Palo Alto Research Center

Browsing Electronic Mail: Experiences Interfacing a Mail System to a DBMS

Jack Kent, Douglas Terry, and Willie-Sue Orr

XEROX

Browsing Electronic Mail:
Experiences Interfacing a Mail System to a DBMS

Jack Kent, Douglas Terry, and Willie-Sue Orr

CSL-89-7 October 1989 [P89-00171]

Abstract: A database management system provides the ideal support for electronic mail applications. The Walnut mail system built at the Xerox Palo Alto Research Center was recently redesigned to take better advantage of its underlying database facilities. The ability to pose ad-hoc queries with a "fill-in-the-form" browser allows people to browse their mail quickly and effectively, while database access paths guarantee fast retrieval of stored information. Careful consideration of the systems' usage was reflected in both the database schema representation and the user-interface for browsing mail.

This paper appeared in the *Proceedings of the 14th International Conference on Very Large Data Bases (VLDB)*, Los Angeles, California, August 1988, 112-123.

CR Categories and Subject Descriptors: H.2.1 **[Database Management]:** Logical Design - *schema and subschema*; H.2.8 **[Database Management]:** Database Applications; H.3.3 **[Information Storage and Retrieval]:** Information Search and Retrieval; H.4.3 **[Information System Applications]:** Communications Applications - *electronic mail.*

Additional Keywords and Phrases: database, electronic mail, queries, browsing.

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Browsing Electronic Mail: Experiences Interfacing a Mail System to a DBMS

Jack Kent
Douglas Terry
Willie-Sue Orr

Computer Science Laboratory
Xerox Palo Alto Research Center

Abstract: A database management system provides the ideal support for electronic mail applications. The Walnut mail system built at the Xerox Palo Alto Research Center was recently redesigned to take better advantage of its underlying database facilities. The ability to pose ad-hoc queries with a "fill-in-the-form" browser allows people to browse their mail quickly and effectively, while database access paths guarantee fast retrieval of stored information. Careful consideration of the systems' usage was reflected in both the database schema representation and the user-interface for browsing mail.

1. Introduction

Electronic mail is used extensively within the Computer Science Laboratory at the Xerox Palo Alto Research Center. An average lab member may receive fifty messages per day, many of which he will file for future reference. Fast access to stored mail is essential. We have discovered that a database management system (DBMS) provides the ideal foundation for electronic mail applications [6]. The ability to pose ad-hoc queries allows people to browse their mail quickly and effectively, while database access paths guarantee fast retrieval of stored information.

Walnut is an electronic mail storage and retrieval system developed for the Cedar environment [12]. Users can pose mail queries through a "fill-in-the-form" browser; the browser interfaces to an entity-relationship database system [3]. Walnut was created both to satisfy our need for a mail system and as an experiment. The objective of the experiment was simple: to determine if the needs of an electronic mail application would be better met by a DBMS than by a file-system.

The majority of existing mail systems, such as those provided in UNIX™ [11] as well as our previous mail system [2], are file-system based. Justifying the additional overhead of transactions, a high-level data model, and a browsing tool requires an understanding of how users interact with mail readers. Our understanding came from several sources, including user-feedback, user information automatically gathered by the system, and performance measurements. This information proved invaluable in refining both the user-interface and the data model of Walnut.

Despite our assertions that electronic mail is an ideal client application for a DBMS, relatively few Cedar users initially switched from the file-based mail system to the original version of Walnut. There were many reasons:

- *Performance Degradation*

  Walnut operations ran slower than their counterpart file-system based mail operations. As an example, displaying a Walnut message-set generally requires an IO operation per message. Opening a folder of equal size in a conventional mail system is considerably faster since a file-system can stream in bytes faster than a DBMS streams in ordered tuples.

- *Questionable benefits of transactions*

  To many Walnut users, it was unclear how transaction support was truly beneficial in our environment. For example, the value of concurrency control is minimal since most mail databases are private. Even the value of

crash recovery is not immediately apparent to most mail users.

- *No query facility*

  The inability to pose ad-hoc queries, such as show me all messages from time X to time Y sent by user Z, muted the value of the DBMS.

- *Limited data model*

  Our model of mail was inadequate. Walnut initially treated mail messages as uninterpreted text rather than parsing fields such as subject, sender, date, or recipient and including them in the data model.

