 1.8.2). The issue of confidentiality can also be addressed by standard encryption schemes. There are standard non-repudiation protocols [1, 2, 3], which address the repudiation issues. But for persistent documents, the issue of signature replacement and the content integrity issue, where even the original author of a part of a MPMSD is not allowed to modify the content after it is dispatched to the next reviewer, are not addressed by standard digital signature schemes.

4.2 A Protocol for Secure Production of Cases

Based on the security framework discussed, we propose in the present section a protocol for secure production of MPMSDs, which are the cases of a DPW. The protocol addresses the issues particular to MPMSD. The protocol is based on a central arbitration mechanism. A preliminary version of the protocol appeared in [37].

4.2.1 The Central Arbiter

To make a protocol as general as possible, researchers attempt to avoid the use of an arbiter. However, to provide non-repudiation with time information, an in-line TTP is necessary. To provide non-repudiation of a digital signature, time of the signature is essential as the key used in the signature may become public at a later time. In our environment, documents are persistent and so non-repudiation of a digital signature is essential. So an in-line TTP will be required. Further, to prevent the reuse of parts (Sec. 1.8.3, item 2), an in-line TTP will evidently be required as immediate detection of such reuse will be necessary to prevent the taking of wrong decisions. When B and D collude to bypass C, C will not be aware of this till much later if no in-line TTP is present. Since an in-line TTP is mandatory for addressing issues like repudiation of sending and receiving documents as evident from [44], we can address other issues on MPMSD as well, with an arbiter as an in-line TTP. Therefore, the production of cases in the DPW system is based on a central arbitration mechanism. It is basically a client/server computing paradigm, where the arbiter is the server. All the cases flowing from one client(reviewer) to another are routed through the arbiter. In case of dispute, the decision of the arbiter, based of the stored evidences, is final and binding.

A light weight TTP is usually favoured as this reduces the overhead of communication. However, in our case, the TTP has to perform more tasks. With our assumption of an organization as a single trust domain, within which the documents flow, an 'in-house' TTP can be implemented and this can therefore be made heavyweight.

4.2.2 Signed Session Keys

In most of the office solutions, public-key based digital signatures are used. Commonly, the RSA [30] algorithm is used for encryption. The computational complexity of the algorithm is high. Therefore, digital signatures based on RSA are slow. In an e-office a reviewer may have to sign many documents per day. This may lead to performance degradation of an e-office system. In our scheme, we have a novel idea of using the signed session keys for the digital signatures of the reviewers on a MPMSD. Actually this idea makes our protocols efficient. During a session, each reviewer has a session key, and a signed copy of the same, signed by the reviewer and certified and time-stamped by the arbiter, is available to only the reviewer and the arbiter. Now since the session key is known only to the reviewer and the arbiter and the arbiter is trusted and will not cheat as per our assumption, any message encrypted with this session key during the session can be treated as the digital signature of the reviewer on the message. In most practical implementations, public-key cryptography is used to secure and distribute session keys. Those session keys are used with symmetric algorithms to secure message traffic. Session key establishments are dynamic in nature, whereby the key established by a fixed pair of entities vary on subsequent executions. A key establishment protocol involves generation, transport and confirmation of keys. Authentication of communicating entities is also typically needed in a secure session key setup and is achieved by a signed session key. There are many protocols for signed session key establishment. X.509 [22] recommendations define 'strong two-way' and 'strong three-way' protocols. Both provides mutual entity authentication with optional key transport. Here 'strong' distinguishes these from simple password based methods and two-way and three-way refers to protocols with two and three message exchanges. In all these protocols for key establishment with entity authentication, session keys are either signed by the initiating entity and then encrypted with the public key of the recipient, or the encryption of the session key is done first and then the encrypted session key is signed before transport. Any standard signed session key establishment protocol can be used. For the ease of discussion we present here a protocol, in the line of the X.509 recommendation, integrating entity authentication, key transport and key confirmation together.

Notations

Before discussing the protocol, we present the notations here. The notations are also used in subsequent protocols almost unchanged.

w : a Document Production Workflow

$A_1, A_2, A_3, \ldots\ldots, A_n$: reviewers of w

N : a neutral arbiter, which is an in-line TTP.

CA : an off-line certifying authority, which generates digital certificates of entities like A and N, certifying the association of entity X with its public key.

$k_{i,N}$: shared session key between the i^{th} reviewer A_i and N.

s_X and p_X: secret key and public-key of a principal X; X may be a reviewer A_i or N or CA.

$\{m\}_k$: message m is encrypted with the key k.

$X \rightarrow Y : m$: principal X sends a message m to principal Y and Y receives it intact.

$Cert_X = \{CA, X, p_X, t_v\}_{s_{CA}}$: Certificate of X, where t_v is the expiry time of the certificate.

r_1, r_2, r_3 etc : randomly generated nonces.

t_1, t_2, t_3 etc. : timestamps to test the freshness of messages

The following flags are used in the protocols indicating the intended purpose of the messages transferred in the protocol steps-

f_{rsk} : flag for session key request

f_{ssk} : flag for shared session key

f_{csk} : flag for confirmation of session key.

PROTOCOL 1 : Signed Session Key

Summary :

A reviewer A_i requests N for a session key, N sends a session key $k_{i,N}$ to A_i securely, confirms the key and collects a signed copy of $k_{i,N}$ from A_i, signed by A_i using its secret key s_{A_i}.

Assumptions

-N also serves as session key generator and distributor.

-A_i and N have their own certificates. Additionally A_i has the certificate of

N and N knows the certificate of A_i.

-A_i synchronizes its clock with that of N at the initial start up.

Protocol Steps

1. $A_i \rightarrow N : \{f_{rsk}, \{A_i, r_1, t_1\}_{s_{A_i}}\}_{p_N}$
2. $N \rightarrow A_i : \{f_{ssk}, \{N, \{r_2\}_{k_{i,N}}, k_{i,N}, r_1, t_2\}_{s_N}\}_{p_{A_i}}$
3. $A_i \rightarrow N : \{f_{csk}, \{A_i, r_2, k_{i,N}, t_3\}_{s_{A_i}}\}_{p_N}$

Protocol actions at each step

1. $A_i \rightarrow N$: A_i requests N for a session key. The request is signed by the secret key s_{A_i} of A_i. N can verify the origin of the request. An adversary T cannot masquerade as A_i since s_{A_i} is known only to A_i. The signed request is again encrypted with the public key of N, p_N. This encryption provides privacy, so that none other than N can decrypt the request. Additionally, the request also contains a nonce r_1 for later reference and a time-stamp t_1 so that N can verify the freshness of the request to avoid replay attacks. For the same reason time-stamps are also included in the next two steps.

2 $N \rightarrow A_i$: N decrypts the request received in step 1, verifies the signature of A_i and the freshness of the request. If it is satisfied it then sends a session key $k_{i,N}$ along with other parameters. The nonce r_1 is incorporated to convince A_i that $k_{i,N}$ is in response to the original request in step 1. Another nonce r_2, encrypted with $k_{i,N}$ is provided for the confirmation of the key. The key and the parameters are signed

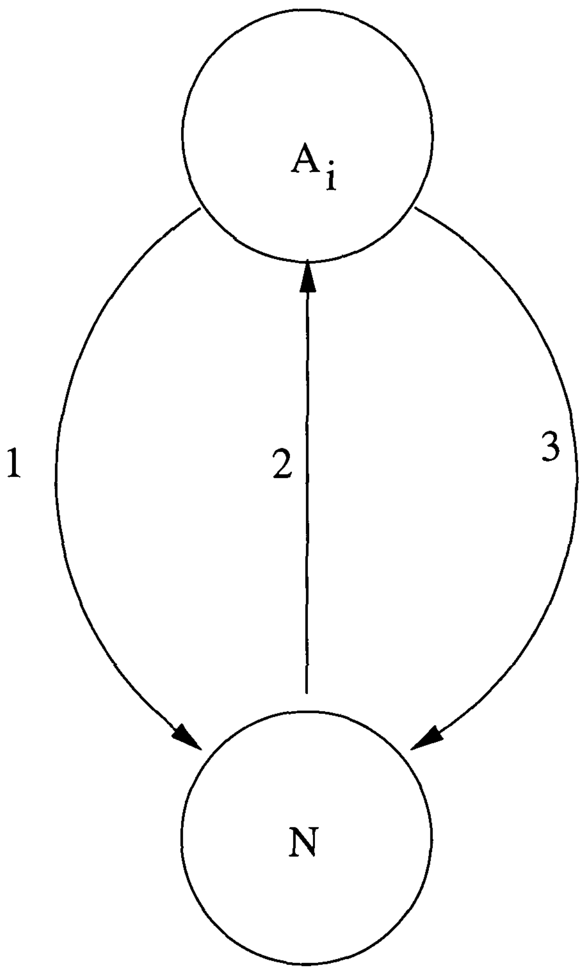

Figure 4.1: Protocol for Signed Session Key

by N using its secret key s_N, and then encrypted with p_A for proof of origin and privacy.

3. $A_\imath \rightarrow N$: $A_\imath$ decrypts the message received in step 2, verifies the signature of N and the freshness of the message and then collects the nonce r_2 by decrypting $\{r_2\}_{k_{\imath,N}}$ with $k_{\imath,N}$. $A_\imath$ sends a signed copy of the session key along with r_2. N verifies the signed copy of $k_{\imath,N}$. Incorporation of plaintext r_2 confirms that $A_\imath$ received the correct key $k_{\imath,N}$. N stores a time-stamped and signed (by N) copy $\{N, \{A_\imath, r_2, k_{\imath,N}, t_3\}_{k_{\imath,N}}, T_N\}_{s_N}$ for later reference. T_N is the time-stamp given by N.

Result : Thus at the end of the protocol, $A_\imath$ gets a session key $k_{\imath,N}$ and N also gets a signed copy of $k_{\imath,N}$ signed by $A_\imath$. Unless N gets a signed copy of $k_{\imath,N}$, it will not allow $A_\imath$ to use $k_{\imath,N}$ in successive transactions with N. Hence step 3 is mandatory for $A_\imath$ to use $k_{\imath,N}$, otherwise, it will be an invalid key.

