Advanced Facilities for Information Classification and Retrieval in Electronic Mail Systems

Simone Nunes Ferreira Karin Becker (Advisor)

Instituto de Informática - Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga 6681 , CEP 90619-900, Porto Alegre - RS, Brazil
Phone/Fax: +55-(0)51-3391511 ext. 3221
{simone, kbecker}@inf.pucrs.br

Abstract

This paper summarizes the main contributions of [FER97], in which mechanisms to aid users in the classification and retrieval of large volume of messages, considering the use of electronic mail as information systems, were proposed. The retrieval mechanism enables to locate and retrieve messages, as well as to obtain information about messages and classification structures, through the definition of ad hoc queries using a specific-purpose language. This language makes use of text information indexing and retrieval techniques. The classification mechanism is based on the concept of virtual folder, which allows messages to logically related to one or more folders, providing additional facilities and more flexibility to message classification procedures. A particular interesting type of virtual folder is the automatic folder, defined by a query that retrieves a set of messages meeting a specified criterion. Automatic folders are thus similar to the concept of view in database systems. Virtual folders help on the automatic and reorganization of messages, taking in charge the maintenance of the consistency between the folder's intent and the set of messages it represents. The paper discusses and illustrates the use of the language and of the classification facilities, and discusses implementation issues.

Key-Words: *query language, full text indexing and retrieval, object-oriented modeling, electronic mail*

1. Introduction

Electronic mail (email) has become an essential form of communication. One of its key features is the facility and rapidity with which information can reach a wide audience, with a very low cost, compared to other forms of communication technology (e.g. telephone, fax)[SPR91]. The growing size of messaging communities, specially when group communication tools like distribution lists or asynchronous conferencing systems are used, increases the capacity of disseminating information in an easy and timely manner [REI93, ROB91].

However, these same advantages are at the origin of the most serious problems of email: *information overload*. Regular users can receive between 30 up to 100 messages everyday [ROB91], and the problem is even worse for those who subscribe distribution or interest lists. The volume and pace of information can be overwhelming, given the amount of "junk mail", the multiple topic threads and the non-sequential patterns of related messages [PAL95]. Managing all this information is a laborious and time-consuming everyday activity, and electronic mail tools must provide advanced managing facilities for efficiently identifying, prioritizing, classifying, storing and retrieving messages [HIL85, PAL95].

A study on the use of electronic mail [MAC88] revealed that, although the use of email is strikingly diverse from user to user, two mail *email handling strategies* can be identified: *prioritizers* and *archivers*. Prioritizers are users interested in using email to maximize efficiency and save time. Prioritizers deal with the problem of information overload by reducing the size of

their mailbox, the number of folders and the number of the subscription lists. This kind of user tends to use message filtering and automatic processing facilities [TAY92, POL88, BER94] to reduce the amount of effort and time spent in the management of his/her mailbox. Archivers, on the other hand, use email as an *information source*, and are willing to spend extra time to avoid the possibility of missing something important. They resist to inhibit the flow of incoming messages by filtering, due to the risk of loosing information potentially useful. Moreover, they tend to save most received messages with the assumption that they may be useful (some day), have a large number of folders, and consequently have difficulty find filed messages. The archivers email handling strategy suggests that email is much more than an efficient communication technology: it is a rich source of *quality and up-to-date information* [MAC88, PAL95]. In both cases, the large amount of mails make it difficult to identify those that are important, to classify them according to appropriate criteria, and to discard useless messages [BAE93].

In this paper, we discuss the striking features of the mechanisms proposed in [BEC96, FER97, FER97b] to aid in the classification and retrieval of large volumes of electronic messages, considering mainly the needs of the *archivers* mail message handling strategy. The *classification mechanism* [BEC96] is based on the concept of *virtual folder*, which allows messages to be logically related to one or more folders, without requiring message duplication. The *retrieval mechanism* was designed as a specific-purpose retrieval language [FER97b] that enables users to locate and retrieve messages, information about messages and/or the classification structures. The language makes use of text information indexing and retrieval techniques in order to efficiently treat semi-structured messages, i.e. messages divided into a *header*, describing the sender, receiver(s), subject, etc., and the *content* of the message, written in natural language. Using this retrieval language, users can define *Automatic Folders*, a particular type of virtual folder, very similar to the concept of view in database systems [DAT86], in which a folder is specified by a query that retrieves a set of messages that meet a given criterion. These mechanisms are complementary to the functions present in mail systems (e.g. send, forward, receive, etc.), and mail tools (e.g. edit, print, etc.).

Among the benefits provided by the proposed mechanisms, two of them are of special relevance. First, automatic folders help on automatic classification and reorganization of messages, taking in charge the systematic maintenance of the consistency between a folder's intent and the set of messages the folder represents. Second, the retrieval language puts forward the foundations enabling mail systems to be treated as real information systems, where information can be easily be located and retrieved. In contrast to existing mail systems, either commercial [LAQ92] or experimental [BOR 91, NIC92, GOL92, MAL87, MAL89, LAI88, POL88, MAE94, EGL93, BER94, TAY92, POL88, WYL92], the retrieval facilities are not limited to the location and retrieval of messages, but enables to easily retrieve information about messages and/or classification structures.

The rest of the paper is organized as follows. In Section 2, current approached for dealing with the problem of information overload in electronic mail environments are discussed. The proposal is presented in Section 3, which describes the underlying philosophy, the related concepts, and the representation of these concepts using and object-oriented model. Section 5 describes the language designed to retrieve information in email systems, exemplifying its use. Section 6 presents the classification mechanism based on the concept of virtual folder. Section 6 discusses implementation aspects, and conclusions are drawn in Section 7, where future work perspectives are also presented.

2. Approaches for Dealing with Information Overload

2.1. Folders

In order to help managing the large volume of electronic messages, users tend to organize them using folders. The grouping of messages related to a common issue in folders is a useful way of locating them later, since it reduces the search space. Messages are assigned to folders using some user defined criterion, implicitly embedded in the name chosen for the folder. Once moved to a folder, the message remains there until its removal is explicitly determined.

Folders are classification structures implemented in most mail tools as *files* [POL88, WYL92, EGL93, BER94], in which messages are stored according to some specified format. Using folders, information contained in messages can located by a) *inspection* of the message contents using mail readers, b) *text location functions* available in message browsers, or c) use of *operational system commands*, such as UNIX commands grep or pick. These facilities are often primitive (e.g. limited to the header of the message, poor facilities - if any - to define complex search expressions using logical operators, etc), or may present a poor performance when a large number of folders, and/or messages is considered.

A consequence of implementing folders as files is that folders must be located *prior* to the location of the messages. However, the consistent and continuous management of email classification is a hard task, as exemplified by the problems discussed below:

- it may be difficult to determine the appropriate folder to move a message. This situation may arise when a message can be conveniently classified in more than one folder, or when the user, as time progresses, forgets the meaning of (some of) the folders he/she has created;
- it may be difficult to select the folder(s) to locate and retrieve messages, often implying the search in various folders. This situation may occur, for instance, when there is no folder for the subject at hand, when the user does not remember anymore the meaning of some folders, when messages were not correctly classified, etc ;
- necessity of periodical reorganization of folder contents and classification structures, according to new interests of the user, or increasing number of archived messages. This restructuring task is laborious and error-prone;
- it is difficult to maintain the consistency of the folder classification criterion and the set of messages it contains. Indeed, as time progresses, messages may not meet anymore the classification criterion defined for the folder (e.g. important messages). Other reasons may be the inadequate choice of a folder to move a message to, given that the folder name is the only an indication of the classification criterion the folder represents, or even the accidental assignment of messages to the wrong folders.

