SaveMe: A system for Archiving Electronic Documents Using Messaging Groupware

Stefan Berchtold, Alexandros Biliris, Euthimios Panagos
AT&T Labs Research
Florham Park, NJ 07932
berchtol@ieee.org, {biliris, thimios}@research.att.com

ABSTRACT
Today, organizations deal with an ever-increasing number of documents that have to be archived because they are either related to their core business (e.g., product designs) or needed to meet corporate or legal retention requirements (e.g., vouchers). In this paper, we present the architecture and prototype implementation of SaveMe, a document archival system that is based on network-centric groupware such as Internet standards-based messaging systems. In SaveMe, the actions of archiving, retrieving, and classifying documents are similar to the actions of sending, retrieving, and classifying email into folders. SaveMe leverages existing messaging infrastructures - the one common denominator sitting on every computer is email - and, thus, it does not require individual users and IT personnel to learn a new technology. The resulting environment is not intrusive, easier to administer, and a lot easier to deploy.

Keywords
Archiving, messaging, groupware, Internet.

1 INTRODUCTION

Most of the organizations today keep information about their core business in an ever-increasing volume of electronic documents. Examples of such documents include product designs and documentation, corporate policies, expense reports, purchase orders, presentations, electronic messages, and data related to electronic commerce. Once these documents are approved, they can not be modified and they have to be stored for extended periods of time i.e., they have to be archived. The ability to manage, share, and control archived documents among collaborative workgroups, workflows, business partners, and across global networks is crucial in gaining advantage in today's competitive business environment.

In addition, organizations have substantial investments in electronic messaging technologies, which provide one of the most effective methods for communication, collaboration, and coordination among workers within decentralized organizations and across different companies. Messaging has evolved from a simple communications tool, which allows users to communicate with each other, to a powerful business communication infrastructure that supports collaborative computing, business process automation, electronic commerce, and distributed work environments. Messaging could even be used as the middleware for accessing and controlling shared resources. For example, printing a document could be achieved by emailing the document to a printer server that prints the document at the appropriate printer based on the available printers, the document type, and the sender's identity.

In this paper, we present the architecture and prototype implementation of SaveMe[1], a document archival system based on network-centric groupware such as Internet standards-based messaging. In particular, SaveMe uses a messaging server that supports SMTP (Simple Message Transfer Protocol [9]) and IMAP (Internet Message Access Protocol [5]). In SaveMe, archiving, retrieving, and classifying documents into meaningful collections is similar to sending email to recipients, retrieving messages from folders, and classifying messages into folder hierarchies. In the simplest scenario, if `saveme.com` is the archiving server's name, sending an email to `abc@saveme.com` causes the content of the email message to be archived in the `abc` mailbox. Archived documents are stored in jukeboxes of non-tamperable media such as Write Once Read Multiple (WORM) Compact Disks (CD), which provide high storage capacity, low cost compared to magnetic disks, random data access, and long-term stability.

SaveMe leverages existing messaging infrastructures and, thus, it does not require installation of new software on every desktop. In addition, individual users and IT personnel do not have to learn a new technology, resulting in substantial saving for an organization in terms of training and administration. The resulting environment is not intrusive, easier to administer, and a lot easier to deploy.

LEAVE BLANK THE LAST 2.5 cm (1") OF THE LEFT COLUMN ON THE FIRST PAGE FOR THE COPYRIGHT NOTICE.

[1] SaveMe: A System for Archiving Electronic Documents Using Messaging Groupware.

Archiving based on electronic messaging has advantages that are beyond the ease of use by individuals; certain application components would be able to exchange and archive messages in a seamless way. For example, enterprise resource planning systems track transactions and generate reports. In the past, most of these packages relied on the post office and fax machines to distribute reports. To speed the reporting process and streamline distribution cost, modern systems increasingly use Intranet and messaging infrastructures. A desired feature from such systems is the seamless archiving of the reports that are generated or transmitted. In our messaging-based archiving system, this can be achieved by simply CC-ing the documents to the archival email server.

In the remainder of the paper, we start by discussing archiving requirements and messaging standards. We then present the SaveMe naming scheme and the choices we were faced with in building SaveMe on top of a messaging infrastructure. Next, we discuss the database problems we had to address in order to provide high performance indexing for the archived documents and the way SaveMe manages physical storage and CDs. We then present the prototype implementation of SaveMe, we offer a comparison with related systems and, finally, we conclude our presentation.

2 ARCHIVING DOCUMENTS

An archiving system must provide economical storage for large volumes of non-modifiable documents for extended time periods as well as quick, easy and controlled access to documents, both navigationally and declaratively (by words and phrases, or by attributes like author and project).

2.1 Document Classification

Given the vast amount of stored documents in an archiving system, document classification based on hierarchically organized collections is important for two reasons: First, it provides an easy and intuitive way to classify documents into collections and sub-collections - this helps focus the search into smaller domains. Secondly, it facilitates easy navigation through the collection of documents, especially when the criteria used to classify the documents in the first place (i.e., the schema of the hierarchy) are not known.

There are many ways to organize documents into hierarchies. An example of such a hierarchy might be the organizational hierarchy of the corporation: business units, centers, and departments followed by the various document types. Another might be a first classification based on the year the document was archived, followed by document types, business units, centers, and departments. Note that a hierarchy may change over time - think of departments being renamed or split (new levels may have to be introduced in the hierarchy) and centers being merged (levels in the hierarchy may have to be merged). Also, hierarchies need not necessarily be stored (materialized). They could be computed views over the set of documents.

Navigational access is certainly a convenient way to search for a document within small collections. However, for large collections the capability for attribute-based and full-text searches is needed. Attribute-based searching locates documents based on values of attributes associated with documents. Full-text searching finds documents based on their content. Both kinds of searches can be performed on the entire hierarchy of folders or on some particular sub-tree.