To improve the utility of Walnut, we added a browser that allows users to query a mail database for messages satisfying a given set of attributes. We also the revised database schema that fully supports this interface. In this paper, we relate our experiences.

In section two, we discuss the Walnut functionality and user-interface. Section three follows with a description of how Walnut's database schema was improved based on early experiences. In section four, we present some proposed extensions to the data model and methods for comparing alternative data models. Section four also summarizes information gathered about how users pose queries and discusses how usage patterns can influence the design of mail systems.

2. An Overview of Walnut

2.1 The Conceptual Model

The Walnut data model includes two classes of objects: messages and message-sets. Message-sets are named folders for categorizing messages; they can be created, deleted or enumerated. Messages are unnamed and are added to, deleted from, or moved between message-sets. Messages belong to one or more message-sets.

Two message-sets are treated specially: Active and Deleted. The NewMail operation gathers mail from the mail transport service and adds it to Active. The Deleted message-set contains all messages that belong to no other message-set. Expunge is a special operation that irrevocably discards Deleted messages, thereby reclaiming the storage they occupied.

2.2 The User-Interface

The Walnut user-interface supports three types of windows: the control window, message-set display windows (one per message-set), and message display windows (one per message). Figures 2-1, 2-2, and 2-3 contain representatives of these types of windows. Each window provides a menu of operations specific to its type. For instance, the NewMail and Expunge operations can be found in the control window (Figure 2-1). Clients invoke an operation by using a mouse to select a menu entry.

The Walnut control window contains a list of named message-sets. Using the mouse to click on one of these message-set names causes a message-set display window to be created. A message-set window contains a line summarizing each message in the message-set. The menu at the top of this window includes operations for displaying, moving, and deleting messages. Clicking on an entry in a message-set window causes a message display window to be created containing the textual contents of the selected message.

2.3 The Query Tool

2.3.1 Specifying queries

Figure 2-4 shows the message browsing form used to specify queries on a mail database. There is a slot in the form for each of five attributes of a mail message: the message-set that it belongs to (one of possibly many), the sender of the message, the recipient(s) of the message, the carbon-copy (cc) recipients of the message, the date that the message was sent, the subject of the message, and the actual text of the message. (Note: including the complete message text as a field in the form allows full text searches.) The user constrains the value of an attribute by filling in the slot corresponding to that attribute. After initiating the query, any mail message that "matches" the form is returned; the user can then perform further operations on the matching messages.

Exactly how the filled-in text in the message form is "matched" against mail attributes is determined by user-selected filters. The filter currently being used for a given attribute is depicted in a box to the left of the attribute. In Figure 2-4, all of the attributes specify the "Do What I Mean" (DWIM) filter. Alternative filters can be chosen by clicking with a mouse over the filter button.

As of this time, the mail browser provides nine pattern-matching filters: exact, prefix, wildcard, regular

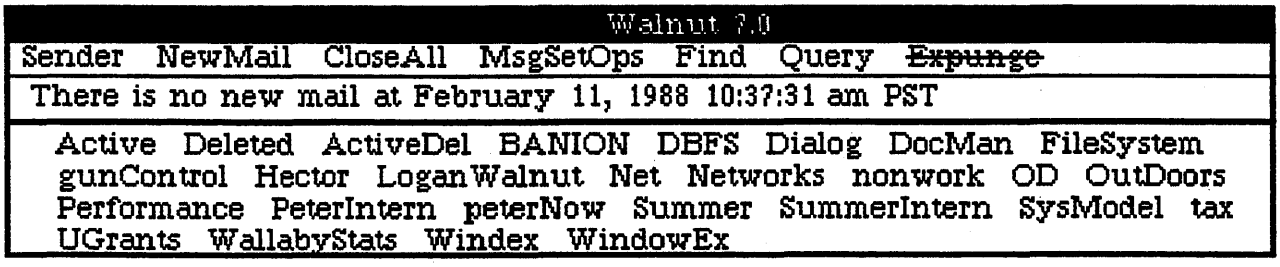

Figure 2-1. A Walnut Control Window
Select a message-set button to open a message-set display window.