All the above mentioned problems are only symptoms of the real problem, which is the *lack of explicit classification criteria* for folders. Consequently, message classification in folders is a laborious and ineffective task. The use of folders as storing units may worse this problem, given that it prevents the retrieval and reorganization of messages independently of the way they were stored.

Another problem is that available message location facilities enable to locate messages, but not to locate *information* in general. One cannot, for example, easily query "When is the last time John sent me a message?", "Who sent me messages today?", "How many messages I exchanged with John yesterday?", "To which folder did I move that message?", etc, without inspecting the headers of all emails contained in one or more folders, or even the contents of these mails. Manual inspection can be a error-prone activity, in particular if the user cannot locate the appropriate folders to inspect.

2.2. Filtering and Automatic processing

Message filtering is characterized by the selection of a set of messages that meet some criterion [MAL87]. Filters are established in terms of information contained in messages, either in the header (e.g. to:, from:, etc.) or contents parts. The use of filters aims at reducing the user effort in the identification of messages for later processing (e.g. classification, deletion, etc.) [PAL95]. Filtering can be triggered automatically for *incoming messages*.

In most mail tools [ROB91, BOR91, LAI88, POL88, MAL89, GOL92, WYL92, TAY92, TER92, BER94, MAL87], automatic processing is implemented using rules. A rule is composed of a *selection part*, which describes a filter, and an *action part*, which defines a set of procedures to be applied to the set of filtered messages. Rules are triggered by system events, such as incoming mail, enabling the automatic processing of messages (e.g. automatic deletion or classification). Different proposals of rule based systems can be found in the literature [BOR91, NIC92, GOL92, MAL87, MAL89, LAI88, POL88, MAE94]. Tools such as ISCREEN [POL88], Tapestry [NIC92] and Procmail [BER94], ELM [TAY92], Mail Filter [WIL92], are based on semi-structured messages, whereas others use the concept of structured messages, discussed in Section 2.3.

The facilities for defining filters are normally the same message location functions available in mail tools. An alternative approach for filtering semi-structured messages can be found in TAPESTRY [TER92], which is based on a query language named TQL. The idea in TAPESTRY is to reduce information overload by having a central repository of messages. Individual users can then define TQL filters to search for interesting messages in the repository, and bring them to his/her own mailbox. An example of TQL query is *m.sender='john' and m.words = {'SBBD'}*, which filters all messages coming from john and which have the word "SBBD" in its contents. It is not reported in [GOL92] the use of TQL for searching messages in the user own mailbox.

The use of rules for message classification makes explicit the criteria to classify messages, aiding in the identification of messages and minimizing some of the problems mentioned in Section 2.1. However, it does not solve the problem of *maintaining the consistency* between the folder intended classification criterion and the messages it contains, since messages can be assigned to folders by other procedures (e.g. manually, or by other conflicting rules). It also does not address the problem of{\em inconsistencies due to the action of time}, nor help much in the reorganization of folder contents and classification structures, given that often filtering can only be applied to *incoming mail.*

Filters defined over the contents part of semi-structured messages have their expression power limited by the facilities available for dealing with text. Since the semantics of message contents, written in natural language, can not be easily identified [MAL89, HAM90], the construction of precise and useful filters can be a complex, if not impossible task. Existing filtering software such as [WYL92, TAY92, BER94] are not easy to setup and require the user to prepare special control files in a language that cannot be easily understood by non-computer specialists [PAL95]. The poor expressiveness of filters or the complexity of their definition can result in a selection of messages that is either incomplete or imprecise, which in the context of automatic processing can lead to undesired effects, such as deleting important messages or classifying a message in an unexpected way. Such kind of side-effect can be disastrous, specially considering the archivers profile.

Therefore, filtering and automatic processing can be invaluable for managing large volumes of messages, but they do not constitute a complete and fully reliable solution for identifying and classifying electronic messages, in particular considering the archivers profile.

2.3. Structured Messages

Structured messages are proposed as a means to increase the precision of filter description. A structured message has its content part divided into a number or pre-defined fields, defining a message template. Messages are normally associated to classes or types, and organized in a generalization hierarchy. Information Lens [MAL87, MAL89, ROB91], Object Lens [LAI88], Andrew Message System [BOR91] and PAGES [HÄM90] are examples of systems based on structured messages.

The use of structured messages makes easier the process of message identification, prioritization and classification, since its contents are composed of pre-defined fields, which can be used to construct more precise and reliable filters to base automatic processing [MAL87, ROB91, BOR91].

The use of structured messages is appropriate for those applications where it does exist a certain degree of structuring and standardization of the information exchanged by messages, such as office automation, work flow, etc. [KOS91], typical of organizations using email to fulfill time and task management work functions [MAC88]. However, its adoption becomes awkward in environments where there is the need of manipulation and exchange of information to which standardization cannot be easily introduced, or the need of communication among heterogeneous systems, which do not recognize each other's type system, if they recognize any type system at all. Therefore, despite the great advantages provided by the use of structured messages, its applicability is limited.

3. Proposal

3.1. Overview

The retrieval and classification mechanisms proposed in this work are targeted mainly to fulfill the needs of those individuals who use electronic mail as information source, i.e. the archivers. Archivers, according to [MAC88], are individuals that :

a) increase mail volume by subscribing to voluntary distribution lists;
b) save a large percentage of their mail messages;
c) maintain a large number of mail folders;
d) tend to read all of their mail, or try to;
e) have difficulty finding mail that has been filed.

The rationale for this proposal is that the solution for information overload considering this profile, can not be achieved by reducing the volume of received messages. Indeed, archivers tend to not trust filtering and automatic processing [MAC88], and they are always afraid of being unable to locate an automatically classified message or accidentally delete an important information. Archivers thus need a flexible solution to classify and locate messages and information contained in messages.

The classification mechanism in our approach is based on the concept of *virtual folder*, which allows messages to be logically related to one or more folders without requiring duplication. Automatic folders are a particular useful type of virtual folders, specified by a query that retrieves messages that meet a certain criterion. By explicitly associating to a folder its classification criteria, this mechanism helps on the automatic classification and reorganization of messages, maintaining the consistency between the folder intent and the set of messages the folder represents.

The retrieval mechanism offers advanced facilities for text information search, and enables to treat electronic mail as an information system. It was designed as specific-purpose retrieval language enabling the location and retrieval of *information* considering large volumes of electronic messages. It offers facilities to locate information independently of message or folder inspection, through the definition of ad hoc queries to locate and retrieve not only messages, but also

information contained in messages, folder classification structure, or any combination of these. Indeed, it allows not only to locate and retrieve messages, but also information contained in messages and/or its classification structures.