4.2.3 Production of Cases

In this section we discuss a protocol for production of a case, which is a MPMSD. Production of a case involves case examination and production of a part. The verification of membership of a page to a MPMSD as well as to a context of a DPW is based on the existence of active paths in the Category Graph and it is discussed in Chapter 3.

Notations

In addition to the notations used in PROTOCOL 1, the following notations are used in the protocol for production of cases.

f_{rdpw} : flag for request for a dpw

f_{dpw} : flag for response for a dpw

f_{rpage} : flag for request for a page

f_{page} : flag for response of a page

f_{spart} : flag for submission of a part

f_{nrs} : flag for non-repudiation of submission of a part

pid : pageId of a page. Since timestamps are generated by the Arbiter, which also serves as the storage manager, without any loss of generality, we can use the timestamp associated with a page as unique pageId.

$H(k, P)$: a collision intractable one-way keyed hash function [26] which produces a unique fixed length message digest of a page P using k as an input key to the hashing function.

P_{c_w} : context page of w

P_{log_i} : a log page containing the list of pending cases to be reviewed by A_i.

P_x : a page of category x. x may be $case, part$

PROTOCOL 2 : Case Production Protocol

Summary :

A reviewer selects a dpw . The arbiter sends the context page of the DPW and the pending cases to the reviewer. The reviewer selects a case and the arbiter sends the case. The reviewer composes a page of category part as her own comment and submits it to the arbiter, who plugs it to the case as a new part and to other relevant pages based on the values of the attributes of the profile of the new part. The arbiter now plugs the case to the log page of the next reviewer and also unplugs the case from the previous reviewer. During the composition of the page the reviewer may peruse previous parts of the case and several pages rooted to the context and cite some of them in the page under composition at different points of references as a part of case examination.

Assumptions :

- A_i and N have set up a signed session key $k_{i,N}$ using PROTOCOL 1.

Protocol Steps

1. $A_i \rightarrow N : \{f_{rdpw}, A_i, N, r_1, t_1, w\}_{k_{i,N}}$
2. $N \rightarrow A_i : \{f_{dpw}, N, A_i, r_1, t_2, P_{c_w}, P_{log_i}\}_{k_{i,N}}$
3. $A_i \rightarrow N : \{f_{rpage}, A_i, N, r_2, t_3, pid\}_{k_{i,N}}$
4. $N \rightarrow A_i : \{f_{page}, N, A_i, r_2, t_4, P_{case}\}_{k_{i,N}}$
5. $A_i \rightarrow N : \{f_{rpage}, A_i, N, r_3, t_5, pid\}_{k_{i,N}}$
6. $N \rightarrow A_i : \{f_{page}, N, A_i, r_3, t_6, P_{part}\}_{k_{i,N}}$
7. $A_i \rightarrow N : \{f_{rpage}, A_i, N, r_4, t_7, pid\}_{k_{i,N}}$
8. $N \rightarrow A_i : \{f_{page}, N, A_i, r_4, t_8, P_{ref}\}_{k_{i,N}}$
9. $A_i \rightarrow N : \{f_{spart}, A_i, N, r_5, t_9, pid, P_{part}, H(k_{i,N}, P_{part}), A_{i+1}\}_{k_{i,N}}$
10. $N \rightarrow A_i : \{f_{nrs}, N, A_i, r_5, t_{10}, \{pid\}_{s_N}\}_{k_{i,N}}$

Protocol actions at each step

1. A_i requests for a dpw w, to N. The proof of origin of the message, freshness etc are verifiable as in PROTOCOL 1.

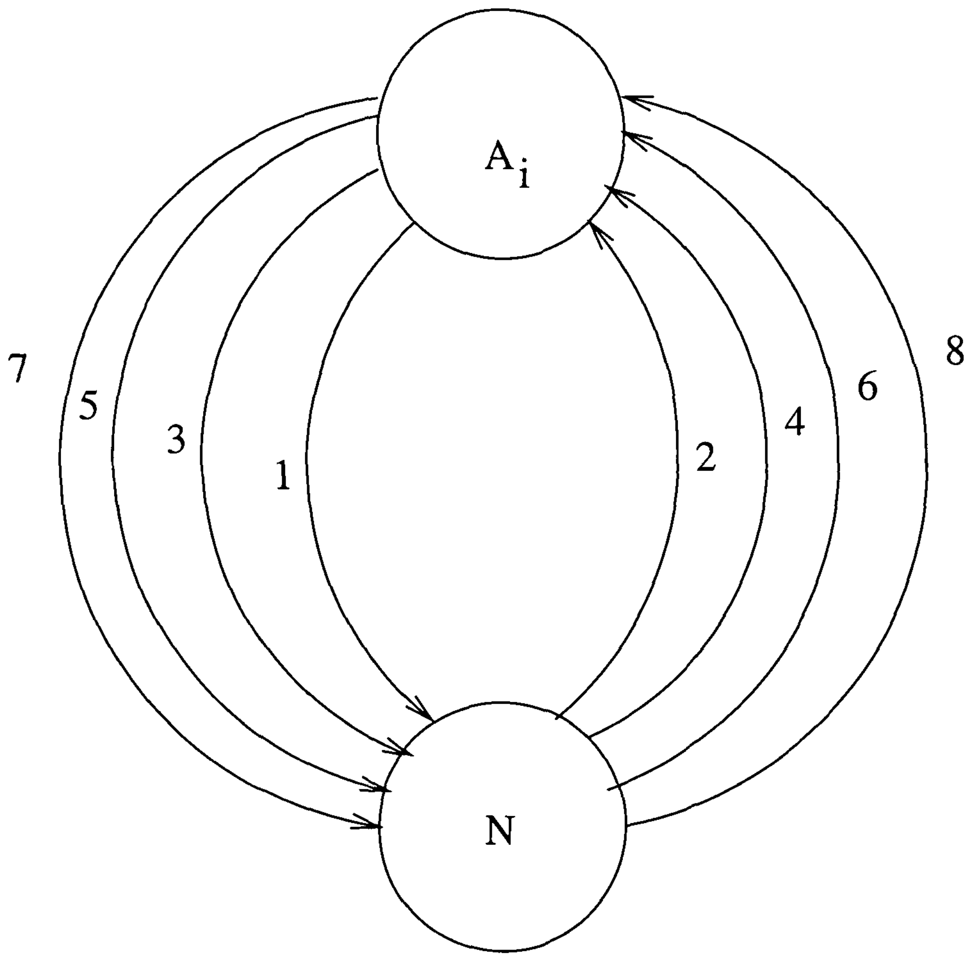

Figure 4.2: Protocol for Case Production

2. If A_i is an authorized reviewer of the dpw w then the context page P_{c_w} of w and the log page P_{log_i} are sent by N to A_i.

3. A_i requests N for a case P_{case} with pageId *pid* for review.

4. If there exists a path from P_{case} to P_{log_i} then A_i is a valid reviewer of the case and P_{case} is sent by N to A_i.

5. A_i requests N for a part of a case P_{part} with pageId *pid* for review.

6. If there exists a path from P_{part} to P_{log_i} then A_i is a valid reviewer of the case and P_{part} is sent by N to A_i.

7. A_i requests N for a reference page P_{ref} with pageId *pid* for perusal during case examination. P_{ref} belongs to the context P_{c_w} obtained in step 2.

8. If there exists a path from P_{ref} to P_{c_w} then P_{ref} is sent by N to A_i. Steps 7 and 8 may be repeated for many times.

9. A_i produces the new part P_{part} as comments on the case. A_i may mark P_{ref} as a citation in P_{part}. A_i generates her signature $H(k_{i,N}, P_{part})$ after completion of composition, and marks A_{i+1} as the next reviewer. If the next reviewer is not explicitly given then the default next reviewer is automatically selected. Finally, A_i submits the new signed page to N as a part to be plugged to the case P_{case}, whose pageId *pid* is obtained in step 3.

10. $N \rightarrow A_i$.: N records the time of receipt, t_r. If there exists a path from P_{case}, whose pageId is *pid*, to P_{log_i} and it is not unplugged and P_{case} is in active state then A_i is the current reviewer of the case. It then verifies the signature of A_i on P_{part} by recomputing $H(k_{i,N}, P_{part})$ and

then comparing with the value received from $A_\imath$ in step 9. If the signature matches then P_{part} is registered with pageId $p\imath d = t_r$. N plugs P_{part} to P_{case}. All citations marked in P_{part} are also plugged by N with t_r as the time of plugging.
Issues of part integrity and the reuse of parts does not arise since the parts are added to the case by the arbiter. Moreover the flow is controlled by the arbiter, hence the access right issue is also automatically addressed. N generates the evidence of non-repudiation of submitting, by encrypting $p\imath d$ with the secret key of N, of P_{part}, stores the original copy of it and then sends a verbatim copy to $A_\imath$. Since N is trusted, if a dispute arises, the original copy available with N will be accepted as the only valid proof. The verbatim copy is available to $A_\imath$ is for information only.

Result: At the end of the protocol a MPMSD is produced as a case. Once it is closed by the last reviewer, it becomes a precedent, goes into in reference state and is automatically plugged by N to P_{c_w}.

In this protocol we have not attempted to provide mandatory proof of receipt, as we do not think it is so important in our problem. Of course, if required then mandatory proof of receipt can be ensured by redesigning the protocol in the line of the fair non-repudiation protocol discussed in [44].

In the protocol, N has multiple roles of a trusted third party, an arbiter in case of disputes, the co-ordinator of the flow of documents and the storage agent of the signed documents. This centralization has enabled us to use shared secret keys and this has brought in efficiency. Session keys will have to be changed periodically to prevent security attacks. However, $A_\imath$ presents a signed (using a public key based scheme) version of the session key being used to N who stores it with a time-stamp. If $A_\imath$'s secret key of the public-key scheme is later compromised, A_i cannot repudiate the signature on the session key as N will certify that it was signed before the time of compromise. In case of a dispute, N will produce all the messages received from A_i using the shared session key and since N is trusted, this evidence will be the basis of resolving the dispute. It is easy to see that all the security requirements enumerated in section 1.8.2 and 1.8.3 are met. For the sake of brevity we do not examine these requirements separately.