2.2 Access Control

Archived documents need to be protected against unauthorized usage. User *authentication* prevents unauthorized usage of a system and its resources, and it can be implemented using a *user-name/password* scheme. Once a user has been successfully authenticated, the archival system has to enforce *access control* restrictions on the documents and the collections it manages.

Since hierarchies are used for document classification, it is appealing to directly use them for enforcing access restrictions in a way similar to the one employed by operating systems on directories and files. This organization also simplifies the assignment of default access rights since each newly created object can inherit its parent's permissions; also, changes in the permissions at some level in the hierarchy can selectively be propagated down the hierarchy.

Another alternative is a *role-based* access control. Roles are sets of permissions on objects. A user might have many roles. Role-based access control separates the assignment of access rights and permissions to roles from the assignment of users to roles. Therefore, the security structure can be kept essentially static, even though users may change frequently over time.

3 INTERNET-BASED MESSAGING PROTOCOLS

The design of the SMTP protocol, used for transferring messages, is based on the following model of communication. In response to a user mail request, the sender-SMTP establishes a two-way transmission channel to a receiver-SMTP, which may be either the ultimate destination or an intermediate. SMTP commands are generated by the sender-SMTP and sent to the receiver-SMTP, which generates replies in response to the commands. The sender-SMTP negotiates with the receiver-SMTP the recipient list of the message, one recipient at a time. If the receiver-SMTP can accept mail for a recipient it responds with an OK reply; if not, it rejects the recipient (but not the whole mail transaction). When the recipients have been negotiated, the sender-SMTP sends the mail data.

According to the SMTP specification [9], email is sent to *mailboxes*. The standard mailbox naming convention is `user@domain`, a string that consist of the user to whom mail is to be sent and a host specification. Although the user specification part of an address is allowed to contain several components (separated by dots, slashes, or other characters), existing implementations do not map these components to folders in some folder hierarchy. Consequently, if there are subfolders under a user's mailbox, these folders cannot be explicitly addressed using SMTP.

The IMAP protocol is a method for accessing folders and mail messages stored with an email server. In particular,

the protocol includes operations for creating, renaming and deleting mailboxes, checking for new messages, permanently removing messages, setting various flags associated with messages, selectively fetching portions of messages and searching existing messages. In IMAP, the interpretation of mailbox names is implementation specific. However, the case-insensitive mailbox name *Inbox* is used to mean the primary mailbox of a user on a particular server. IMAP supports hierarchies of mailbox names. Such hierarchies must be left to right, using the same hierarchy separator at all levels.

While existing IMAP implementations provide user authentication, they provide limited or no access control primitives at all. Once a user is authenticated and her mailbox is selected, no access checks are performed on the messages and folders under her mailbox. Extensions to the original IMAP standard to include access control lists are currently under consideration by the Internet community. Under the IMAP ACL extension proposal [11], access rights may be associated with folders. These access rights specify the particular IMAP operations a given user can issue with respect to a particular mailbox. The proposed access rights allow users to: (1) see mailboxes but not operate on them; (2) operate on the messages stored in a mailbox, including searching and copying messages to the end of the mailbox; (3) append data as messages to the end of a mailbox; (4) create sub-mailboxes, and (5) obtain administrative privileges.

Finally, the Lightweight Directory Access Protocol (LDAP) [7] is used to provide efficient access to an X.500 directory. LDAP is used during SMTP and IMAP sessions. In the former case, an LDAP directory is used for locating the recipient(s) or messages and their mailboxes. In the latter case, an LDAP directory is used for authenticating the users logging onto the messaging system and for locating the mailboxes of these users. In existing messaging systems, new messages are delivered to the Inbox of a recipient, and the particular folder hierarchy under the Inbox is not represented in the LDAP hierarchy; rather, the messaging server maintains it.

4 THE SaveMe ARCHITECTURE

SaveMe is built on top of a messaging server that supports SMTP and IMAP. In SaveMe, archiving, retrieving, and classifying documents into meaningful collections is similar to sending email to recipients, retrieving messages from folders, and classifying messages into folder hierarchies. We first present the naming scheme used in SaveMe and then discuss alternatives for implementing this scheme using a messaging server.

4.1 Naming

SaveMe employs a hierarchical naming scheme that is used for both identifying collections of archived documents and providing access paths to them. Documents are archived by storing them into folder hierarchies belonging to email recipients. For example, a document sent to the address `A/B/C@saveme.com`, where `saveme.com` is a SaveMe server, will be archived in folder `C`, which is under folder `B`, in the hierarchy rooted at `A`. The hierarchy structure is maintained in a directory.

In a distributed system, multiple SaveMe servers can manage the folder hierarchy. Servers can be added, removed and reconfigured to accommodate load conditions and business considerations, without breaking existing applications and user habits. This is achieved by allowing the naming components of a folder hierarchy to be resolved by different servers, supporting local autonomy. In particular, a name is logically split into a left part (*prefix*) and a right part (*remainder*). A directory service maintains information that maps a prefix to a SaveMe server responsible for managing the remainder. We should note that the two name parts may not consist of a fixed number of components neither do they have to stay the same over time. Given a name, its prefix is identified by the directory service at the time the lookup is performed. A subsequent resolution for the same name may return a different prefix if the SaveMe servers have been reconfigured.