The concept of virtual folder and its modeling using an object-oriented model, which underlie the retrieval and classification mechanisms, are discussed in the remaining of this section. The query language is then presented in Section 4, whereas the classification process using virtual folders is the object of Section 5.

3.2. Virtual Folders

From a user perspective, a folder is a message classification unit, as described in Section 2.1. In this work, however, folders are not implemented as a physical unit for storing messages. Instead, messages are related to folders by logical relationships. This allows a greater flexibility for message classification, enabling a message to be associated to multiple folders without having to be duplicated, and making it easier the process of message reorganization. There are three types of folders, referred to as manual, automatic and system folders, described below:

a) Manual Folder

In terms of functionality, manual folders act as regular folders, as described in Section 2.1, where users are in charge of message classification, and definition and maintenance of classification criteria. Messages are assigned_to, removed_from manual folders through the explicit application of classification operations, described in Section 5.1.

b) Automatic Folder

Using automatic folders, the classification of messages is achieved by the execution of a query expression, which explicitly states and maintains the classification criteria. In other words, an automatic folder represents the set of messages that meet a certain criterion at any given moment. An analogy can be established between the concept of automatic folder and the concept of *view* in databases, which is a virtual relation defined by a query over existing, concrete relations [DAT86]. Therefore, an automatic folder is specified by a name and a message retrieval expression. This expression is executed every time a user wishes to retrieve the set of messages represented by the folder criterion. This approach guarantees the consistency between the classification criterion specified for a folder, and the set of messages associated to it. To guarantee this integrity, it is not possible to explicitly assign or remove messages to/from automatic folders.

c) System folders

In systems folders, the classification criteria are determined and maintained by the system. A minimum set of such folders was defined in this work, based on those normally provided by mail tools [LAQ92]:

- NEW: messages received which have not been read by the user;
- OLD : messages read by the user;
- REPLY : messages received in response to messages sent by the user;
- ANSWERED : messages received to which the user has already replied.

Messages cannot be explicitly assigned/removed to/from system folders by users. Instead, the application of operations like read, reply, etc. to messages results in their classification in the appropriate system folder, under the control of the system.

Each message is classified into one of the system folders (NEW or OLD), and possibly according to other folders, either automatic, manual or system.

3.3. Data Modeling

An object-oriented model was chosen in order to properly represent the inherent characteristics of these concepts, such as :

- multivalued attributes, needed to characterized both message properties (e.g. to:, cc:) and folder properties (i.e. the set of messages associated to folders);
- optional properties of messages (e.g. not all messages have a cc: field in its descriptor);
- complex objects, needed to represent the relationship between messages and folders;
- behavior that can be associated to messages (e.g. discard, reply), and folders (e.g. create, classify).

Object-oriented models conveniently handle all these requirements, and have additional facilities that can be extensively used, such as data abstraction, inheritance, overriding, late binding, etc. [ATK89]. Figure 1 depicts the main classes used to represent messages and folders using the O2 data model [DEU91], and highlights the properties relevant to the rest of our discussions.

```
Class MSG
  type tuple (  Msg_id: string,
                From: Address,
                To: set (Address),
                Cc: set (Address),
                Date: Date,
                Subject: string,
                Size: Integer,
                Content: Text )
  method   create,
           discard,
           receive,
           send,
           ...
  with extension

Class FLD
  type tuple (  Name: string,
                Creation: Date,
                Author: string,
                ... )
  method   msgs_in: set(MSG),
           count_msg : integer,
           rename,
           create,
           remove,
           remove&discard,
           ...)
  with extension

Class MAN_FLD inherit FLD
        type tuple ( msgs: set(MSG))
        method   classify (msgs: set (MSG)),
                 unclassify (msgs: set(MSG)),
        with extension
end

Class AUT_FLD inherit FLD
        type tuple ( expression: Query)
        method   set_expression (exp: Query)
        with extension
end

method body msgs_in: set (MSG) in class MAN_FLD
          { return self.msgs}

method body msgs_in: set (MSG) in class AUT_FLD
          {return self.expression.execute}

method body set_expression (exp: Query) in class AUT_FLD
          {self.expression = exp;}
```

Figure 1 - Data Modeling

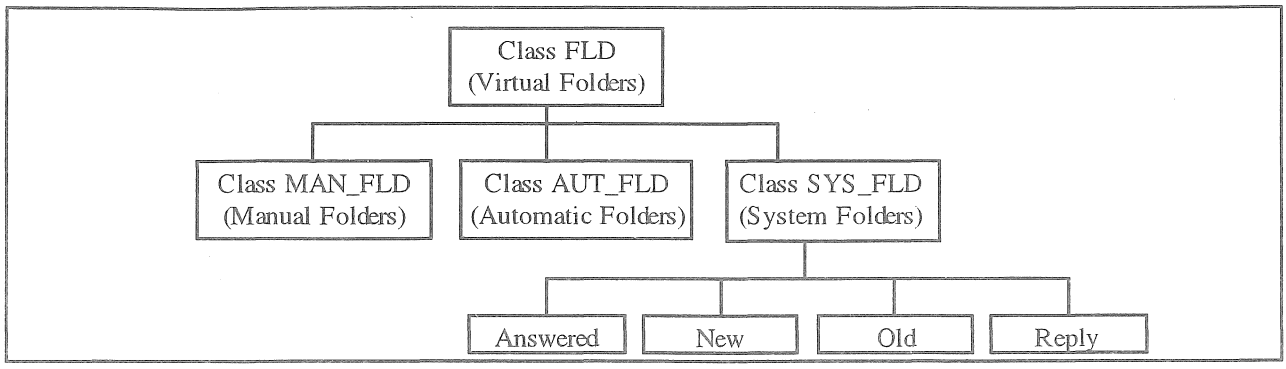

Figure 2 - Hierarchy of Classes Representing Folders

As it can be seen in Figure 1, messages are modeled by the class ***MSG***, characterized by the usual fields of semi-structured messages (e.g. *To:*, *Cc:*), and the common operations that can be applied to them (e.g. *send*, *discard*). The class ***FLD*** captures the properties common to all folders. The operation *msgs_in* returns the set of messages currently associated to the folder object. This operation is overridden in the subclasses of ***FLD*** (Figure 2) in order to define the proper implementation, according to the specificities of each folder type. For manual folders (class ***MAN_FLD***), this operation returns the value of the attribute *msgs*, which contains the message objects presently associated to the folder. For automatic folders (class ***AUT_FLD***), the operation triggers the execution of the query that defines the folder's classification criterion, also stored as an attribute (*expression*). The implementation of the subclasses representing system folders are not shown here due to space limitations. Notice that this modeling allows to treat folders uniformly, relying on dynamic binding to select the proper implementation according to folder type.

4. Query Language

A specific-purpose retrieval language was designed with the aim of locating and retrieving information considering large volumes of electronic messages. It offers facilities that enable users to locate information independently of message or folder inspection, through the definition of ad hoc queries to locate and retrieve not only messages, but also information contained in messages, folder classification structure, or any combination of these. This language is intended to fulfill mainly the needs of *archivers*, who often have difficulty finding filed information, due to large number of filed emails and folders used to classify them. Queries can be used to exploit electronic mail as an useful information system, or to define automatic folders. They could also be used in the definition of rules for automatic processing of retrieved messages or folders (e.g. discard, forward).