4.3 Discussion

In this chapter a security framework has been presented to study the security of the DPW problem. A protocol for secure production of MPMSDs using a central arbiter is also presented. This protocol addresses all the security issues particular to the production of MPMSDs. Although, a formal proof is not provided, it should be clear from the above discussion that no protocol can be designed without a central TTP component to address the security issues of production of MPMSDs.

Chapter 5

Production of Context Pages

Production of cases is discussed in Chapter 4. A case is a MPMSD, produced in the context of a set of relevant documents, where each document is again a MPMSD. The documents relevant to all the cases of a DPW, in general, constitute the *DPW context* for that DPW. A case may have a set of documents relevant specifically to itself. This set of specific documents and the DPW context of the corresponding DPW constitute the *Case Context* of the case. In this Chapter, production of both DPW Context as well as Case Context are discussed. Moreover, production of a state of the context, or of a part there of, based on a given time is a major issue in DPW.

5.1 Production of DPW Context

We recall that a DPW context is a collection of pages of category document, relevant to a DPW. Therefore, a page of category context has links to a set of pages of category document. A DPW has only one context page. Now, the issue is how to define the relevance of pages to a DPW. In conventional information retrieval systems, relevance is normally defined by a set of keywords. There exists models, like Vector Space Model [11, 16], Latent Semantic Indexing [11] etc., for retrieval of relevant documents based on keywords. Keywords alone do not adequately describe an office document. Key words are only a part of the profile of an office document. Other parts are, record attributes, as discussed in the storage framework in Chapter 2, section 2.2.2. These record attributes are represented in the Page Cube as dimensions. Therefore, keyword based definitions of relevance in DPW in an office are not adequate. The relevance of pages to a DPW can be defined by the designer of a DPW using a set of *requirement templates.*

Definition 5.1 *A requirement template is a predicate defined by a set of attribute-value pairs. The attributes are dimensional attributes of the page cube. The attribute-value pairs are in conjunctive form, whereas, the list of values of an attribute are in disjunctive form. A range of values is an alternative form of expressing a list of values.*

Requirement templates based on keywords can also be easily designed. Since well developed models on keyword-based document retrievals are available, we will exclude it from our discussion. But these models can easily co-exist with our Page Cube model.

The basis of construction of the context page of a DPW is a set of requirement templates. Using this set of requirement templates, a query is framed and the result set of the query is the context page of the DPW. For example, a template

T_1 = {category="document", class="rule", topic="s1, s2, s3 ", time="t1-t2"}

signifies that documents containing rules on topics $s1$ or on $s2$ or on $s3$, produced during the time period $t1$ to $t2$ are relevant to the DPW concerned. Sometime, a single document, relevant to the DPW, may have to be included in the context of the DPW. For such a single document, the corresponding template consists of only pageId, *pageId* = *pid*, where *pid* is the pageId of the page.

5.1.1 An Algorithm for DPW Context Maintenance

The context of a DPW can be obtained by searching the entire Page Cube for pages maching the given requirement templates. But this will be expensive to do for every document that is to be created. Therefore, in this section an algorithm for maintaining the DPW context based on a set of requirement templates is discussed. The dynamic nature of the context associated with a DPW is due to the following facts. When a new template is added to the template list by a reviewer during case examination, the set of pages of category document, satisfying the template are to be automatically plugged to the context, if it is not already plugged. It is to be mentioned here, that for the pages of category part corresponding pages of category *document* need to be plugged only. When an existing template for a DPW is removed from the set of templates, corresponding pages plugged to the context are to be unplugged. Since the result-sets of templates are not necessarily disjoint, some pages may belong to the result-sets of templates other than the template to be removed. Such common pages are not to be unplugged. We recall, that unplugging does not necessarily mean deletion of the edges from the Category Graph. It simply means changes to the temporal label of the edges. The history of plugging and unplugging is the basis of creation of the case context of a precedent as discussed in the next section. When a new page of category document is registered in the Page Cube, if it satisfies any of the templates of a DPW, it is to be plugged to the context of the DPW automatically. Moreover, when a page is burned, if the page is of category document then if it satisfies any of the templates of a DPW then it is to be unplugged. Lastly, when an existing template is modified, then consequent changes are to be done to the context.

Let $T_w = \{T_i | T_i$ is a requirement template for dpw="w"$\}$ be a set of requirement templates for a DPW w, R_i be the result-set (of pages) of a query formed by using T_i, and P be a Page Cube. The result-sets of the templates in T_w are not necessarily disjoint. A result-set may contain many pages and at the same time a page may belong to many result-sets of templates of T_w. To take care of the relaton between pages and templates for each DPW a separate Page Table can be maintained, where rows represent pages and the columns represent templates. If a page p_i belongs to the result set of a template T_j of T_w then the $(i, j)^T$ element of the page table will be equal to 1, 0 otherwise. The page table will normally be a sparse binary matrix. Therefore, we can choose an alternative representation of the page table, called Page List, which is a row-wise representation of the page table. With each page a list of corresponding templates in the row with only non-zero entries will be stored. For a DPW w, the structure of the page list is
$PageList_w(Page, TList)$
For example, if a page p belongs to the result sets of templates T_1, T_5, T_7 then the tuple in the $PageList_w$ will be $(p, TList_p)$, where $TList_p = \{T_1, T_5, T_7\}$. With this an algorithm to maintain a DPW context is designed as follows:
ALGORITHM DpwContextMain(T_i or p, P, w, T_w, $PageList_w$)
Algorithm to maintain a DPW Context using requirement templates
INPUT :
T_i : a requirement template which is for either inclusion or exclusion from T_w
p : a page which is either newly registered or burned

P : a page cube

T_w : a set of requirement templates for a DPW w

$PageList_w$: list representation of page table for a DPW w, as described above

OUTPUT :

$status$: a flag, on successful completion the algorithm returns $status = 1$, 0 otherwise

ASSUMPTIONS :

$status = 0$ initially

NOTATIONS :

$include(x, X)$: a function which includes x in the set X as a new member

$exclude(x, X)$: a function which excludes x from X

R_i : result-set of a query

$\sigma_{T_i} P$: select pages from P satisfying the condition formed by template T_i

$p_i \times p_j$: page p_i plugs page p_j

$p_i \div p_j$: page p_i unplugs page p_j

p_{c_w} : DPW Context page for the DPW w

$T_i(p)$: check whether page p satisfies a condition formed by template T_i

$element.attribute = value$: the structure is a short form of writing the value of the attribute of an element

BEGIN

begin case

```
step 1: case 1: a new template $T_i$ is submitted for inclusion in $T_w$
step 1.1:  $T_w \leftarrow include(T_i, T_w)$
step 1.2:  $R_i \leftarrow \sigma_{T_i} P$
step 1.3:  for every $p \in R_i$
step 1.4:      if $\exists$ a tuple $t \in PageList_w$, where $t.page = p$ then
step 1.5:             $TList_p \leftarrow include(T_i, TList_p)$
                      // update the template list of existing page
                else
step 1.6:             $TList_p \leftarrow T_i$ //new entry in $PageList_w$
step 1.7:             $PageList_w \leftarrow include((p, TList_p), PageList_w)$
step 1.8:             $p_{c_w} \times p$ //plug the page to the context page
                endif
            end for
step 2: case 2: an existing template $T_i$ is submitted for exclusion from $T_w$
step 2.1:   $T_w \leftarrow exclude(T_i, T_w)$
step 2.2:   $R_i \leftarrow \sigma_{T_i} P$
step 2.3:   for every $p \in R_i$
step 2.4:       select $(p, TList_p)$ from $PageList_w$
step 2.5:       $TList_p \leftarrow exclude(T_i, TList_p)$
step 2.6:       if $TList_p = null$ then
step 2.7:             $PageList_w \leftarrow exclude((p, TList_p), PageList_w)$
step 2.8:             $p_{c_w} \div p$ //unplug the page from the context page
                endif
            end for
step 3: case 3: a new page $p$ of category="document" is registered
step 3.1:   if $\exists T_i \in T_w$ and $T_i(p)$="true" then
step 3.2:       $PageList_w \leftarrow include((p, TList_p), PageList_w)$
step 3.3:       $p_{c_w} \times p$ //plug the page to the context page
            endif
step 4: case 4: a document $p$ of category="document" is burned from $P$
step 4.1:   if $\exists T_i \in T_w$ and $T_i(p)$ ="true" then
step 4.2:       $PageList_w \leftarrow exclude((p, TList_p), PageList_w)$
step 4.2:     $p_{c_w} \div p$ //unplug the page from the context page
            endif
step 5:     case 5: a template $T_i$ is modified to $T_i'$
step 5.1:   do step 2 to exclude $T_i$ from $T_w$
step 5.4:   do step 1 to include $T_i'$ to $T_w$
step 6:     return(1)
end case
END
```

This algorithm maintains the DPW context of a DPW w. All the five cases, where the context may change, are taken care of in the algorithm. In case 1, when a new template is added to the template list if the page p is already plugged, then only the $TList_p$ of p is modified by including the template (step 1.5) otherwise a new page entry is made in the $PageList_w$ (step 1.7), and the page is plugged to the context page (step 1.8). In case 2, when a template is excluded from the template list, the result-set of the template is regenerated and for every page belonging to the result-set the template is excluded from the corresponding *TList*. After exclusion, if the *TList* of the corresponding page is not empty, it means that the page is also common to the result-sets of other templates in the list. Hence, the page is not unplugged. If the *TList* is empty then the page is not common to any other template and hence its entry in the PageList is deleted and it is unplugged from the context page. Case 3 and 4 are logically simple and straight forward. But the efficiency depends on how to find out whether the page satisfies any template of a DPW, and this depends on the data structures that are used. Similar situations are handled in Active Databases using rules [11]. Modification of a template is a two step process: exclusion of a template and inclusion of a new template.