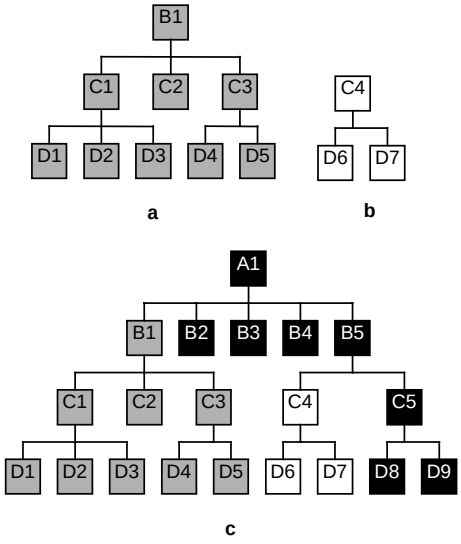

Figure 1. Two SaveMe servers manage the hierarchies shown in **a** and **b.** These hierarchies are then conceptually merged with a third hierarchy to form the global hierarchy shown in **c**

Figure 1a shows a document hierarchy managed by a SaveMe server on behalf of an organization. This hierarchy is rooted at B1, and the levels below the root could represent sub-units of B1 such as centers, departments, employees or projects that have been undertaken in B1 at some point in time. Figure 1b shows a hierarchy for another organization rooted at C4 and managed by a second SaveMe server. Assuming that at some point in time an organization higher up than both B1 and C4 decides to employ the archiving service for its own needs and for some additional sub-units. The resulting configuration is shown in Figure 1c. The first two servers still operate in an autonomous fashion for the hierarchies they manage. However, each one of them is able to resolve names that correspond to the entire hierarchy. The three paths A1, A1/B1 and A1/B5/C4 are the possible prefixes that will be used for name resolution. Given a name, the longest prefix from these three that matches a portion of the left part of

the name is the prefix of that name. For example, the prefix of A1/B5/C5 is A1 while the prefix of A1/B5/C4/D7 is A1/B5/C4.

The global name resolution can be implemented as follows. The mapping between prefixes and SaveMe servers is kept in a global directory that is updated every time servers are added to or removed from the system. Changes to the internal structure of each server do not need to propagate to the global directory as long as the path of their root to the root of the global hierarchy remains the same. Figure 2a shows the global directory for the example shown in Figure 1c. Every node in the hierarchy may denote a prefix (i.e., the server that maintains the hierarchy rooted at that node). However, some nodes, such as B5 in Figure 2a, are simply intermediate paths to prefixes.

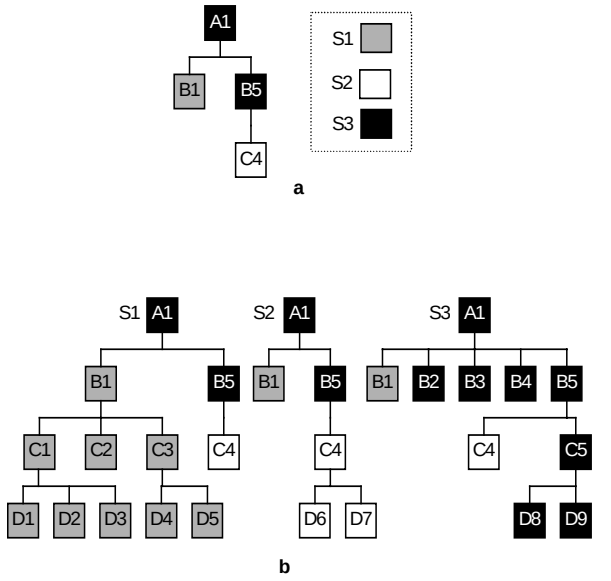

Figure 2. Alternative name resolution implementations

An alternative implementation would require that each server kept track of the prefixes for all other servers. Figure **2**b shows the directories S1, S2, and S3 where each one includes replicated information for the roots of the other two directories. This scheme avoids a lookup to a, possibly remote, directory service in order to find the prefix of a name. As with the global directory scheme, changes to the internal structure of each directory need not propagate to the other directories as long as the path of their root to the root of the global hierarchy remains the same. The replicated scheme is better than the one that utilizes a global directory because the expected number of archiving servers that participate in a global archiving system is small (typically in the range of tens or even hundreds of servers for big organizations).

4.2 Interface Architecture Alternatives

While electronic messaging systems provide numerous facilities for sending, organizing and retrieving messages, some of the implementations have shortcomings when it comes to supporting two features that are of paramount importance for electronic archiving: (a) addressable folders anywhere in a folder hierarchy and (b) enforcing access restrictions on individual folders. In particular, IMAP does not provide sufficient access control mechanisms, and most existing SMTP implementations do not support folder addressability (see Section 3 for details). In this section, we discuss four design alternatives based on the functionality provided by the messaging system to overcome these restrictions. The first three alternatives assume that the messaging system does not support folder addressability and access restrictions.

4.2.1 The Adjunct Approach

Here, users interact directly with the messaging system. SaveMe manages the hierarchical relationships between folders and enforces access control restrictions explicitly.

Since the messaging system does not support folder addressability, the following alternatives can be used for archiving documents into hierarchical folders:

- Every folder in the hierarchy corresponds to a (virtual) user. For example, we could define a user mailbox named a/b/c to store the folder a/b/c, another mailbox to store a/b/d, etc. ;
- All documents are sent to a single predefined mailbox, and the user specifies the actual folder path in the *subject* or in a user-defined header attribute, e.g., *X-folder*.

In both cases, SaveMe monitors the appropriate mailboxes, extracts the folder path from the message/folder name and forwards the message, if needed, to the server responsible for managing this path.

Regarding IMAP-based navigation, email clients navigate the folder hierarchies that are dynamically constructed by SaveMe in response to user queries. Here, users send queries to SaveMe (using a Web browser), and SaveMe creates folder hierarchies[2] for these users to store the documents that satisfy the specified selection criteria and any access restrictions that may apply.