Considering the contents of semi-structured messages, advanced text retrieval facilities were also required to implement this language. Text information retrieval techniques can be very powerful, allowing inexact string matching, use of synonyms, positional search, etc [KOS92]. We have considered an initial set of useful text search facilities for our language, namely prefixed match, synonyms and occurrence frequency.

The language was designed inspired on O2SQL features [BAN89], but it is restricted to specific constructors for dealing with messages, folders, and text retrieval. A language as O2SQL would be too general for our purposes, since a limited number of classes need to be considered, and O2SQL generality would unnecessarily complicate the language. The main features of this language and its implementation are discussed in the remaining of this paper. As a final remark, we do not intend this language to a final user, though it could be easily used by those with a minimum experience on query definition (a similar assumption is done in TAPESTRY, with regard to the use of TQL [TER92]). At the present stage of this work, the issue of providing a proper graphical

interface enabling users to formulate *ad-hoc* queries by themselves, perhaps in a "Mail by Example" style, has not been addressed yet.

4.1 Language Syntax

A simplified syntax of this language is presented in Figure 3. In the figure, words in **bold** are reserved words of the language. The reader can refer to [FER97] for a detailed discussion of this language, as well as its complete syntax. Due to space limitations, we will present the striking features of this language using examples, after a brief description of each of its clauses. The language is composed of four main clauses, as follows:

```
Single_Query  ::=  SELECT    <result>
                   FROM FOLDER      f   IN  <flds>   [WITH <fld_cond>]
                   [FROM MESSAGE    m   IN  <msgs>   [WITH <msg_cond>]
                   [WHERE  <comp_cond> ]]
                   |
                   SELECT    <result>
                   FROM MESSAGE     m   IN  <msgs>   [WITH <msg_cond>]

<result>      ::=  <folderlist> | <msglist> | <msg_att> | <fld_att> |
                   TUPLE((msg_att | fld_att)+)

<folderlist>  ::=  (fld)+  /* where fld is an instance of FLD*/

<msglist>     ::=  (msg)+  /* where msg is an instance of MSG */

<flds>        ::=  FLD | MAN_FLD | AUT_FLD |
                   NEW  | OLD | ANSWERED | REPLY | <named-folder-query>

<msgs>        ::=  MSG | f.msgs_in | <named-message-query>
```

Figure 3 - Simplified Query Language Syntax

- ***Select clause***: specifies whether the result of a query is a set of messages (*msglist*), set of folders (*folderlist*), or one or more message and/or folder attribute values (*msg_att* and/or *fld_att*). *msglist* and *folderlist* are, respectively, a set of message/folder object identifications (internal surrogates).

- ***From Folder clause***: allows users to specify which folders will be considered during query processing (*flds*). The default is the set of all existing folders (i.e. instances of Class ***FLD***). Restrictions can be made through the specification of a ***FLD*** subclass, or using folder objects selected by a previous query (*named-folder-query*). Optionally, additional restrictions can be applied to the resulting folder subset by the specification of conditions on folder attributes (WITH subclause), using relational, logical and text retrieval operators. The *From Folder clause* is not required if the user wants to retrieve messages or message information independently of any folder information.

- ***From Message clause***: allows users to specify the set of messages that will be considered during query processing (*msgs*). The default is the set of all existing messages (i.e. instances of class ***MSG***). This set can be delimited in two ways. First, by using only the messages

associated to the folders filtered by the *From Folder clause* of the query, as specified by the expression *f.msgs_in*. The other possibility is to use the set of message objects selected by a previous query (*named-message-query*). As with folders, the resulting set of messages can be further restricted by using conditions over message attributes (WITH subclause). In addition to the text, relational and logical operators, set operators (e.g. ∈ ,,) can be used for the multivalued attributes of messages (e.g. To:). The *From Message clause* is not required if the user wants to retrieve folders or folder information, independently of associated messages information.

- ***Where clause*** : this clause is necessary only when the user wants to define selection conditions that compare message attributes to folder attributes. Conditions are specified using relational, logical, set and text retrieval operators. To use the *Where* clause, both *From Folder* and *From Message* clauses must have been specified.

All possible combinations of these 4 query clauses are shown in the BNF of Figure 3. Though not indicated in the syntax, query expressions can be composed using UNION, INTERSECTION and MINUS set operators over compatible atomic queries *(Single_Query)*. Query results can also be named such that they can be used in other queries, allowing in this way a query strategy based on incremental refinements. The possible combinations of the clauses and the striking features of the language will be illustrated through the following examples:

Example 1 : Selecting messages independently of folder information and use of synonyms

a) Which messages received after 10/03/97 are about databases?

```
SELECT m
FROM MESSAGE m in MSG WITH m.content HAS 'DB' and m.date > 10/03/97
```

If the user defines synonyms for the expression "database", say, "DB", "DBMS", "Object Oriented Databases", etc., the condition expressed using the operation *HAS* selects all messages where one of these defined synonyms appears at least once in their content part. In this example, the set of all existing message objects is considered, since the From Message clause specifies the use of the class *MSG*. Notice that message objects are being retrieved in this query, but any message attribute could have been specified as well (e.g. m.date, if the user would like to know only the sending date of these messages).

Example 2 : Selecting folders independently of message information and prefix searching

b) Which automatic folders have words starting with *paper* in its definition expression?

```
SELECT f
FROM FOLDER f in AUT_FLD WITH f.exp HAS 'paper*'
```

The query structure is similar to the previous example. The only difference is that it retrieves folders, instead of messages. Only automatic folder objects are considered, since the From Folder Clause specifies the use of class *AUT_FLD*. Recall that automatic folders are defined by a query that selects a set of messages meeting a given criterion. In class *AUT_FLD*, the attribute *exp* stores the query defined for the automatic folder. Queries such as the one shown in this example could be useful, for instance, during classification structure reorganization or inspection [BEC96]. This

example also illustrates the use of prefix matching, allowing the search of any expression starting with *paper* by using **HAS** operator and the symbol * after *paper* substring.

Example 3 : Using From Folder and From Message clauses together

c) Which are the *new* messages from 'john smith'?

```
SELECT m
FROM FOLDER f in NEW
FROM MESSAGE m in f.msgs_in WITH m.from HAS 'john smith'
```

In this example, using the From Folder clause, the user restricts the query to consider exclusively the folder NEW (class *NEW* has a single instance). In the From Message clause, the user selects only messages belonging to *NEW* system folder, using the operation *msgs_in* available for folder objects. He/she then further selects those messages where John Smith is the sender. This example shows how to retrieve messages qualified by their classification relationship with folders. The result of this query could be as well folders, or message/folder attributes.

Example 4 : Using WHERE clause (A typical example of archivers ...)

d) Where is the message that I received from John in which he was talking about the price of the system? I remember that I moved it to a folder with the same name of the message subject, but I can not remember which one ...

```
SELECT m
FROM FOLDER f in MAN
FROM MESSAGE m in f.msgs_in WITH m.content HAS ('sys*' and '$*') and m.from HAS 'john'
WHERE m.subject = f.name
```

This example illustrates how helpful a retrieval language for email systems can be for archivers. Note that the Where clause is used to compare folder and message attributes and that this comparison is made considering only those messages that are related to the retrieved folders (this condition is assured by the use of *f.msgs_in* in the From Message clause).