From the perspective of security, we can put the following restriction on operations on a context. A reviewer cannot unplug any page from the context page of a DPW but can plug new pages by incorporating suitable requirement templates in the template set. Only the designer of the DPW, who may be a higher authority in an office, not necessarily a reviewer of the DPW, can exclude or modify an existing template from the template list of a DPW.

5.1.2 Retrieval of a DPW context

Once a DPW context is maintained using the algorithm $DpwContextMain()$ discussed in the previous section, the question arises as to how to retrieve the current state or the state at a particular time of a DPW Context. A $GetDpwContext()$ algorithm can be designed, similar to the $GetDocument()$ algorithm discussed in Chapter 3. A DPW context will have only *docPart* kind of edges.

ALGORITHM GetDpwContext(w, t)

Algorithm to retrieve the state of the DPW context page p_w, for the DPW w, at time t

INPUT :

w : dpwId of a DPW

t : time which defines the state of p_w

OUTPUT :

p_w^t : state of the page p_w at time t

ASSUMPTIONS :

A DPW has only one DPW Context page

NOTATIONS :

P : a page cube

null : a constant with value 0

now : a timestamp, which signifies the present time

element.attribute = *value* : the structure is a short form of writing the value of the attribute of an element

```
BEGIN

step 1: $p_w \leftarrow \sigma_{category="context" \wedge dpw="w"}(P))$ // selects the DPW context page
step 2: From $p_w$, form a set of forward edges of kind $conDoc$
            $L^s_+ = \{e^s | e^s.source = pid$ and $e^s.kind = conDoc$
            and $e^s.dir = +$ and $e^s.t_+ <= t$ and $(e^s.t_- > t$ or $e^s.t_- = null)$ $\}$
step 2.2: if $L^s_+ \neq null$ then
            sort edges in $L^s_+$ in ascending order of $e^s.t_+$
            endif
step 3 :    return($p^t_w$)

END
```

The algorithm *GetDpwContext*() returns the DPW context page p_w, with a set of of edges to the pages of category *document* that were plugged but not unplugged from the context page p_w before time t. Unlike *GetPart*() or *GetDocument*() algorithms, discussed in Chapter 4, there is no attemp to concatenate the contents of all documents defining the state of p_w at time t. It can be done logically, but normally, in practice, a such concatenated DPW context page is likely to be huge. Therefore, we assume that on selection of an edge from p^t_w thus returned, the implementation software will supply the pageId of the target document of the edge selected and t to the *GetDocument*() algorithm and thus the document can be retrieved.

5.2 Production of Case Context

We recall from the definition of Case Context (Chapter 2, definition 2.8), that the context of a case is the set of documents specific to the case, plugged at different times by different reviewers of the case under a DPW and also the DPW context page for the DPW. Since a case is a MPMSD, for every part of a case, the state of the context may be different. In the present section we discuss the retrieval of a state of the Case Context for a case at a particular time. Normally, at the time of creation of a particular part of the case, a case context is automatically produced by the arbiter and it is registered as a separate page with a pageId. Initially the case context is plugged to the DPW context of the DPW of the case with *conCon* kind of edges. The case context, in turn, is plugged to the corresponding case. Every case will have one and only one case context page, and every case context will have one and only one case. At the beginning of the review process of a case under a DPW, the case context will contain only one edge to the DPW context. As soon as a reviewer finds a relevant document specific to the case under examination, he/she will plug the document to the case context. Thus a case context page will have one forward edge of kind *conCon* to the DPW context page and zero or more forward edges of kind *conDoc* to relevant documents.

The algorithm to find the state of a case context, at a particular time, say $GetCaseContext()$, is similar to the $GetDpwContext()$, discussed in the previous section. The only difference is that, it will call the $GetDpwContext()$ algorithm once inside $GetCaseContext()$ to get the set of relevant documents available in the DPW Context at time t and then find the set of documents available in the Case Context at time t. Union of these two sets of documents defines the state of the context of a case at time t. Therefore the $GetCaseContext()$ algorithm is not given explicitly.

5.3 Discussion

In this Chapter maintenance and productions of both DPW Context as well as Case Context have been discussed. The algorithms for production of these two types of contexts have been presented. In Chapter 3, we mentioned that the category graph is a mandatory dimension graph for the DPW problem. Without the category graphs it is not possible to handle the issues relating to DPW contexts and Case Contexts. Thus the Page Cube model and the algorithms discussed in this Chapter together provide solution of organizational memory hitherto not addressed in office information systems. Here, the page generated to represent a state of an actual page, stored in a page cube, at a particular time is not a persistent page. Such a page is generated on the fly during case examination and hence it need not be registered in the Page Cube.

Chapter 6

Authorization

Authorization is an important security aspect of a DPW. All subjects of an office are not authorized to access all objects. In a DPW the subjects are the users of the office system and the objects are mainly the pages belonging to a page cube. Authorization in a DPW environment is dynamic in nature. Along with the document flow, authorization also flows synchronously. The existing authorization models for general workflow are briefly reviewed in this context and found that they are not sufficient to address the issues of authorization in case of a DPW. In an office environment, who accessed which object, at what time, is also an important aspect for office security. Therefore, an *authorization audit trail* is also equally important. In this chapter, an authorization policy and an authorization model suitable for a DPW are discussed.

6.1 The Authorization Policy

Before discussing the authorization model, we discuss here an authorization policy which comprises of the following elements:

1. *Authorization Administration*: A centralized policy for authorization administration is adopted. Since an in-line TTP called an arbiter is mandatory for secure production of MPMSDs, therefore the arbiter is also made the central authority for authorization administration. Only the arbiter can grant or revoke authorization to or from a subject for an object.

2. *Access Protocol*: A subject other than the arbiter cannot access the objects directly. The subject, submits an access request for an object to the arbiter. The arbiter finds out whether an authorization can be granted by consulting the set of authorization rules. If the request is as per rules then the arbiter grants an authorization, otherwise it denies the access. If an authorization is granted then the arbiter performs the access requested on the object on behalf of the subject and the result is communicated to the subject in response to the request.

3. *Authorization Flow*: The authorization in document production workflow is dynamic in nature. When a case flows from reviewer to reviewer, authorization on the case also flows from reviewer to reviewer synchronized with the document flow. Authorization on an object is granted and revoked based on the occurrence of a pair of conjugate events. For example, authorization of a reviewer on a MPMSD for reading previous parts and adding comments to the case is granted as soon as the document is received by the reviewer and the authorization is revoked as soon as the the case is sent to the next reviewer.

4. *Object Privacy Policy*: Information stored in the objects are always encrypted with an internal encryption key and the encryption key is known only to the arbiter. This is to enhance privacy to mitigate the

threat even from the system administrator of the host system.

6.2 General Authorization Models

The most general authorization model is probably the Discretionary Access Control (DAC) using access control matrix model, introduced by Lampson, and afterward developed by Graham and Denning and Harrison, Ruzzo and Ulman[32]. The model is used as a security model in operating systems and in database environments. The model represents the authorization as a matrix. Let A be the authorization matrix. The matrix rows correspond to the subjects and the columns to the objects. Entry $A[s, o]$ contains the access modes for which subject s is authorized on object o. Since the matrix is usually quite sparse, typical implementations adopt one of the three representations: *Access Control List(ACL)*, *Capability List(CL)* and *Authorization Table(AT)*.

The ACL approach represents the corresponding access matrix by column. To each object o, a list of pairs $(s, A[s, o])$ is associated indicating the subjects and their access modes on the object o. Therefore, only non-null matrix entries are considered. In this approach all subjects granted access on an object can be easily found; however, it is inefficient to look for all the objects a subject can access. The CL approach represents the corresponding access matrix A by row. To each subject s, a list of pairs $(o, A[s, o])$ is associated for each object o such that $A[s, o]$ is not null. Therefore, if a subject holds no rights on an object, this object does not appear in the list. In this approach, given a subject, all the objects, the subject is authorized to access can be easily found; however, it is inefficient to find out the set of subjects granted access on a given object. The paradigm of authorization in ACL as well as in

CL approaches is distributing the authorizations either object wise or subject wise. In ACL-based operating systems and databases, in general today, the owner of an object can grant or revoke or modify access rights. Similar is the case with CL-based systems. Moreover, the presence of a *superuser* or *system administrator*, whose power is unlimited, can be a major threat of security breach in office document systems.

The AT approach represents the access matrix by a central table of tuples $(s, o, A[s, o])$. This approach has also advantages and disadvantages. The AT is suitable for centralized authorization administration, where the authorization granting and revoking is done by a central authority. Finding a set of authorized objects for a given subject and a set of subjects authorized to access a given object is easier by simply querying the table. The document production in a DPW is done under a central authority, called the arbiter. In the document production workflow environment, where the authorization is dynamic and propitiatory in nature, the authorizations for a reviewer are granted for a fixed time period only and the authorization propagates to the next reviewer, the authorization table representation is easier to manipulate. The main disadvantage of the AT approach is related to the size of the table. The table may contain many tuples of inactive subjects/objects making the table size large.

Role-Based Access Control(RBAC) appears to be unavoidable in all groupware solutions in office environments. A user is allowed to access an object on the basis of the role(s) that the user plays at the time of access. If the responsibility of a user s changes due to transfer to another department of the office or due to promotion etc. current role of s can be reassigned to his/her replacement, and new roles can be assigned to s as required by the new responsibilities. It is not necessary to go through the ACLs of all the objects one by one to remove entries pertaining to s in his/her previous capacity and add entries pertaining to the new capacity. Roles greatly simplifies the task of security management. Moreover, this enables a user to play multiple roles with different responsibilities, which is common in an office. RBAC also has a disadvantage in DPW. For example, If a document is sent to a reviewer and the authorization is granted to the role of the reviewer, then any other employee belonging to the same role can at least read the document, which may be a serious breach of security.