4.2.2 The Proxy-with-Mail-Server Approach

Here, a proxy is placed between the users and the messaging server, as shown Figure 3. During document archiving, an email message containing the document is sent to an address denoting a folder hierarchy. During the recipient negotiation part of SMTP, the proxy looks up the hierarchy specified in the address of the message in a directory to determine its validity and locate the SaveMe server responsible for this hierarchy. If the server is not the local one, an SMTP session is established with the appropriate remote proxy. Otherwise, the proxy executes the following steps. First, it communicates with SaveMe to get an appropriately formatted URL to use as the message body, which will be stored with the messaging server. Next, it accepts the body of the message and stores it in the cache used for the CD jukebox. Finally, it establishes an IMAP session with the IMAP component of the messaging server

[2] Regarding the folders that are created, there are several policies that may be used, e.g., provide each user with a mailbox and store the answer set in the mailbox.

and stores the modified message in the appropriate folder hierarchy.

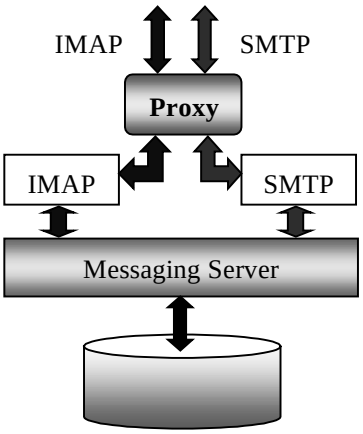

Figure 3. The SMTP and IMAP proxy architecture alternative

For accessing archived document, email clients establish an IMAP session with the local SaveMe proxy. The proxy is responsible for enforcing any role-based access control restrictions that may apply to both documents and folders. In addition, the proxy may access folder hierarchies stored with remote servers, providing a unified view of the entire document archival space. The proxy relays the IMAP commands it receives from an email client to the IMAP component of the messaging system. The IMAP proxy enforces access control when the messaging server's IMAP server returns folder names and messages as responses to commands issued by the email client. The IMAP proxy removes from the list of returned documents the documents that are not accessible by the given user.

4.2.3 The Proxy-with-no-Mail-Server Approach

As in the previous scheme, the proxy implements the SMTP and IMAP protocols. However, in this approach the messaging system is bypassed - SaveMe communicates directly with email clients and their commands are processed internally by SaveMe.

4.2.4 The Shared-Folders Approach

The last alternative, which we employ in our prototype implementation described in Section 6, is based on the assumption that the underlying messaging server supports folder addressability and enforcement of access rights through *public (*a.k.a. *shared) folders.* Several messaging and groupware products, including Microsoft Exchange, Eudora WorldMail and HP OpenMail support public folders. Usually, the implementation of public folders offers a full-fledged access control mechanism on a per folder basis. Therefore, SaveMe can utilize the public folder mechanism of the underlying messaging system for implementing its naming scheme and enforcing access restrictions. We should note, however, that shared folders do not correspond to any internet standard and, consequently, the messaging component of SaveMe may have to be chosen from a small list of existing systems that offer this functionality.

5 DATABASE PROBLEMS AND SOLUTIONS

In a typical archiving system used by an enterprise, we have to deal with huge amounts of data. If one for instance assumes that each employee of AT&T creates and archives only a single document per working day, this leads to about 30 Million documents per year. To handle such an amount of data we need database technology.

When a user archives a document, she may also provide additional information to SaveMe including, for example, a classification of the document to a node in the document hierarchy, who authored the document, what project is the document assigned to, or when will the document expire. This information can be either mandatory or optional. The detailed mechanism how a user provides this information will be described in Section 6. In general, each document has a specific document type. For instance, we might have *voucher*s, *purchase-order*s and *time-report*s as document types. Documents of a certain document type can occur at arbitrary locations in the hierarchy. For example, a voucher or time-report might be classified to an arbitrary project, department, or person. Thus, classification and document types are orthogonal. From a database perspective, this means that for each document type, we can identify a set of attributes that a user has to provide in order to archive documents of this type. In case of a purchase-order this may be "po_number", "po_date", and "po_amount". We can assume that the majority of the attributes of different document types are disjoint. However, there are attributes shared by all document types. Examples include "person_who_archived", "archive_date", or "department".

Thus, we have to address the following database problem: we are given a large set of documents with attached attributes, and each document should be classified in a hierarchy. We need to organize the documents, both logically and physically, so that operations such as retrieval and deletions of documents are supported efficiently. We do not consider updates because update operations are not allowed in the legal domain of archiving. Since the most frequently performed and time-critical operation in an archiving system is query processing, we use database technology for query processing, namely, schemas and indexes. However, since the number of archived documents is assumed to be very large, indexes require a significant space and maintenance overhead. Therefore, we have to choose our database schema with care.

Typically, users retrieve documents by either interactively navigating the document hierarchy or by declaratively specifying document selection criteria. The user interfaces and the according processes are described in the next chapter. We regard the following queries as typical queries in an archiving context:

Q1: "Give me all documents that have been archived by "Alexandros Biliris", "Thimios Panagos", or "Stefan Berchtold" some time in "April, 98".

Q2: "Give me all documents that contain the phrase "Indexing in SaveMe".

Q3: "Give me all documents which have been classified "ATT/Research/Databases".

Q3: "Give me all documents that have been archived in "April, 1998" and classified "ATT/Research" of which the project number is "SaveMe1" that contain the phrase "ha, ha, hi".