Example 5 : Naming Queries and Incremental Query Definition

e) Which messages received from 'John Smith' have not been answered yet?

```
J_TOREPLY :=    SELECT m
                FROM FOLDER f in ANSWERED
                FROM MESSAGE m in MSG WITH m.from HAS 'John Smith'
                WHERE m f.msgs_in
```

This query selects all existing messages from 'John Smith' that are not related to folder ANSWERED. Notice that messages and folders are selected independently one of another. They are related here through the *Where clause*. The example also shows how to name a query, such that it can be used later on other queries.

f) Which important messages, received from 'John Smith', have not been answered yet?

```
SELECT m
FROM FOLDER f in FLD WITH f.name = 'important'
FROM MESSAGE m in J_TOREPLY
WHERE m f.msgs_in
```

Notice that the same query could have been formulated by the intersection between the named query *j_toreply* and a query selecting all messages associated to the folder named *important*.

Example 6 : Retrieving Information about messages and folders

g) Which are the names of the folders containing more than 50 messages, with at least one message received today?

```
SELECT f.name
FROM FOLDER f in FLD WITH f. count_msg > 50
FROM MESSAGE m in f.msgs_in WITH m.date = today
```

h)When 'John Smith' sent important or urgent messages to me, and which were their subject?

```
SELECT TUPLE (f.name, m.date, m.subject)
FROM FOLDER f in FLD WITH f.name = 'important' or f.name = 'urgent'
FROM MESSAGE m in f.msgs_in WITH m.from HAS 'John Smith'
```

Notice that the TUPLE constructor is used when more than one attribute compose the result of a query. It is important to highlight that all previous examples could have been defined to return messages, folders or their attributes.

The intent of the above examples is to illustrate the flexibility and expressiveness of the proposed query language, which associates:

- characteristics of database query languages (e.g. selection and projection over attributes, set operators, limited forms of join);
- characteristics of full text search systems, allowing searches using prefix, synonyms and occurrence frequency;
- operators to deal with multivalued complex objects (e.g. $\in$, $\notin$, $\subset$, $\supset$, msgs_in, etc.)

5. Classification Mechanism

The goal of the classification mechanism is to enable the easy and flexible organization of messages into folders, based on the concept of virtual folder, described in Section 3.2. The logical association of messages to folders presents a number of advantages, among them a) the possibility of relating a same message to various folders, without message duplication; and b) it frees users from the burden of locating appropriate folders prior to the search and retrieval of messages with desired properties. This is a constraint presented by systems in which folders are both the conceptual and the physical storing unit of messages, as discussed in Section 2.1.

The use of automatic folders presents additional advantages, in particular the maintenance of the consistency between a folder's classification criterion and the messages related to it. Indeed, messages are dynamically associated to an automatic folder only when the user wishes to manipulate the folder's content, triggering thus the execution of a query that retrieves the set of messages meeting the folder's criterion at that particular time. Another major advantage of automatic folders is that messages can be restructured into another classification structure simply by

modifying the query expressions that define automatic folders. The approach is thus more powerful than rules for automatic processing with regard to classification, where consistency is not automatically maintained, and restructuring is not supported (cf. Section 2.2).

The use of manual folders provides users with the same flexibility for message classification offered by regular folders systems, but with the additional advantages provided by the logical association between message sand folders. Using manual folders, users can group messages by any subjective criteria, and they can explicitly associate/dissociate messages to/from manual folders.

Operations defined for class ***FLD***, inherited by ***FLD*** subclasses, enable the user to create (*create*), update (e.g. *rename*) and delete (e.g. *remove*, *remove&discard*) manual and automatic folders. In the case of system folders, the triggering of these operations is exclusively in charge of system. ***FLD*** subclasses add behavior specific to each type of folder. In the remaining of this section, we further discuss and illustrate the behavior defined for manual and automatic folders.

5.1. Classification with Manual Folders

Most operations available for manual folders are inherited from the superclass ***FLD***. The behavior specific to manual folder objects is defined by the operations *classify* and *unclassify*, added to ***MAN_FLD*** interface (Figure 1), since the operations allowing messages to be explicitly associated_to / dissociated_from folders have its applicability restricted to the classification using in manual folders. This decision was taken in order to guarantee the consistency of automatic folders with regard the classification criteria defined for them. However, this does not represent a serious burden, given that users can define automatic folders based on the combination of other folders, either manual, automatic or system, as it will be discussed in Section 5.2. Figure 4 illustrates the use of operations defined for ***MAN_FLD***, to create, rename, classify a message and remove a folder.

```
my_folder := MAN_FLD.create('important')
my_folder.rename('this_week_important')
my_folder.classify(this_message)
my_folder.remove
```

Figure 4 - Manual Classification

It is important to highlight that a conceptual difference is assumed between dissociating a message from a folder (operation *unclassify*, defined for ***MAN_FLD***), and discarding a message (operation *discard*, defined for ***MSG***). In the first case, the message is dissociated from the manual folder to which the operation is being applied, whereas in the second case, the message is destroyed by the system.

Likewise, folder removal (operation *remove*) does not imply the deletion of the associated messages. When the operation remove is applied to a manual folder, the messages are dissociated from the folder, and the folder is then removed. The operation *remove&discard*, on the other hand, implies as well the deletion of all associated messages.

5.2. Classification with Automatic Folders

The creation of automatic folders is illustrated in Figure 5. It can be noticed that a query expression has to be supplied as argument, together with the folder name. In the example, an automatic folder named *'VLDB'* was created, which gathers all messages where the word VLDB or its defined synonyms appear in the subject or contents part.

Users can define automatic folders based on the combination of other folders, either manual, automatic or system. For example, in Figure 5 another automatic folder named *'important'* is defined as the union of all messages in the manual folder *'this_week_important'* (containing messages selected by the user as important, as illustrated in Figure 4), and those contained in the automatic folder named *'VLDB'*.

Of particular interest is the operation *set_expression*, defined for ***AUT_FLD***. When reorganizing their classification structures, users can simply change the query expression associated to automatic folders to have them associated with a whole new set of messages, instead of the laborious and error-prone task of manually moving messages from the old to the new classification structure. For example, suppose that after meeting the deadline for submitting a paper to the Conference VLDB, the user changes his/her priorities to the activities of his/her research project involving the implementation of a query optimizer. The folder *'important'* can be completely restructured simply by updating the related query expression, as shown in Figure 5.

```
vldb_fld := AUT_FLD.create('VLDB',    select m
                                      from m in MSG
                                      with m.subject has 'VLDB'
                                      or m.content has 'VLDB')

imp_fld := AUT_FLD.create('important', select m
                                       from f in FLD with f.name = 'VLDB'
                                       or f.name = 'this_week_important'
                                       from m in f.msg_in)

imp_fld.set_expression( select m
                        from f in FLD with f.name = 'optimizer'
                        from m in f.msg_in)
```

Figure 5 - Automatic Classification

The operations *remove* and *remove&discard* applied to automatic folders have the same consequences as for manual folders. In the first case, it destroys the folder without any impact on the messages that meet the specified folder criteria. In the later, the query is executed, and all retrieved messages are discarded, after folder deletion.