6.3 The Workflow Authorization Models

Synchronization of *authorization flow* with the workflow is a fundamental security requirement in workflow environments [4]. Other essential requirements include role based security policy and separation of duties [9]. Separation of duties are imposed to reduce the risk of frauds by not allowing any individual to have sufficient authority within the system to perpetrate a fraud on his own. WFMSs like Lotus Notes provide role-based access control but do not have a formal model to synchronize authorization flow with the workflow. Recently Atluri and Huang [4, 5] proposed a Workflow Authorization Model(WAM) that provides synchronization of authorization flow with workflow, role-based authorization and separation of duty. The WAM model properly addresses the issue of authorization flow when there are temporal constraints in the definition of tasks of the workflow. This means that there is a fixed time of starting and completing a particular task and these are defined a priori in the workflow template itself. The genesis of the model is in automatic time-bound execution of tasks and the corresponding authorization flow. Our problem is slightly different. We cannot set a priori the time period for a review process. We need event-based authorization flow. Apart from temporal constraints, a DPW has other types of constraints as discussed in section 6.1.

6.4 The Authorization Model for DPWs

In this section we propose a DPW Authorization Model (DPWAM). Authorizations in DPW are event-based and dynamic in nature. By dynamic we mean that authorizations are granted to a subject, acting as a reviewer, on some objects based on occurrence of an event and the authorizations are to be revoked automatically on occurrence of the other conjugate event. That is, the subject will have authorizations for certain privileges only during the time period between the occurrences of the conjugate events. The conjugate events may vary from object type to object type. For example, a reviewer of an active MPMSD under review, will have the authorization for the privilege to read the previous parts and to comment on the document as soon as she receives the document. The authorizations will be revoked as soon she sends the document to the next reviewer. That means the reviewer will not be allowed to read or comment on the document after forwarding it to the next reviewer. Here *receive* and *send* are the conjugate events. In case of paper document, since the document itself is moved physically from the current reviewer to the next reviewer, therefore the revocation of authorizations is automatically accomplished. In a more complex situation, the revocation may be partial. Even after forwarding the document, the i^{th} reviewer may be allowed to read the document, but only up to the i^{th} part of the document. In a DPW, *request* and *response* for a page may also be conjugate events.

The page cube model can be extended in the following way to incorporate event-based dynamic authorization model for a DPW. An office worker may play one or more roles in a an office. Some roles may be grouped together to form a larger role. This forms a role hierarchy. Therefore, *role* may be a new dimension of the page cube. To take care of the disadvantage of general RBAC model for a DPW, discussed in section 6.2 we may assume that the leaf nodes of the role hierarchy are *atomic* and the internal nodes are non-atomic and consists of either atomic or non-atomic children roles. By atomic, we mean that only one user can be assigned to an atomic role. Normally in a DPW, atomic roles are assigned as reviewers. In an office, atomic roles can be specified from a non-atomic role. For example in a University there are three Assistant Registrar. *AssistantRegistrar* is a non-atomic role, which consisists of three atomic roles : Assistant Registrar(Finance), Assistant Registrar(Academic), Assistant Registrar(Administration).

So far, we discussed only intra-dimensional graphs in our Page Cube model. Apart from the intra-dimensional graphs, limited within a dimension, like the category graph, a Page Cube may have Inter-Dimensional Graphs (IDG).

The DPWAM comprises of five components: *User-Role Graph(URG), Role-DPW Graph(RDG), Rule-Base(RB), Authorization-Base(AB)* and *Authorization Audit Trail(AAT).*

6.4.1 User-Role Graph

A user may be assigned to many roles and a role may be played by many users, if the role is non-atomic. Therefore, it is a many to many relation. An user may be assigned to a particular role at a particular time and the assignment may be revoked at some other time. The history of user role assignment may be captured as an IDG.

Definition 6.1 *A User-Role Graph is a bipartite graph where user and role are the two types of nodes. An assignment is represented by an edge which connects a user node with a role node and the edge is labelled as* (t_+, t_-)*, where* t_+ *is the time of plugging (assigning) and* t_- *is the time of unplugging(revocation of assignment). A user remains active in a role till it is unplugged.*

In an office not only the active assignment of users to roles, but also the record of history of assignments is equally important for security auditing and dispute resolution. Using a page cube, this is easily accomplished

6.4.2 Role-DPW Graph

A role may have access privileges to many DPWs and a DPW may have many roles as reviewers. Due to obvious reasons, definition of authorization rules based on role rather than on users is easier. A role may be assigned to a particular DPW at a particular time and the assignment may be revoked at some other time. The history of role-DPW assignment may also be captured as an IDG, called *Role-DPW Graph.*

Definition 6.2 *A Role-DPW Graph is a bipartite graph where role and DPW are the two types of nodes. An assignment is represented by an edge which connects a role node with a DPW node and the edge is labelled as* (p, t_+, t_-)*, where* p *is the position of the role as a reviewer in the DPW,* t_+ *is the time of plugging and* t_- *is the time of unplugging. A role remains active as a reviewer of a DPW till it is unplugged.*

Within a DPW, the same role may appear as reviewer more than once but at different positions. A connected pair of nodes in both URG and RDG may have parallel edges with different labels.

6.4.3 Rule-Base

Authorization constraints for a DPW can be represented as a rule. A rule can be defined in Event-Condition-Action (ECA) paradigm. The structure of a Rule-Base is

Rule-Base(ruleId, event, conditions, actions, privileges, t_+, t_-**)**

ruleId is the unique identifier of a rule. Event may be one of the pair of conjugate events: *request, response.* The conditions may be of two types: conditions on the subjects and the conditions on the objects. The conditions on the subject may be defined by the predicates on user, user-role graph, role-DPW graph etc. Whereas, conditions on objects may be defined by page profiles, dimension hierarchies, dimensional graphs etc. For example, path existential conditions used in the protocol for production of cases discussed in Chapter 4 is an object condition. The conditions may have a partial order. Subject conditions followed by object conditions. Within a subject condition, the partial order may be user > user-role > role-dpw. Similar is the case in object conditions. The actions are basically two: *grant* and *revoke* authorizations. The privileges may be *read, write, plug, unplug, close, reopen, archive, dearchive* and *burn.* t_+ signifies the time of inclusion of a rule in the Rule-Base and t_- signifies the time of exclusion of a rule from the Rule-Base.

On occurrence of an event, a rule is triggered. If the conditions are true, actions are taken: either to grant an authorization for certain privileges or revoke an authorization. An active rule-based authorization constraint modelling for general workflow is available in [12, 13].

6.4.4 Authorization-Base

Authorization-Base is a collection of authorizations granted and revoked during production of pages in different DPWs in an office. An authorization is a tuple as defined in Chapter 4, (4.5). Here subject attribute is replaced by three attributes: (*user, role, dpw*). The modified authorization tuple is

(user, role, dpw, object, privileges, t_+, t_-**)**

6.4.5 Authorization Audit Trails

In the DPWAM model, the size of AB grows fast. The growth of AB will have an effect on performance. Moreover, to keep AB small, we cannot simply delete the authorization tuples from AB as soon as it is revoked. Because, in an office audit trail of access of objects may be an important function. For example, who accessed a particular page, during a particular time can be found out from AAT. The access records are also persistent in nature. Therefore, the records are to be stored securely. One simple way is to move the revoked authorization tuples from AB to a Revoked Authorization-Base (RAB) of similar structure. The RAB will be maintained in reference storage and this will be accessed very rarely, only during audit. Hence its growth will have negligible effect on overall performance. Moreover, it can be archived. RAB will serve for audit trails for document access history in an office.

6.5 Discussion

In this Chapter authorization for DPW problem is discussed. The authorizations are dynamic in nature and moreover history of authorization is also a security requirement of an e-office. Therefore, issues of authorization for DPW are addressed with the help of an authorization model proposed here. DPWAM is an extension of Page Cube model. The authorization management is a central one controlled by the arbiter. Since the arbiter is necessary for other aspects of security, as discussed in Chapter 4, therefore, authorization management is a natural extension of the responsibility of the arbiter.

Chapter 7

Implementation

In the present chapter, an outline of the implementation of DPW architecture using the state-of-the art technology is discussed. Only the outline of an implementation is given here. No implementation has been done as a part of the present work. Implementation of the system has not been done, because of the following reasons:

- The system is complex. Even a prototype implementation will take many man hours.
- Our focus is more on academic interests, mainly identification of issues and logical solutions. Implementation is beyond our focus.
- An office automation software consists not only of DPW, it also consists of legacy software for financial accounting, database, e-mail, conferencing etc. Therefore, the DPW software is to be integrated as a component with other commercial office automation software, like Lotus Notes. A DPW software itself cannot survive as an alternative to Lotus Notes. The integration can only be done as a joint venture with the authority of the software.

- Tools like workflow, XML etc. required for DPW were in a primitive stage when the work was started. Vendors like Oracle, only recently incorporated workflow and XML in their new products. But the technology as a whole is still in a flux, and is yet to settle down as accepted standards. In such a situation, an implementation may not be long lasting.
- The security algorithms and protocols are proprietary in nature. There are restrictions in using them.
- Acceptability and performance of such a system is meaningful when we get feedback from real life offices using such a system, which again is not possible with the present setup of ours.

In fact a part of DPW architecture was implemented in a project in IIT, Guwahati, based on our papers [36, 37]. Here the security and context part was not fully implemented. The prototype implementation is working fine and the performance is reported as satisfactory [33].

7.1 An Outline of an Implementation

Java introduces a new model of client/server interaction for the Web. A small component like program called *applet* can be written in Java and can be downloaded into a browser that is Java compatible. An applet is portable, can be executable inside a java-enabled browser on any computing platform. Java applet provides, on its own, a secure environment within a browser. Web and Java provides a solution for distributed computing but the solution is not complete, particularly in workflow automation environment. HTTP is a stateless protocol and the interaction is initiated by the client (*client pull*).

The problem lies in the basic dichotomy between the Web paradigm and the workflow paradigm. The Web is composed of a collection of non-persistent state objects, whereas workflow, in particular office document production workflow, requires a degree of persistence of state among its objects to enable the completion of the assigned task. Moreover, Workflow very often needs server initiated interaction (*server push*). Of course, Netscape and Microsoft have announced support for push technology for scheduled broadcast communication. This needs to be enhanced for an event-based push.