In this section, we address the following issues. First, we show how the problem maps to a relational database design and what additional functionality one has to provide. Second, we address the problem of indexing a hierarchy of objects and how relational technology can be tuned to support this. Finally, we discuss problems and solutions related to the physical organization of documents. The main problem is that for legal reasons, documents have to be stored on durable media such as optical disks or CDs rather than magnetic disks. There are several technical challenges one faces when dealing with devices of this type such as large seek times, write-once-read-multiple (WORM), or the fact that disks can only be written efficiently in one single pass.

5.1 Efficient Declarative Retrieval of Documents

We use a relational database system to manage the attributes associated with the archived documents, including a pointer to their physical location. The actual documents are organized in a file system on the CDs, independently from the database system. The reason for this is that the design of commercial database systems is tightly bound to the properties of magnetic disks and, thus, CDs or even WORMs are not supported appropriately. In addition, SaveMe has to have control over the transfer of documents to CDs and their clustering.

Also, we assume to have available a full-text search engine such as Verity [12]. These search engines take a set of documents, extract the text out of each document and build appropriate indexes. Then, one can ask queries such as: "Give me all documents that contain the word SaveMe". As a result, the engine returns a set of pointers to actual documents. We do not go into further details of information retrieval in this paper. Rather, we assume to have such a search engine and treat it as a black box.

5.1.1 Database design

In SaveMe, a document type is a collection of attributes that describe some information about a group of documents. Obviously, given this definition, there is a strong relationship between a database schema and the document types. However, there are some technical challenges. First, attributes can be either mandatory or optional. Therefore, some of the attribute values may be unknown to the system. Second, the semantics of a particular attribute is unknown to the system. Furthermore, for practical reasons, we cannot even enforce that attributes are named uniquely. Thus, an attribute "x" might appear in document type "1" and in document type "2" having a different meaning. Third, the number of document types might be in the order of thousands whereas the total number of attributes might be in the order of tens of thousands for a large system.

To avoid problems with the semantics of a query specified by the user, we have two mechanisms. First, we provide a user interface (see next chapter) that restricts the user to meaningful queries. Second, if we get a query that contains attributes "x" and "y", we only return documents that are of a type that includes attributes "x" and "y". Third, when an attribute value "x" appears in a query and "x" is not present in a given document, although "x" is in the schema of the this document type, the document is not included in the result. This can occur when "x" is an optional attribute.

In SaveMe, a document is uniquely identified with a document ID, which is attached to a document when the document arrives in the system. This is the primary key of the document. For simplification, we only allow a relatively small number of document types (e.g. up to 1000), which all are pre-defined by the system administrator. He is responsible for maintaining the set of document types and for assuring that all information relevant for searching is present in the system. He is also responsible for keeping the number of document types reasonably small.

Because we assume a large number of document attributes, it is not feasible to use a single table for all document types. This would lead to a very sparse table since most of the attributes are unknown for most document types. Therefore, we assign each document type to a single table that includes the document type specific attributes. Additionally, we keep a global table that contains the attributes shared by all documents. Each table includes the document ID as a primary key. Indexes may be placed on any attribute according to the query mix and the size of the table. If a new document type is introduced into the system, the system administrator must create a corresponding table in the relational database. As the number of document types is proportional to the number of tables, this number should be reasonably small. Figure 4 shows an example for the SaveMe database design.

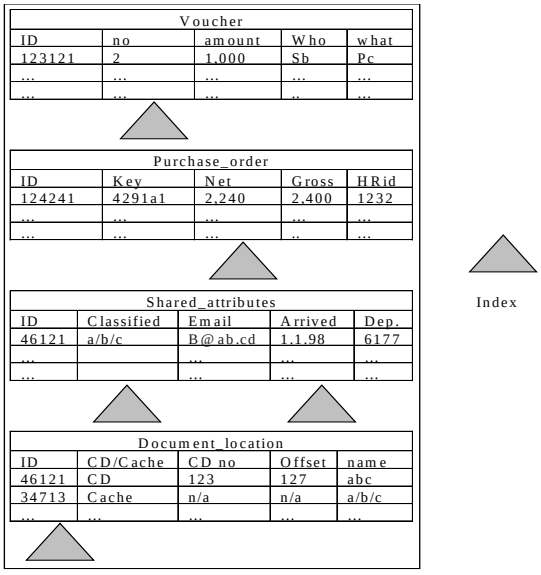

Figure 4. SaveMe relational database design

5.1.2 Query Processing

Queries are generally issued via the interactive query interface. The query interface performs some sanity checks such as formats of numbers or if a string represents a correct date. The query is transferred to the system as a set

of attributes together with query ranges. Additionally, the query processor gets the information to what part of the hierarchy the query is restricted and any search-strings that must be contained in the resulting documents. In order to process a query that arrives in the query processor, we have to consider all three parts of the query specification: selection of attributes, selection based on the classification in the hierarchy, and full-text search. Thus, we split the query into the three parts and handle them separately. Finally, we merge the results.

We first perform the full-text search portion of the query. As a result, we get a set of document IDs that point to documents satisfying the query condition (with respect to the full-text search). The selection based on the hierarchy is more sophisticated. As described in the next section, we map each node of the hierarchy to a string such that all nodes in a sub-tree share a common prefix. Thus, the query specification on a hierarchy can be transformed in a range query on that string. Therefore, we simply include the string as an additional attribute "Classified" in the table of the shared attributes. For query processing, we now are able to treat queries based on the hierarchy as range queries (prefix queries on strings can be seen as range queries) on the attribute "Classified".

As a next step, we determine the set of attributes that are involved in the query. If the query specification includes the hierarchy, we add the attribute "Classified" to this set. Then, we remove all attributes from the set that are contained in the *Shared_attributes* table. Finally, we determine all tables that include all attributes remaining in the set and generate a single SQL statement querying all affected tables. As a result of such query, we again get a set of document IDs. In the last step, we merge this set of document IDs with the set resulting from the full-text search and deliver the final result of the query back to the user. This merge step is equivalent to the intersection of two sets of document IDs. This can be done efficiently by sorting or hashing. The user has the ability to either look-up all the actual documents or to look-up only single documents. To locate a document given a document ID, we use the information stored in the *Document_location* table.