6. Implementation

6.1. Query Processing

The main purpose of the proposed retrieval mechanism is to provide the easy and fast access to all information related to messages (i.e. message contents and header fields), and messages classification structure. To attain a reasonable performance, full text indexing/retrieval techniques [BAE92b] were chosen to implement it. Full text information indexing/retrieval systems are more flexible, powerful and avoid the complicated and expensive process of semantic indexing [BAE92]. When using full text indexing, all text information is referenced by an index that provides easy and fast access when it has to be retrieved later. The adopted library of full text indexing/retrieval used [BAE92b] has primitives for incremental full text indexing, and one text retrieval primitive that allows prefixed match using synonyms, and which also returns text occurrence frequency.

To establish the correspondence between the queries defined using the proposed language and the text retrieval primitive available in the library, a set of intermediate *retrieval functions* was

defined, presented in Section 6.1.1. In this way, each query expression using the language is mapped into a sequence of intermediate retrieval functions, which in turn call accordingly the text retrieval primitive to obtain the result of the query. This mapping is used to integrate the querying features provided by the language with the one provided by the library, associating a semantics to a set of calls of the text retrieval primitive.

To guarantee the expressiveness of the language, this set of retrieval functions was defined based on the Complex Values Algebra (ALG ^{CV}) described in [ABI95], given that *MSG* and *FLD* objects represent complex values. The set of defined retrieval functions was not intended to implement the complete set of algebra operators. Instead, it was designed only to cover the required mapping possibilities for query processing, as discussed in Section 6.1.2.

6.1.1 Retrieval Functions

The retrieval functions, described below, were designed based on the possible combination of query clauses, as discussed in Section 4.1 (Figure 3). Each retrieval function is implemented using calls to the text retrieval primitive of the adopted full text indexing/retrieval library. There are also retrieval functions performing union, difference and intersection between any compatible sets, not discussed here.

- **SELECT_FLD (fldlist_in, fld_cond): fldlist_out**
  Select all folders (fldlist_out) from a list of folders (fldlist_in) that satisfy the condition fld_cond
- **SELECT_MSG (msglist_in, msg_cond): msglist_out**
  Select all messages (msglist_out) from a list of messages (msglist_in) that satisfy the condition msg_cond
- **MSG_IN (fldlist_in): F-Msglist_out**
  Return a flat structure (F-Msglist_out) in the form of pairs of messages with their corresponding folders, for all folders specified in fldlist_in. This function corresponds to the application of the unnest operator of the ALG CV to folders, with regard to their associated messages. This function is used when folders must be selected based on characteristics of messages to which they are related (or vice-versa), such as in the queries of examples 3 and 4.
- **FLD_MSG (F-Msglist_in, comp_cond): F-Msglist_out**
  Select all associations between folders and messages (F-Msglist_out) from a list of such associations (F-Msglist_in) that satisfy the condition comp_cond, which compares message and/or folder attributes.
- **MSGxFLD (msglist_in, fldlist_in, comp_cond): M-fldlist**
  Join messages in msglist_in with folders in fldlist_in considering the join condition specified in comp_cond, generating M-fldlist. This function is used to compare message and folders attributes, when no association between folders and messages was previously specified, such as in query f of example 5.
- **GET_INFO (List, result_spc): Result_out**
  Return message, folder, or message/folder attribute(s) (Result_out) from the previous results in List, according to the specifications in result_spc. This function implements projection operations.

6.1.2 Mapping Query Expressions

The mapping of a query expression into a set of intermediate functions is based on the parameters specified in each query clause. The query processing starts by the execution of the *FROM FOLDER* and *FROM MESSAGE* clauses, followed by *WHERE* clause. It then ends by the execution of *SELECT* clause. Figure 6 sketches all the possibilities for the mapping process, which are detailed below.

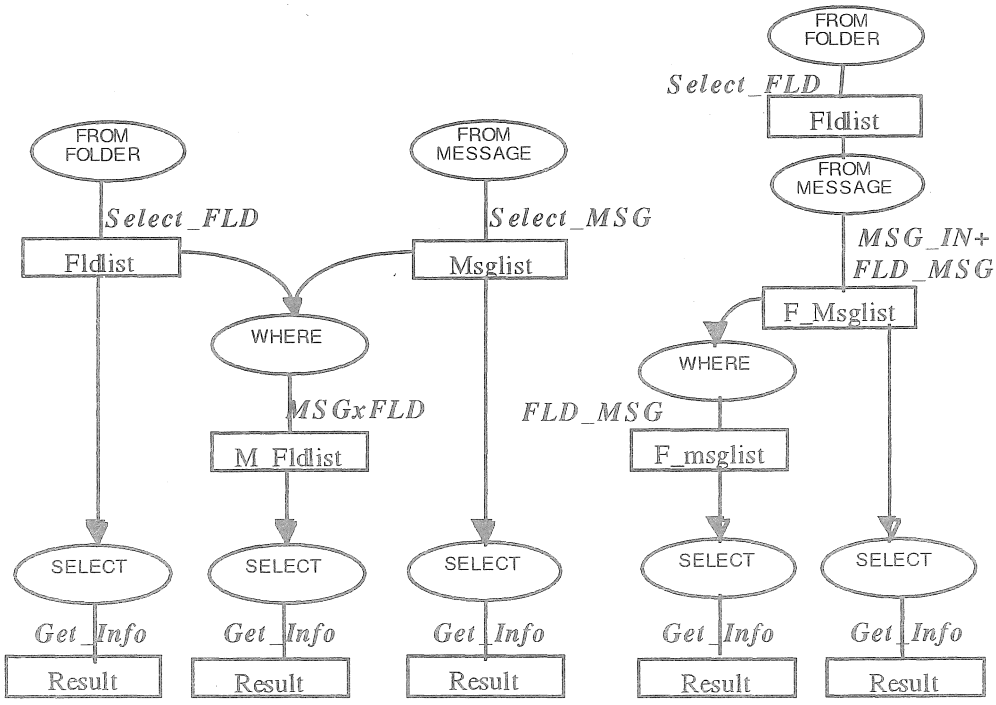

Figure 6 - Mapping Possibilities of Query Clauses into Retrieval Functions

The mapping of the *From Folder* clause, if defined in the query expression, is directly made using the SELECT_FLD retrieval function applied either to the extension of class FLD (or one of its subclasses) or to a given folder list (in the case of a named query), resulting in a folder list to be used later as argument, in other retrieval functions.

If the *From Message* clause is specified in the query expression, there are two mapping possibilities:

- From Message clause does not use *f.msgs_in* operation (Figure 6.(A)): the resulting message list is obtained by applying the retrieval function SELECT_MSG either to the extension of class MSG, or to a given message list (in the case of a named query). The resulting message list is used later as argument, in other retrieval functions.
- From Message clause uses *f.msgs_in* operation (Figure 6.(B)): the resulting message list will be obtained by using MSG_IN (to retrieve all messages associated to the folders specified in the From Folder clause) and FLD_MSG retrieval functions (to restrict the messages to those meeting the condition stated in the WITH subclause, if it is specified). The result of this function is a F-Msglist that will be used later as argument in other retrieval functions.