The architecture for the DPW can be implemented using Web technology. The client environment may be a general Java enabled Web browser, where the client is a *Java applet*. Agents of the client logic layer may be implemented as Java objects in the applet. The interfaces of the view module can also be implemented in the applets. The protocols can be implemented on the top of the HTTP protocol. Since server push mechanisms are not yet matured, we can modify the protocols, where necessary, with client pull only. The manager modules of the server logic can be implemented as *Java servelets*. Of course, server push can be simulated using the Java RMI mechanism. In DPW, server push is required in a scenario like, a reviewer is online and a case is received by the arbiter for the reviewer, the arbiter updates the inlog of the reviewer in the storage and the same updated inlog is to be sent to the online reviewer. Here the communication is initiated by the arbiter.

Alternatively, the DPW architecture can be realized in a distributed object model. We have a matured *de facto* as well as *de jury* open standard for distributed object infrastructure, that is *Common Object Request Broker Architecture (CORBA)*. CORBA objects can exist anywhere on a network, and their location is completely transparent. Details such as the language in which an object is written or the operating system on which it currently runs are also hidden to clients. The implementation of an object is of no concern to an invoking object; the interface is the only consideration a client must make when selecting a serving object. CORBA provides an open environment built for integration, change and evolution, and is suitable for office automation software. The standard CORBA objects are grouped as *CORBAservices, CORBAfacilities* and *CORBAdomains*. The CORBA services provide system-level services which may be used to design the objects for other groups. The CORBAservices provided in CORBA 2.0 are: *Concurrency, Events, Externalization, Licencing, Lifecycles, Naming, Security, Time, Trader, Start-up, Persistence, Properties Query, Relationship, Transactions, Collections etc.*. CORBAfacilities, which provide horizontal and vertical application frameworks used by business objects, include *User Interface, Information Management, System Management, Task Management etc.*. The CORBAdomains provide object solutions to some standard domains like Financial services, Health-care, Telecommunications etc. Application objects can be designed from these standard objects though inheritance and that is what exactly needs to be done in realizing the DPW architecture. Another advantage of CORBA is that legacy software can be integrated in a CORBA-based architecture using CORBA wrappers. Details are available in [23]. CORBA 2.0 also incorporates a binary protocol for communication between ORBs, called the *Internet Inter-ORB Protocol(IIOP)*, through which two ORBs can talk to each other. IIOP and HTTP can coexist in the same setup.

The DPW architecture can be implemented within the framework of CORBA so that it can be a component of an open system and can be integrated easily with existing software packages. With the CORBA object bus, ORB, in both the client as well as server side, a Java object can directly talk to the server or vice-versa. The client environment is the standard web browser. The client is a Java Applet which will run in the browser. The agents of client logic module and corresponding interfaces are designed as Java objects in the applet. A Java based lightweight CORBA ORB, called Java ORBlet, is loaded in the client side. The objects of the client applet can communicate among themselves or with remote server objects through the interfaces and stubs of the ORB. The applet can be initially loaded using a HTML page and HTTP protocol. The client authentication can be implemented by authenticating the applet during loading, using Sun Microsystem's *applet certification scheme.* Initial synchronization of clock of a client with that of the arbiter can be done by using the CORBAservice called*Time.* The Crypto Agent of the client can be designed as a derived object which inherits the*Security* and*Time*objects of the system level CORBAservices. The Security service of CORBA provides authentication, confidentiality, content integrity, non-repudiation etc. The crypto can get time attributes for timestamping by inheriting from Time service. The User agent also inherits some traits from Security service. The other services like Naming and Event etc. can be automatically used by ORB.

On the server side, the managers: Production Manager, Worflow Manager, Storage Manager and User Managers of the architecture can be implemented as CORBA server applications. CORBA server applications are regular CORBA objects. The CORBA servers are plugged to an ORB object bus. If the HTTP server is CORBA compatible then it can be directly plugged to ORB otherwise it can be wrapped with a CORBA wrapper to make it CORBA compatible. Similarly a standard workflow engine can be wrapped to make it a workflow manager. Of course, CORBA incorporated a workflow service in its recent version. In that case, the workflow manager can be implemented inheriting this service. The storage manager can be implemented by inheriting from the Persistence and Query services of CORBAservices. The Persistence service provides a single interface for storing objects persistently on a variety of storages- including Object Databases, Relational Databases and simple files. PSQL can be implemented inheriting traits from Query services.

7.2 Implementation of Page Cube

The data storage model for the DPW is Page Cube, discussed in Chapters 3,5 and 6. The Page Cube can be implemented in different ways. One of the following approaches can be taken.

7.2.1 Relational Database Approach

The Page Cube model can be implemented using relations. It is useful to consider these relations as part of a multi-dimensional *data cube* model used in data warehousing. The relations are then regarded as dimension tables and the hierarchical relationship can be represented by either a *star schema* or the *snowflake schema* [28]. In the star schema, the dimension tables are not normalized. Where as in the snowflake schema, they are normalized. Based on the complexity of the dimensions, we can choose one of the two alternatives. The advantage of a star schema is that it is easy to define hierarchies and it reduces the number of joins. It is a common practice in multi-dimensional modelling that when the dimensions are simple in nature, such dimensions are incorporated into the central fact tables as attributes as there is no justification of maintaining a separate dimension table [29]. The advantage is that expensive join operations between the central fact table and those dimension tables can be avoided thereby improving the efficiency. Of course, there is no clear rule for such justification. It is a design decision. In the schema of the page cube, some of the dimensions can be incorporated in the central fact tables. *Time, Category, Class* and *Status* are such dimensions which can be incorporated in the central tables without any loss of generality. Time is a significant dimension and needs a special treatment for it. Once time is incorporated as an attribute of the central tables then there is no need for a separate timeId. Since, in our scheme, the central arbiter will provide the time, a unique time can be assigned by the arbiter to a page. Therefore, it can also be used as a unique identifier for a page, the pageId. According to this modification, the arbiter ensures that at a particular time one and only one page will be created. A representative simple star schema of PC is as follows:

Dimension Tables:

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

Other Central Tables:

Production (pageId, topicId, typeId, category, class, userId, roleId, domainId, dpwId, ...)

Storage(pageId, location, size, signature, state, pageText,....)

Flow(pageId, senderId, senderRoleId, sentTime, receivedTime, ReceiverId, ReceiverRoleId)

Relations to represent graphs: DGs and IDGs

Citation(citingPageId, citedPageId, plugTime, unplugTime, citeCount)

DocumentPart(documentId, partId, plugTime, unplugTime)

ContextDocument(contextId, documentId, plugTime, unplugTime)

CaseContext(caseId, contextId, plugTime, unplugTime)

PartPortion(partId, portionId, plugTime, unplugTime, kindOfegde)

userRole(userId, RoleId, plugTime, unplugTime)

RoleDpw(RoleId, dpwId, position, plugTime, unplugTime)

The authorization base and revoked authorization base relations are discussed Chapter 6 and are not incorporated here again. The Rule-Base for authorization discussed in Chapter 6 can be implemented as active rules of active database systems. Authorization as active rules is also discussed in [13] and the rule structures can be taken from this work. The names of the relations and of the attributes are chosen in such a way that they are self explanatory. For brevity, a data dictionary for the schema is not provided. A page may be on more than one topic, may also belong to more than one class. Therefore, the normalized Production relation will have more than one tuple with the same pageId. Therefore, pageId alone cannot be the primary key of the relation. Without any loss of generality, if we consider multiple values of topicId as well as of class as a single atom, where values are delimited somehow, then the given denormalized relations serve the purpose. The content of a page is considered as an attribute, pageText, in the relation Storage. But in practical implementation each page can be stored as a separate file, with pageId as the file name, and only the location of the file in the file system is stored in the location attribute.

7.2.2 XML Database Approach

The present trend for digital document encoding and interchange is to use descriptive markup. In the light of the state-of-the art technology, XML is the best tool for describing contents of office documents. Therefore, it can be a suitable tool for implementing the page cube model. An XML page cube is a collection of XML pages. The root tag of a page is $< page >$. The $< page >$ tag contains three tags $< profile >$, $< body >$ and $< tail >$. The $< profile >$ describes the dimensions of the page cube, $< body >$ describes the content of a page. DGs and IDGs are implemented by nesting of elements in the $< body >$ or with external references as links. The $< tail >$ section contains external reverse links of different graphs. The structure of a sample XML page is as follows:

```
<?xml version="1.0" ?>
<!DOCTYPE page SYSTEM "pageCube.dtd">
<page id="2001.12.30.13.56.45">
     <profile>
         <dimension>
           <type id="  "> message.officeOrder.notice </type>
           <topic id= "  " > leave.casual.special </topic>
           <category> document </category>
           <class> case </class>
           :
           :
```

```
    </dimension>
</profile>
<body>
    <olinklist>
       <link>  first part
          <kind>docPart</kind>
          <target>2001.12.30.13.50.59</target>
          <ptime>2001.12.30.56.46</ptime>
       </link>
       <link>  second part
          <kind>docPart</kind>
          <target>2001.12.30.14.15.10</target>
          <ptime>2001.12.30.14.15.12</ptime>
       </link>
       :
       :
    </olinklist>
</body>
<tail>
    <olinklist>
       <link>
         <kind>docPart</kind>
         <target>2001.11.09.13.50.59</target>
         <ptime>2001.12.09.56.46</ptime>
       </link>
       <link>
         <kind>docPart</kind>
```

```
            <target>2001.10.30.14.15.10</target>
            <ptime>2001.12.12.14.15.12</ptime>
          </link>
          :
          :
      </olinklist>
    </tail>
</page>
```