Note that an appropriate generation of indexes in the relational database system is required as in any database application. The creation of the optimal set of indexes for the given query mix and database population is task of the system administrator as mentioned above.

5.1.3 Efficient Searching in Hierarchies

Several ways to index objects that are organized in a hierarchy have been discussed in the literature [8,10]. Most of these proposals focus on a small and static hierarchy where a user asks hierarchically restricted attribute queries. The most common technique is to map the position of an object in the hierarchy into a number such that all nodes in a sub-tree are assigned a number from a known disjoint interval. Then, search on a hierarchy is reduced to a range query on this number. However, none of these proposals works on a dynamic hierarchy. Thus, we propose the following simple mapping to strings instead of numbers.

Suppose that each node in the hierarchy is assigned a label such that sibling nodes have different labels. If we want to denote the position of a node *N* in the hierarchy, we simply concatenate the labels on the path from the root node to *N* adding a "/" to each label. Thus, we get a string of the form "a/b/…/x". If one wants to select all documents in a certain sub-tree (rooted in a node S having a label "s"), one simply has to query the database for all documents that have assigned a position "a/b/…/s/*". This translates to a range ["a/b/…/s", "a/b/…/t"). If a document arrives in the system, the user will classify it. This means, it is assigned to a node in the hierarchy. If we insert the document's information in the database, we compute the according string defining the position of the document in the hierarchy as described above and insert this attribute ("Classified") with the other attributes into the according tables. In order to process a query based on the hierarchy, we first have to determine the sub-tree to which the query is restricted. Then, we determine the string defining the position of the root node of this sub-tree in the hierarchy and query the database with a range query as described above.

Note that the strings that identify a position in the hierarchy may become very long. This however is not a particular problem because modern relational database systems can handle large strings easily and even the implemented index technology (usually prefix B+-trees) is not severely affected.

As an extension of our technique one might want to use multidimensional index technology in order to answer combined queries more efficiently. This however corresponds to the problem of having a multidimensional index structure storing both string and numerical attributes. So far no such index structure has been proposed and we think that this is a non-trivial task, which exceeds the focus of the work presented in this paper.

5.2 Physical Storage

Magnetic disks are not an economical option for storing a large number of documents over a long period of time. SaveMe uses a CD jukebox as the underlying physical storage device. A good physical organization of the data on the optical storage system is essential for an efficient document retrieval (search time and throughput), CD storage utilization, and bulk-drop of expired documents.

Usually, optical devices contain between 100 and 1000 media (e.g. CDs), each storing about 650 Mbytes, keeping a total amount between 50 Gbytes and a few TerraBytes. A typical CD Jukebox has four reading and one writing device and a single robot arm loading and unloading the devices. In contrast to magnetic disks, the time to access a new CD, which is not currently mounted in one of the drives, might be very long (in the order of a minute). Thus, each single retrieval process of documents should be restricted to as few CDs as possible.

For the SaveMe prototype system, we have the following storage hardware available: main memory in the order of 1 Gbyte, secondary storage on magnetic disks in the order of 10 Gbytes and optical storage (CD Worm) in the order of 100 Gbytes. We use the secondary storage to store the database including all indexes and as a cache for the optical

disks. As CDs must be written in a single pass, we have to cache 650 Mbytes on magnetic disc before burning a CD. Thus, the available cache restricts the number of CDs that can be written simultaneously. (Given the above amounts of storage, this number is in the order of 15 disks.) Note that, although most CD writers technically allow for more than one write-process on a single CD, each additional write costs a large space overhead. Furthermore, CDs written in one step tend to be more stable over time. Therefore, it is advisable not to make use of this feature.

In order to define a good strategy to assign documents to CDs, one has to consider the following points:

1. Documents expire after an arbitrary period of time such as 1 year, 5 years, 10 years, or never. Sometimes expiration must force document deletion. For example, financial organizations need to hold credit card data for seven years, but legally they are not allowed to retain it after that time.
2. We cannot assume that archiving of all documents expiring in a certain year can be done in a short period of time. Rather, documents expiring in 2007 will probably be archived during 1997, 1998 and 1999.
3. In order to avoid the system to be filled with expired documents and to maintain high storage utilization, we have to provide a mechanism that allows us to discard entire CDs if all the documents stored on the CDs have expired. We call this deletion of a large set of expired documents "bulk-drop".

If one looks at all these technical restrictions, there are two different criteria to optimize. On the one hand, we intend to cluster documents on CDs according to their expiration date and, on the other hand, we want to cluster documents that are frequently retrieved simultaneously. Furthermore, the query mix is unknown in advance and will definitely change over the years such that it is unknown a priori which documents will be requested together. Therefore, we use the following solution. The cache is primarily organized as a set of documents. Each document D_i in the cache is assigned a value v_i that can be used to cluster the document to disk. This value typically is the expiration date of the document. However, it could also be a value such as project number or even a combination of both. The documents in the cache are sorted[3] according to these values. If the cache overflows, we have to select a subset of the documents in the cache of size 650 Mbytes and write a CD. The following algorithm selects the shortest subset (with respect to the values v_i) containing no more than 650 Mbytes by only making a single pass through the documents stored in the cache.

```
ShortestSubset (documents D, values V, int N)  {
    current_size = 0; start = 0; end = 0;

    while ( current_size <= 650 )     // Find initial candidate
        current_size += size(D_end);
        end++;
    shortest_start = start;
    shortest_end  = end - 1;

    while (  end < N  )                    // Find shorter  subsets
        current_size -= size(D_start);
        start++;
        while ( current_size <= 650 )
            current_size += size(D_end);
            end++;
        if ( (V_end - 1 - V_start)  < (V_shortest_end - V_shortest_start) )
            shortest_start = start;
            shortest_end  = end - 1;
    return (shortest_start, shortest_end);
}
```