When the *Where* clause is specified in the query, the next mapping step is the processing of message and folder attributes comparison it defines. The same possibilities described above for the From Message clause exist for processing the Where clause:

- In the first case (Figure 6.(A)), the Where clause processing is performed by MSGxFLD function using as parameter the previous results obtained during From Folder and From Message clauses processing. Indeed, since no relationship was established between folders and messages retrieved by the From Folder and From Message clauses, respectively, they must be joined based on conditions specified in the Where clause.

- In the second case (Figure 6.(b)), the Where clause processing is by FLD_MSG function, using as parameter the previous results obtained from the processing of From Message clause processing.

Finally, the resulting structures are used as GET_INFO parameters to project the final results. Figure 7 presents the queries of examples 4 and 5 (*query e*) mapped into the corresponding retrieval functions.

Query	Mapping
SELECT m FROM FOLDER f in MAN_FLD FROM MESSAGE m in f.msgs_in WITH m.content HAS 'sys*' and '$*' WHERE m.subject = f.name	SELECT_FLD (MAN_FLD, ""): folderlist1 MSG_IN (folderlist1):f_msglist2 FLD_MSG (f_msglist2,"m.content HAS 'sys*' and '$*'"): f_msglist3 FLD_MSG (f_msglist3,"m.subject=f.name"): f_msglist4 GET_INFO (f_msglist4, m): msglist5
SELECT m FROM FOLDER f in ANSWERED FROM MESSAGE m in MSG WITH m.from HAS 'John' WHERE m f.msgs_in	SELECT_FLD (ANSWERED, ""):fldlist1 SELECT_MSG (MSG, "m.from HAS John"):msglist2 MSGxFLD (fldlist1, msglist2, "m f.msgs_in):m_fldlist3 GET_INFO (m_fldlist3, m): msglist4

Figure 7 - Mapping Examples

6.2. Prototype

Figure 8 presents an overview of the prototype currently under implementation. The prototype is being implemented in C++ language, and a full text indexing/retrieval library [BAE92b] is used to guarantee a reasonable performance. At the present stage of the implementation, only standard Internet messages (RFC 822) and UNIX operating system have been considered. The architecture is divided into 4 main subsystems, namely *storage*, *indexing*, *retrieval*, and *interfacing*, described below.

a) Storage Subsystem : Message contents, Message headers and folder description are stored as files maintained in distinct directories. This choice aims at facilitating the indexing process. The Storage Subsystem is also responsible for maintaining all corresponding indexing structures used.

b) Indexing Subsystem : there are four indexes used to fully index *message contents* (index ***msg***), *message header* (***header***), *folder descriptor* (***fld***), and *message-folder association* (***fld-msg***). These indexes allow the retrieval of messages, folders and information about them in a completely independent way.

c) Retrieval Subsystem : This subsystem is responsible for translating every query in terms of the text retrieval primitive, using the two step mapping described in Section 6.1. The full text retrieval primitive makes extensive use of the above defined indexes to efficiently access all data.

d) Interfacing Subsystem : This subsystem has the goal of integrating mail system operations (e.g. send, receive, reply) with the proposed mechanisms. Any email system that provides functions to extract mails to single files can be used in association with the proposed mechanisms. The interfacing subsystem takes the files containing the incoming or outgoing messages, and process them to extract header and content information, so as to prepare all required data to the indexing

subsystem. Messages are fully indexed when they arrive, using incremental indexing (indexes ***msg*** and ***header***). Indexes are also updated when folders are created, renamed or destroyed (index ***fld***), when messages are classified/unclassified in/from manual folders (index ***fld-msg***), or when messages are discarded (indexes ***msg*** and eventually ***fld-msg***).

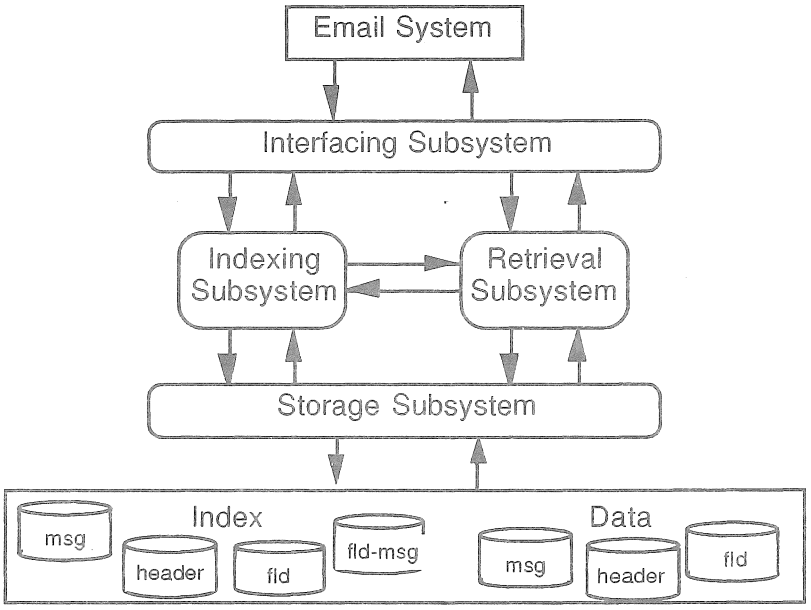

Figure 8 - Prototype Architecture

7. Conclusions

In this work, we have presented classification and retrieval mechanisms intended to aid in the management of large volumes of semi-structured electronic messages. These mechanisms are particularly targeted to fulfill the needs of archivers email management strategy, typical of people who uses electronic mail as an up-to-date and essential source of information in their everyday activity. Though we have considered in this proposal the use of semi-structured messages, seeking for a wider applicability of these mechanisms, they can be easily integrated in environments considering structured messages, in a complementary way. In the same way, these facilities are also complementary to filtering and automatic processing in the solution to information overload.

The retrieval mechanism, in the form of a *query language*, puts forward the foundations enabling mail systems to be treated as real *information systems*, where *information* can be easily be located and retrieved. To the authors' knowledge, no existing electronic mail tool explicitly offers this latter facility.

Using the proposed classification mechanism, users can classify messages using manual or automatic folders. Manual folders present the following advantages compared to regular folders : a) the possibility of relating a same message to various folders, without message duplication; and b) it frees users from the burden of locating appropriate folders prior to the search and retrieval of messages with desired properties. The use of automatic folders presents as additional advantages: a) the maintenance of the consistency between a folder's classification criterion and the messages related to it, and b) flexible reclassification simply by modifying the query expressions that define automatic folders. With regard to classification, automatic folders are thus more powerful than rules in automatic processing, since in the later consistency is not automatically maintained, and restructuring is not supported.