The dimension hierarchies can be implemented as nesting of tags in $< profile >$. Each dimension may have a separate page with a distinct page ID or all dimension hierarchies may be defined in a single page. In the dimension page(s) the < *body* > and < *tail* > section may be naturally null. For checking the validity of a page belonging to the XML page cube a Document Type Definition(DTD) is to be designed, which specifies the grammar of elements (tags), attributes of tags and there relationships. A representative XML DTD, named *pageCube.dtd*, is as follows:

```
<!DOCTYPE page [
<!ELEMENT page (profile, body, tail)>
<!ELEMENT profile (dimension)>
<!ELEMENT dimension (type?, topic?, time?, category?, class?,
                     user?, domain?, dpw?, state?)>
<!ELEMENT type (type*, name)>
<!ELEMENT topic (topic*, name)>
<!ELEMENT time (year, month, day, hour, minute, second, msec)>
<!ELEMENT category (#PCDATA context | document | part)>
<!ELEMENT class  (#PCDATA rule | case | support| )>
<!ELEMENT user (name, address)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (mail, phone*, email*)>
<!ELEMENT mail (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT domain (domain*, name)>
<!ELEMENT role (role*, name)>
<!ELEMENT dpw (name)>
<!ELEMENT state (#PCDATA) born|active|reference|archived|expired>
<!ATTLIST (type, topic, domain, dpw) id ID #REQUIRED>
<!ELEMENT body (olinklist*, linklist*,........)>
<!ELEMENT olinklist (link+)>
<!ELEMENT oinklist (link+)>
<!ELEMENT link (#PCDATA, kind, target, ptime, uptime?)>
<!ELEMENT kind (#PCDATA)>
<!ELEMENT target (#PCDATA)>
<!ELEMENT ptime (#PCDATA)>
<!ELEMENT uptime (#PCDATA)>
<!ELEMENT tail (olinklist|linklist)>
]>
```

7.3 Discussion

In the present chapter, an outline of the implementation of a DPW system is discussed. The integrated environment of the Web, Java and CORBA provides a secure and reliable platform to implement a DPW system. Implementation of each tier of the three-tier architecture has been for DPW discussed in detail. The implementation of the client as a Java applet provides platform independence of the process, because a Java applet can run in any Java enabled browser. The independence is at the cost of performance. Since Java applets are interpreter based, it is slower than compiled programs. As a part of Java security policy an applet is not allowed to write in the local file system of the client. This may pose a problem in real life implementations, because local caching may be required. Of course, this can be handled by presently available solutions to address the problem, like Jamie's file system. So far as the implementation of the page cube is concerned, relational model is matured and mathematically well defined. Moreover, the relational query languages are powerful. But this approach may be possibly inefficient, because the representation of such irregular dynamic structure in flat relations is complex and retrieval of information may involve many join operations. Performance degradation and lack of openness is obvious. On the other hand, XML approach may be more efficient but it is too early to say, because XML tools are less available. Platform independence of data can be provided by XML, because an XML data file is an ASCII file with simple encoding. But XML is verbose, as a result increase in file size is a problem. It can be addressed in XML compression techniques. The product *XMill* is an example of an XML compressor. But since there is a possibility of a part being split, the content of a page needs to be marked up somehow. XML may provide a meaningful markup scheme for office documents. There are different XML query languages like Xpath, XML-QL, Quilt, Xquery etc. It is to be seen whether PSQL can be implemented on top of one of these existing query languages. Only a guideline of implementation is given here. During actual implementation, new issues may prop up and the guideline itself may need to be modified if necessary.

Chapter 8

Locus Standi of the Work

In the present Chapter we tried to find out where does the present research work stand, in the space populated by current academic as well as commercial solutions for office automation. Since, office automation is an active field of research for a long time, a number of work are available. The work related to document production workflow only are discussed here.

8.1 OMAIL

The Office Mail System (OMAIL) [8] , developed at Indian Institute of Technology, Kanpur, addresses some of the issues of MPMSD. The system is based on client-server model. In this system server is trusted. By making the server trusted, the responsibility of ensuring integrity, protection and failures can be handled by them. Content integrity of MPMSD is ensured by the server. The server computes a signature of the MPMSD using its secret key whenever a document is given out to reviewers and stores this signature along with the length of the document on which it is computed on, in a per user record. Later on, when the reviewer, after reviewing and adding comments on the original document, resubmits the document to the server to forward it to another reviewer, this stored signature is used to ensure content integrity of the document by recomputing a signature on the same length of the submitted document and comparing with the stored one. In this system, the document flows from one reviewer to the other via the trusted server, which acts as the arbiter. The reviewer is allowed to append his / her signed comment to the original MPMSD passed to him / her. This implies he/she is allowed to edit the document. But any modification to a previous parts, even by the creator of the parts, are detected by the server when the document is resubmitted to the server by comparing signatures as mentioned above. Similarly, order of parts and dropping of any part can obviously been detected. But the issue of reuse of parts in not addressed in this system. The loophole, in the system lies in allowing the reviewer to add comments directly to the original document and resubmitting it to the server. Moreover, with every review of the document, generation and verification of the signatures on the entire document up to the point of review, overloads the server and thereby slows down the performance. This can be avoided as shown in our schemes. Moreover, the context component is altogether absent in the discussion.

8.2 POLITeam

The decision of the German parliament to move the capital from Bonn to Berlin, increased the demand for a computer and telecommunication based support of ministerial process within and between dislocated government departments. In this context, four projects have been launched by the German research ministry in the framework of POLIKOM. One of the four projects is the POLITeam. The objective of the POLITeam project is the development of an adequate electronic support for workflows and the coordinated document and task processing in a ministry. A scenario of preparing a speech for a minister is presented in [27]. The request for a speech from a minister is issued by the minister's office to the manager of the department responsible for the speech topic. The request along with the background information are send to the manager in a circulation folder.The managers at the departmental, sub-departmental level acknowledge the receipt of the folder and additionally provides comments or advises. Then it goes to the unit level where the speech is prepared in a cooperative process between different people of the same unit or of different units. After creation of the draft version, it is sequentially processed by the managers of the unit, sub-department and department. Each manager reviews the speech and additionally annotates the document with her own comments. During the process we get the multiple versions of the document. Only the final version will be presented to the minister. Since the result of a ministerial procedure will have long lasting political consequences, the reconstruction of the document history and the authentication of the reviewer of the concerned version are the fundamental requirements of the system. We observe that the multi-version workflow document in this speech preparation process is a specific case of our more general MPMSD framework, where each part is a version of the speech. However, the security issues are not properly spelled out and addressed in the POLITeam solutions [27] as done in the present work.

8.3 Lotus Notes

Lotus Notes/Domino suite is a popular and commercially successful groupware product. The workflow component of the suite is basically the document production workflow. The production of MPMSD can be implemented in this software by designing a form with multiple sections. Each section of the form contains a comment of a reviewer. But the workflow is rigid in nature. It does not support flexibility in rerouting a document under review. That means, the channel of the review is determined a prori and a form with multiple sections is designed accordingly. A reviewer cannot reroute a particular case of a workflow during review unless the form is modified first in this effect. But flexible routing is a common phenomena in document production workflow in an office. For example, in the scenario discussed in Chapter 1, if the Directer feels that before taking a final decision of the travel plan, the comment of the registrar on the matter may be better to be taken for a particular case. Therefore, the registrar may be included as a reviewer in a certain DPW, but for some special cases only and not for all cases. Moreover, the Director may sent back the case to previous reviewer, that means to the finance officer, for further comment. Therefore, a priori fixed form-based solution may be suitable for routine industrial production workflow but it becomes a bottleneck to implement a flexible and dynamic document production workflow in an office. Our solution is naturally flexible. The context component is also not incorporated in Lotus Notes till date.

Lotus Notes implemented RSA crypto method for document security and for transport security it adopted standard web security like Secured Socket Layer(SSL). The protocols based on signed session key, discussed in Chapter 4 are more efficient than, the Lotus Notes protocols, because public-key schemes, like RSA, are much slower than simple symmetric key schemes. Moreover, security model of Lotus Notes has multi-level access control. The access levels are server, database, document, section and field levels. Conventional ACLs are attached to the objects at each level. Therefore, dynamic authorization is not directly implemented. Currently these authorization constraints have to be implemented as ad hoc application code [5].

8.4 Signcryption

To enhance confidentiality, the current standard approach is to sign a message and then to encrypt it with a randomly chosen encryption key. The encryption key would then be encrypted using a recipient's public key. This two step approach is called signature-then-encryption [42] and is popularly known as a digital envelope. The best example for this is Privacy Enhanced Mail (PEM), a standard for secure e-mail on the Internet. Signature generation and encryption consumes machine cycles, and also introduce 'expanded' bits to an original message. Hence the cost of a cryptographic operation on a message is typically measured in the message expansion rate and the computational time invested by both the sender and the recipient. In the signature-then-encryption, the cost for delivering a message in a secure and authentic way is essentially the sum of the cost for the digital signature and that for the encryption. Whether is it possible to transfer a message of arbitrary length in a secure and authentic way with an expense less than that required by signature-then-encryption. To answer this question Zheng has discovered a new cryptographic primitive termed as 'signcryption', which simultaneously fulfills both the functions of digital signature and public-key encryption in a logical single step, and that with a cost significantly smaller than that required by signature-then-encryption [42, 43]. The signcryption protocol have certain limitations on verifiability, Only the recipient can verify the signature. A third party cannot verify the origin of a signcrypted message independently. In a situation, like the one discussed in the present work, where one of the two parties involved in the communication is trusted, the digital signature based on signed session key, discussed in Chapter 4 is more efficient than Signcryption. Moreover, there there no limitation on varifiability by a third party.

8.5 Semi-Structured Data

Research on semistructured data was in a primary stage, when the present work was started. By now, semi-structured data has been studied from the database perspective in different works. Prominent among them are the *Lore* project at Stanford University (*http://www-db.stanford.edu/ lore*), and Sengupta's work, DocBase [35]. Both works try to build formalisms to deal with data which are structured with some form of markups, but the structure is irregular and incomplete, in contrast to relatonal databases. With the advent of XML as a powerful markup language, Lore project is implementing its Lore data base and its corresponding query language *lorel* in XML. *Lorel* uses path expressions as the key feature [31].