For clarity of the presentation, we omitted some necessary checks for array boundaries in the algorithm shown above. The algorithm is a greedy algorithm and does not necessarily lead to a globally optimal solution. However, given the various technical restrictions, an optimal solution is far from being feasible. Thus, the algorithm gives us a sufficiently good solution. Furthermore, at any point in time, we are able to reorganize a subset of documents by collecting the appropriate documents from CDs and writing new CDs. For this process of reorganizing *n* CDs, only a single 650 Mbyte cache is required.

6 IMPLEMENTATION EXPERIENCE

In terms of implementation, our efforts have been focused on prototyping SaveMe on top of two messaging systems: Hermod, a messaging system being developed at AT&T Labs [2], and Microsoft Exchange Version 5.5 [6]. Here, we describe our implementation on top of Exchange. Briefly, Exchange is an IMAP compliant server and, thus, it is accessible from any IMAP compliant mail client. It supports addressable public folders on which a rich set of access permissions can be defined. In addition, the product includes LDAP support and server-side scripting. The latter allows running programs (written in JavaScript, VBScript, C++, etc.) on the server as a reaction to events occurring in public folders. The supported events include timer events and actions like posting, editing, or deleting a message.

6.1 Archiving of Documents in SaveMe

As we discussed in the previous chapter, there are several attributes associated with archived documents. SaveMe derives some of these attributes automatically, such as the date and time the document was archived, who archived it, or the size of the document. To archive a document that does not require attributes other than those that can be derived automatically, one simply has to send the document as an attachment to an email addressed to the public folder that should contain the document. This can be done by specifying the folder name in the any of the `To:`, `CC:`, or `BCC:` fields of the e-mail.

For other types of documents, users need to specify values for some pre-determined attributes. For example, an expense report must specify the employee submitting the report, a dollar amount, a manager to authorize the expense, etc. This is accomplished with the use of electronic forms – pages containing the fields that must be filled out for each document type before the document submission. There is one such page for each document type. These forms are

[3] In our implementation, we use a heap to maintain the sorted set.

usually stored as regular email messages in the SaveMe server. Therefore, in order to archive a document of a particular type, the user replies to the appropriate message. She fills out the form and attaches the document she wants to archive to the message. Then, she sends the message to the desired public folder.

All folders in the SaveMe server are being monitored by the event service. The deposit of a message in a folder triggers the `Folder::OnMessageCreated` event. As a result of the triggered event, a server-side script associated with this event is invoked and is passed the folder identification and the identifier of the newly posted message. The script first collects the properties of the message (date, sender, etc.) that can be automatically derived and then assigns a unique id to the document. Then, it parses the body of the message and, if the body of the message contains a form, it extracts all information contained in the form and, finally, it populates the appropriate database tables for this document. In addition, the script also checks the completeness and correctness of the submitted form by evaluating the information contained in the form and verifying that all the required ones are present. If any are missing or some constraints are violated, it returns the message to the sender and removes the message from the mail server.

Next, the script removes the message body, including the attached document, from the email server. The document itself is stored on a separate disk cache waiting to be written out to a CD. Another body containing some information about the document, together with a URL pointing to a CGI script that is capable of loading the actual document, replace the original body. Information about the exact physical location of the document (on disk or CD) is maintained in the *Document_location* table in the SQL database used in SaveMe. A user can click on the URL and retrieve the document at any point in time regardless of the physical location of the document. As a next step, the script checks if the cache is overflowing. If this is case, it invokes the algorithm described in the previous chapter to write a subset of the cache to a CD.

6.2 Retrieving of Documents in SaveMe

To retrieve a document or a set of documents that has been archived in SaveMe, a user has two choices: she can either navigate through the document hierarchy and manually select individual documents, or she can declaratively specify a set of documents. In the latter case, SaveMe automatically retrieves the desired documents and delivers the documents to either a folder in the mail server or directly to the Web browser when the Web interface is used.

In order to navigate through the document hierarchy, the user only has to connect to the SaveMe server and browse through the public folders with an IMAP compliant mail client (such as the ones offered by Netscape and Microsoft). After having found the right documents, the user can retrieve a document by clicking on the URL viewed in the message body. When the URL is de-referenced, a CGI scrip looks up the *Document_location* table to determine the physical location of the message and the message is then returned to the user.

For declarative specification of the desired document(s), a dynamic web interface is used. The search process is divided into two steps: First, the right document type to be retrieved is chosen[4]. Second, SaveMe creates an HTML page that offers an appropriate query interface for this particular type. The user then fills out the fields for the attributes that have been defined for this document type. Also, an option for full text search is offered. When the user clicks on the submit button, the appropriate CGI script is invoked. The entered values are checked for correct types and other constrains. Then, the query is processed as described in the previous chapter. As a result of query processing, an HTML page containing the URLs of the actual documents meeting the query conditions is created. In order to transfer the documents to the client computer, the user has to click these URLs.

6.3 Administration

There are four major administrative tasks that have to be performed in order to maintain a SaveMe server: adding and maintaining document types, maintaining the document hierarchy, managing access rights, and database administration. In order to add a new document type, the system administrator has to store the form for this type with the email server. She then has to add the appropriate tables in the database and implement appropriate scripts to parse the form and to make a sanity check on the information delivered by the user. The maintenance of the document hierarchy and management of access rights can be done through the tools that Exchange provides to manage the email server. The main task with respect to database administration is to add and delete appropriate indexes to the database.