A prototype is currently under development, and it is based on full text indexing/retrieval techniques, in order to enable the location of unstructured text (in particular, located in the contents

part of messages) with acceptable performance. A two-step mapping process to transform a query in terms of text retrieval functions was proposed, based on the use of intermediate retrieval functions. This mapping integrates the query features provided by the language and the text retrieval primitive provided by the adopted library, associating a semantics to a set of calls of the text retrieval primitive. The intermediate retrieval functions were based on an algebra for complex values (ALG^{CV}). They were adapted for handling two specific complex types (i.e. messages and folders), so as to cover the required mapping possibilities for query processing. In terms of expressiveness, the functions enable projection, selection, limited forms of joins, set operators (union, minus, intersect), and a specific unnest operator. The current state of the prototype does not allow a sound performance evaluation, but preliminary tests suggests a reasonable level of performance.

Future research topics include, among others, the definition of a visual language for query definition by non-expert users, as well as for the visualization of the results of a query; the extension of text retrieval facilities (e.g. positional text search); an empirical evaluation of the query language, in terms of performance and facilities required; extension of the proposed mechanisms to allow folder/message automatic processing; application of the mechanisms to other environments, such as news reader, on-line document bases, etc.; and structured messages support.

References

[ABI95] ABITEBOUL,S. ;HULL,R.; VIANU, V. Foundations of Databases .Addison-Wesley Publishing Company, Chap. 10, 1995.

[ATK89] ATKINSON, M.; BANCILHON, F.; DE-VITT, D.; DIETTRICH, K.; MAEIR, D.; ZDONIK, S. The object-oriented Database Manifesto. In: International Conference on Deductive and Object-oriented Database, Kyoto, 1989. Proceedings.

[BAE92b] BAEZA-YATES, R; GONNET, G.H. A New Approach to Text Searching. In: Communications of the ACM, Vol. 35(10):74-82, Oct. 1992.

[BAE92] BAEZA-YATES. Text Retrieval: Theory and Practice. Information Processing 92, Volume I, Elsevier Science Publishers B.V. (North-Holland), 1992.

[BAE93] BAECKER, R. M. Groupware and Computer-Supported Cooperative Work assisting human-human collaboration. Baecker, NY, 1993.

[BAN89] BANCILHON, Francois, et al. A Query Language for de O2 Object-Oriented Database System. Technical Report no. 2 Altair. 1989.

[BAR93] BARBOSA, E.; ZIVIANI. From Partial to Full Inverted Lists for Text Searching. Technical Report, Universidade Federal de Minas Gerais, Br, 1993.

[BEC96] BECKER, K.; FERREIRA, S. Virtual Folders: Database Support for Electronic Message Classification. In: International Symposium on Cooperative Database Systems for Advanced Applications, Heian Shrine, Kyoto, Dec. 1996. pp. 239-2346.

[BER94] BERG, S. Procmail - autonomous mail processor. WTH- Aachen, Germany. ftp://informatik.rwth-aachen.de/pub/packages/procmail.

[BOR91] BORENSTEIN, N.; THYBER, C.A. Power, ease of use and cooperative work in a practical multimedia message system. In: International Journal of Man-Machine Studies, 34(2):229-259, Feb. 1991.

[DAT86] DATE, C.J. An Introduction to Data Base Systems. Addison-Wesley Publishing Company Inc. USA, 4th. Ed., 1986.

[DEU91] DEUX,O et al. The O2 System. In: Communications of the ACM, 34(10):34-48, Oct. 1991.

[EGL93] EGLOWSTEIN, H.; SMITH, B. Mixed Messaging . In: BYTE, March 1993, pp. 136-154.

[FER97] FERREIRA, S. Mechanisms for Information Classification and Retrieval in Electronic Mail. Ms.C. Dissertation. Porto Alegre, PUCRS, March 1997.(in Portuguese)

[FER97b] FERREIRA, S. A Query Language for Retrieving Information in Electronic Mail Environments. Accepted for publication in the XII Brazilian Symposium on Database Systems (SBBD97), Fortaleza, Oct. 1997.

[GOL92] GOLDBERG, Y; SAFRAN, M; SILVERMAN, W; SHAPIRO, E. Active Mail: A framework for Integrated Groupware Applications. In: Growpware'92 by D. Colleman, Morgan Kaufmann Publishers, 1992, pp. 222-224.

[HAM90] HÄMMÄINEN, H.; ALASUVANTO, J.; MÄNTYLÄ R. Experiences on Semi-Autonomous User Agents. In: Decentralized A. I.. Amsterdam, Elsevier Science Publishers B. V., 1990. pp. 235-249.

[HIL85] HILTZ, S.R.; TUROFF, M. Structuring Computer-mediated Communication Systems to avoid Information Overload. CACM, 28(7):680-689, July 1985.

[KOS91] KHOSHAFIAN, S. at al. Intelligent Offices. New York, NY, J.Wiley, 1991.

[KOS92] KHOSHAFIAN, S. Intelligent Databases. New York, NY, J.Wiley, 1992. pp. 293-346.

[LAI88] LAI, K.; MALONE, T.; YU, K. Object Lens: A "spreadsheet" for Cooperative Work. In: Readings in groupware and CSCW - Assisting Human-Human Collaboration., pp. 474-484.

[LAQ92] LAQUEY, T. at al. The Whole Internet User's Guide & Catalog. Addison-Wesley Publishing Company Inc., 2Ed., 1992.

[MAC88] MACKAY, W. Diversity in the Use of Electronic Mail: A preliminary Inquiry. In: ACM Transactions on Office Information Systems, 6(4):380-397, Oct. 1988.

[MAE94] MAES, P. Agents that Reduce Work and Information Overload. In: Communications of the ACM, 37(7), July 1994.

[MAL87] MALONE, T. et al. Intelligent Information-Sharing Systems. In: Communications of the ACM, 30(5):390-402, May 1987.

[MAL89] MALONE, T.; et al. The Information Lens: An Intelligent System for Information Sharing and Coordination. In: Technological Support for Work Group Collaboration, pp. 65-88, by M.H.Olson, 1989, Hillsdale, NJ:Lawrence Erlbaum.

[NIC92] NICHOLS, D.; GOLDBERG, D.; OKI, B.M.; TERRY, D. Using Collaborative Filtering to Weave an Information Tapestry. In: Communications of the ACM, Vol.35, No.12, December 1992, pp.61-70.

[PAL95] PALME, J.; KARLGREN, J. ;PARGMAN, D. Issues when designing filters in messaging systems. Report, Department for Computer and Systems Sciences, Stockholm University and Kungliga Tekniska Högskolan, Sweden - http://www.dsv.su.se

[POL88] POLLOCK, S. A Rule-Based Message Filtering System. In: Conference on Organizational Computing Systems. ACM Press, New York, pp. 21-30.

[REI93] REINHARDT, A. Smarter E-mail is Coming. In: BYTE, March 1993, pp. 90-108.

[ROB91] ROBINSON, M. Through a Lens Smartly. In: BYTE , May 1991. pp. 177-187.

[TAY92] TAYLOR, Dave; WEINSTEIN, Syd. The Elm Filter System Guide.gopher.emr.ca as /public/doc/elm-docs/filter-elm.

[TER92] TERRY, Douglas. Avoiding an Information Tapestry. In: Communications of the ACM, 35(12):61-70, Dec. 1992.

[WYL92] WYLE, M.F. A rule-based Electronic Mail Filter . Report, Institute of Technology, Zurich, Switzerland. Gopher at rohan.sdsu.edu Science Publishers B. V., 1990. pp. 235-249