Docbase, a database environment for a structured document, structured with SGML is a major work for semi-structured data [35]. The work provides an in depth study on the formalisms for representing a SGML document as a database, where the DTD is the schema of the database. It also provides formal query languages to query SGML encoded data using simple path expressions. Even though this work deals with SGML document database, without any loss of generality it can be extended to XML databases as well. Both *Lore* and *DocBase* tries to find out a data element contained in a SGML document using some formalism similar to data retrieval, from a relation. Our work looks into a different aspect of document production and storage.

As a result, in our work, the object of retrieval is a page not an element in a page. The content of a page, in the present work, may be unstructured ASCII strings, or may be structured with, say XML. If we consider the content of a page is marked up with XML, our work may be complementary to these two works. For designing query languages for Page Cube, we derived some concepts from both the works.

8.6 Workflow Authorization

Workflow authorization is discussed in brief in Chapter 6. Role-Based Access Control(RBAC) is a standard tool for enforcing authorizations in security policies, but separation of duty constraint is not addressed by RBAC. Bertino el. al. [9] proposed separation of duty and proposed a language to express both the static and dynamic authorization constraints and also mechanisms to check constraint consistency. Atluri et.al. [5] proposed a model, Workflow Authorization Model(WAM), which provides synchronization of authorization flow with workflow. WAM also supports roles and separation of duty constraints. Enforcement of workflow authorization constraints using triggers and rules in active databases is studied in [13, 12]. In the present work, it is shown that how a simple extension to the page cube model takes care of basic authorization requirements for a DPW. Of course all the authorization constraints discussed in the literature are not taken care of in the page cube model. It remains as a future work.

8.7 Discussion

In the present chapter, some of the currently available solutions for office solutions are reviewed. But these solutions are not complete in nature. Our findings in the present work are not alternatives but complements the existing solutions. The output of the present work may be input to the future solutions for paper-less office. Even the future versions of the existing solutions, like Lotus Notes, Cabinet NG (*www.cabinetng.com* etc. may include the results of the present work.

Chapter 9

Conclusions

Secure production and storage of digital documents in an office environment is an important problem to be studied as a major step towards realization of paper-less offices. Security, production and storage are the three aspects of the problem. In the present work, these aspects are studied, different issues are identified in each aspect and solutions for them are provided.

The perimeter of the study is defined by three frameworks: production framework, storage framework and security framework. Documents in an office are termed as Multi-Part Multi-Signature Documents(MPMSD) and are produced as cases of a Document Production Workflow (DPW). Based on the frameworks, a conceptual three-tier architecture for a DPW is presented. It is found that to address the security issues, involvement of an in-line TTP is mandatory. A protocol for production of MPMSDs as cases of a DPW, with an arbiter as an in-line TTP, is presented. This protocol addresses the security issues identified in the problem. Since the arbiter is a TTP and also in-line, therefore it is always an intermediary between two communicating parties. A communication between a sender and a receiver via the arbiter is an aggregation of communication between the sender and the arbiter and the arbiter and the receiver. As a result the arbiter becomes one of the two communicating parties. Therefore, a digital signature scheme based on signed session key, suitable for such a scenario is proposed. This signature scheme is more efficient than RSA-based signature schemes used in existing solutions for office automation.

The context of creation of a document in an office is an important component of the organizational memory of an office. But hitherto this aspect has not been incorporated in any of the existing solutions for office automation. *In the present work a study on this important component of document* production is done. A multi-dimensional model, named Page Cube, for storage and retrieval of MPMSDs in an office with the contexts as the binding elements, is proposed. Documents are modelled in the scheme as pages or a tree of pages. A page is a point in the space defined by a set of orthogonal dimension hierarchies. The points are linked by bidirectional links forming dimensional graphs as well as inter dimensional graphs. Query languages to query pages from a Page Cube are also proposed. From the dimensional graph for the category dimension, the context can be produced. Algorithms to produce the dynamic context of a DPW as well as to produce a previous state of a context are discussed. The scheme to label the edges of the graphs with time stamps provide us the ability to recreate the previous states of a page. A model for dynamic authorization for DPW is also proposed as an extension of the Page Cube model. Implementation strategies of a DPW system, using state-of-the art technology have been provided. The implementation of a DPW system is meaningful if it is integrated as a subsystem of existing office automation software. Integration is easy in the CORBA framework. The output of the present work, which is academic in nature, may be input to future commercial paper-less office software. Here only a guideline of implementation is given. The scope of the study is limited to a single office, where the office is considered as a single trust domain. But in real life, document production may be across multiple domains. The study of such inter-domain DPWs with multiple arbiters remains for the future.

Bibliography

[1] ISO/IEC 3rd CD 13888-1. Information Technology - Security Techniques - Non-Repudiation, part 1: General Model. ISO/IEC JTC1/SC27 N1274. 1996.

[2] ISO/IEC 3rd CD 13888-2. Information Technology - Security Techniques - Non-Repudiation, part 2: Using Symmetric Encipherment Algorithms. ISO/IEC JTC1/SC27 N1276. 1996.

[3] ISO/IEC 3rd CD 13888-2. Information Technology - Security Techniques - Non-Repudiation, part 3: Using Asymmetric Techniques. ISO/IEC JTC1/SC27 N1379. 1996.

[4] V. Atluri and W. K. Huang. An Authorization Model for Workflows. In *Computer Security, ESORICS96, LNCS 1148*, pages 44–64. Springer Verlog, 1996.

[5] V. Atluri and W. K. Huang. A Petri Net Based Safety Analysis of Worflow Authorization Models. *Journal of Computer Security*, 8(2/3), 2000.

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

[7] Bannon and Schmidt. Cscw: Four characters in search of a context. In *Computer-Supported Cooperative Work : A Book of Readings.* Morgan Kaufmann, 1988.

[8] G. Barua and M. Bora. Integrity and Security Issues in Multisignature Document Mailing Systems. In *Computer Networks, Architectures and Applications (C-13)*, pages 193–198. Elsivier Science Publishers, 1993.

[9] E. Bertino, E. Ferrari, and V. Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. *ACM Transactions on Information System Security*, 2(1):65–104, 1999.

[10] M. Burrows, M. Abadi, and R. Needham. *A Logic of Authentication.* System Research Center, Digital Equipment Corporation, 1989.

[11] C. Zaniolo et.al., editor. *Advanced Database Systems.* Morgan Kaufmann, 1 edition, 1997.

[12] F. Casati, S. Castano, and M. Fugini. Enforcing Worflow Authorization Constraints Using Triggers. *Journal of Computer Security*, 6(4), 1999.

[13] F. Casati, S. Castano, and M. Fugini. Managing Worflow Authorization Constraints through Active Database Technology. *Journal on Information Systems Frontiers*, Special Issue on Workflow Automation, 2001.

[14] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. *SIGMOD RECORD*, 23(2):313–324, 1997.

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

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

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

[18] L. Harn and T. Keisler. New Schemes for Digital Multisignatures. *Electronic Letters*, 25(15):1002–1003, July 1989.

[19] K. Itakura and K. Nakamura. A Public-key Cryptosystem Suitable for Digital Multisignatures. *NEC J. Res. and Dev. 71*, pages 1–8, October 1983.

[20] S. Khoshafian and M. Buckiewicz. *Introduction to Groupware, Workflow, and Workgroup Computing.* John Wiley & Sons, New York, 1 edition, 1995.

[21] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. *Authentication in Distributed Systems: Theory and Practice.* System Research Center, Digital Equipment Corporation, 1992.

[22] A. J. MeneZes, P. V. Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1 edition, 1997.

[23] T. Mowbray and W. Ruh. *Inside CORBA Distributed Objects Standards and Applications.* Addision Wesley, 1 edition, 1997.

[24] T. Okamoto. A Digital Multisignature Scheme Using Bijective Public-key Cryptosystems. *ACM Transactions on Computer Systems*, 6(8):432–441, November 1988.

[25] R. Orfali, D. Harkey, and J. Edwards. *The Essential Client / Server Survival Guide.* John Wiley & Sons, New York, 2 edition, 1996.

[26] B. Pfitzmann. *Digital Signature Schemes, General Framework and Fail-Stop Signatures.* LNCS1100. Springer Verlog, 1996.

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

[28] A. K. Pujari. *Data Mining Technigues.* University Press (India) Ltd., 1 edition, 2001.

[29] R. Agrawal, A Gupta and S. Sarawagi. Modelling Multi-Dimensional Databases. In *Research Report: IBM Almaden Research Center, CA*, 1996.

[30] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

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

[32] G. M. S. Castano, M. Fugini and P. Samarati. *Database Security.* Addision Wesley, 1 edition, 1994.

[33] A. S. Sairam. *Design and Implementation of a Document Production Workflow.* M. Tech. Report, Indian Institute of Technology, Guwahati, 2000.

[34] B. Schneir. *Applied Cryptography : Protocols, Algorithms and Source Code in C.* John Wiley & Sons Inc., 2 edition, 1993.

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

[36] 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*, pages 45–52, 1999.

[37] 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, Thailand*, volume 3, pages 157–172, 2000.

[38] S. K. Sinha and G. Barua. PAGE CUBE: A Model for Storage and Retrieval of Documents Relevant to a Document Production Workflow in an Office. In *Proc. of the 7th Int. Conf. on Database Systems for Advanced Applications (DASFAA01), Hongkong, China*, pages 74–81. IEEE Computer Society Press, 2001.

[39] W. Stallings. *Network and Internetwork Security Principles and Practice.* Prentice Hall, New Jercey, 1 edition, 1995.

[40] 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*, pages 47–59, 2000.

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

[42] Y. Zheng. Digital Signcryption or How to Achieve Cost(Signature & Encryption) << Cost(Signature) + Cost(Encryption). In *CRYPTO'97*, LNCS 1294, pages 165–179. Springer-Verlag, 1997.

[43] Y. Zheng. How to Construct Efficient Signcryption Scheme on Elliptic Curves. *Information Processing Letters*, 68(5):165–179, December 1997.

[44] J. Zhou and D. Gollmann. Observations on Non-repudiation Protocols. In *ASIACRYPT'96*, LNCS 1163, pages 133–144. Springer-Verlag, 1996.