7 RELATED WORK

This section summarizes some of the available document management and electronic archiving systems that are related to SaveMe.

Aegis StarView [1] is a document archiving system based on proprietary storage and search architectures. In StarView, documents can be archived using email and up to seven user-defined attributes can be attached to a document. StarView offers a restricted declarative search capability for retrieving documents, using the storage date as the primary retrieval category. Search queries can be sent via email, and the matched documents can be delivered to the user's mailbox. StarView does not expose the document structure to the users and, thus, it does not support navigation. It is a client-server system that requires the installation of client software on each user's computer.

DocuLive [12], a Siemens Nixdorf document management system, supports document classification in a folder hierarchy, access control, searching, and distribution. Access rights can be assigned to documents and folders during creation, and they are based on organizational

[4] One choice here is "all document types" to allow for document type independent queries.

structure and roles as well as ad-hoc target groups. Search queries can use either specific templates for the various document attributes or the Verity full-text search engine. In addition, users can customize the document attributes and the order in which the answer set is going to be displayed. Search results can be stored in mailboxes. However, unlike SaveMe, the messaging system used in DocuLive is a separate entity. Consequently, documents are copied from DocuLive to the messaging system before sent and, on the other hand, incoming documents have to be dragged from the email client and dropped in the DocuLive desktop.

The Domino.Doc [4] document management system offers a common set of tools across a comprehensive range of content. Documents are stored in file cabinets and binders (binders are contained in cabinets). File cabinets are built as database files and they can be personal or shared. Each file cabinet has an ACL associated with it. Members of an ACL can be individual users, groups of users, and even organizational roles. Unlike SaveMe, Domino.Doc does not provide e-mail client interface to document retrieval. Besides the Web browser, Lotus browser plug-ins and the Lotus desktop client can be used for retrieving documents. Similar to SaveMe, searching can be done by either traversing the file cabinet/binder hierarchies or, by issuing attribute-based or full-text search queries. Finally, archiving is supported in Domino.Doc via user-specified migration rules. Documents are moved to tapes or optical jukeboxes and placeholders are left behind in the database.

Documentum [3] provides services for storing, accessing and managing documents and business rules that apply to them. It offers library services that profile, organize, and store documents in cabinets and folders. It employs ACLs (seven hierarchical permission levels are supported) for users and user groups. Documents are stored in either databases or file systems and are indexed by a full-text retrieval engine. Document attributes are stored and indexed in a RDBMS. The search language, an SQL extension, supports full-text and attribute searches across multiple documents.

All these systems use proprietary technology, specialized desktop applications, and are mostly concerned with the lifetime of documents. They are difficult to administer because software has to be installed on possibly thousands of computers. Using a document management system for the pure purpose of archiving is overkill in terms of both money and complexity of the system. In contrast, SaveMe leverages existing messaging infrastructures and thus, it does not require installation of new software on every user's desktop. In addition, individual users and IT personnel do not have to learn a new technology, resulting in substantial saving for an organization in terms of training and administration.

8 CONCLUSION

Electronic archiving of documents is an increasingly important task for most large enterprises. In addition, most large organizations have already deployed an efficient infrastructure for communication and collaboration by using Internet technology. The idea of the SaveMe architecture is to make use of this existing infrastructure for archiving documents. As we have shown in the paper, many tasks that have to be done in order to do archiving are already implemented in the messaging infrastructure. Therefore, we developed the SaveMe architecture based on Internet standards and implemented a prototype of SaveMe on top of a commercially available mail server. In this paper we also addressed several technical issues. In particular, we proposed a database design that is tuned to handle large amounts of documents. We also proposed a physical storage mechanism to manage a large collection of documents on optical media such as CDs or DVDs. Furthermore, we described the user interface and the difficulties we had to overcome in the prototype implementation of SaveMe.

REFERENCES

1. Aegis Star Corporation. The Aegis StarView System. http://www.aegisstar.com/
2. A. Biliris, R. Gruber, G. Hjalmtysson, H. V. Jagadish, M. A. Jones, M. F. McGroary, E. Panagos, M. Rabinovich, A. W. Robinson, S. Spear, D. Srivastava, D. Vista. Hermod: A Distributed Infrastructure for Electronic Messaging. Submitted for publication.
3. Documentum. Technology and Architecture of the Documentum Enterprise Document Management System. http://www.documentum.com/
4. Domino.Doc: Document Management for the Distributed Enterprise. http://www.lotus.com/products/dominodoc.nsf.
5. M. Crispin. Internet Message Access Protocol – Version 4rev1. University of Michigan, December 1996. RFC 2060. Available at http://www.imap.org/docs/rfc2060.html.
6. B. Gerber. Mastering Microsoft Exchange Server 5.5. Sybex, 1998.
7. T. Howes and M. Smith. LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. MacMillan Technical Publishing, Indianapolis, Indiana, 1997.
8. T. A. Mück and M. L. Polaschek. A Configurable Type Hierarchy Index for OODB. VLDB Journal 6(4), 312-332, 1997.
9. J.B. Postel. Simple Mail Transfer Protocol. Request for Comments 821, August 1982. Available at ftp://ds.internic.net/rfc/rfc821.txt.
10. S. Ramaswamy and P. Kanellakis. OODB Indexing by Class-Division. SIGMOD Conference 1995, 139-150.
11. RFC 2086. http://www.imap.org/docs/rfc2086.html
12. Siemens Nixdorf. DocuLive Document Management System. http://www.sni.com/
13. Verity. http://www.verity.com/products/whtpaper.htm