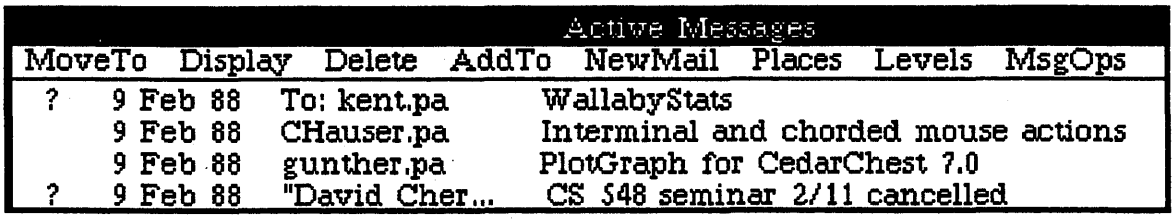

Figure 2-2. A Walnut Message-Set Window.
Select a message-header to open a message.

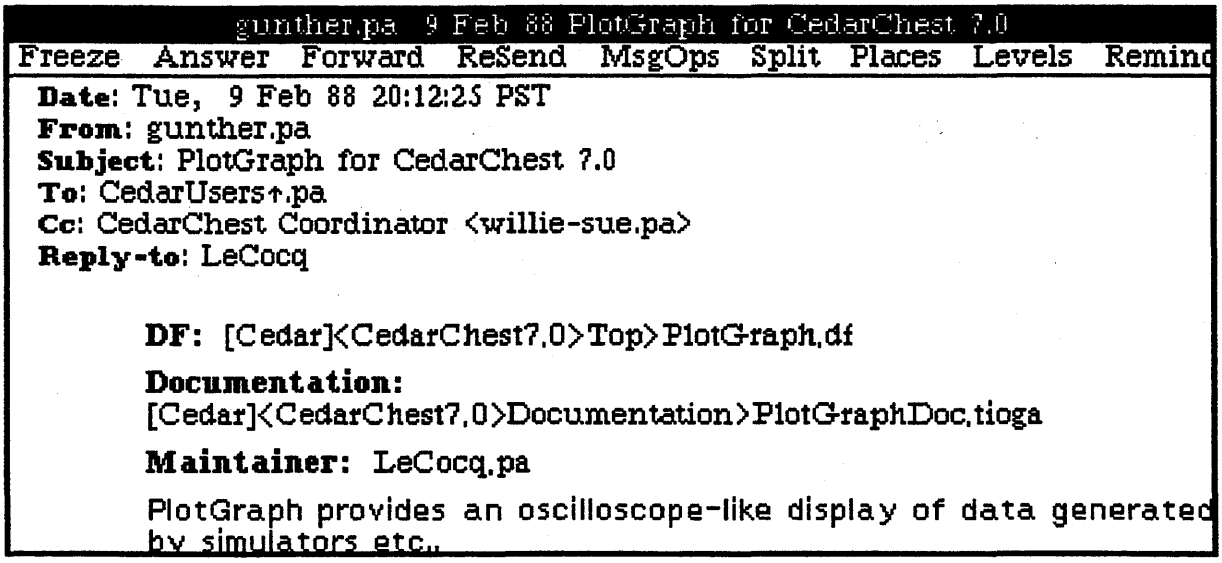

Figure 2-3. A Walnut Message.

expression, soundex, subrange, date, date range, and DWIM. Their meanings can be surmised from their names. For instance, the "date range" filter assumes the user has typed a pair of date/times separated by a dash, where each date can be interpreted by an intelligent date parser. Very often, the type of filter desired by the user can be inferred from the pattern alone. To this end, the DWIM filter has been provided and is the default. More specifically, when DWIM is specified, the browser looks for special characters within the pattern and attempts to infer the filter type from context. If the text contains a "*", for example, then wildcard pattern matching is used.

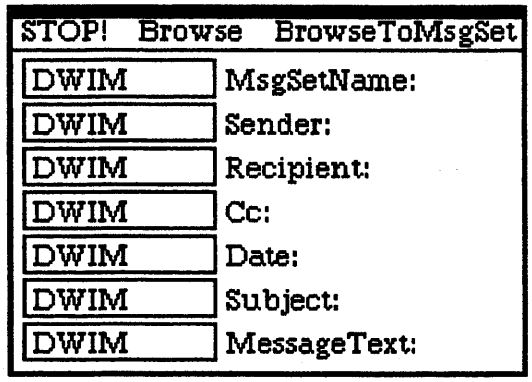

Figure 2-4. The Walnut Query Tool.

Figures 2-5, 2-6, and 2-7 give examples of three different forms and the intended queries they represent. Figure 2-5 depicts a browsing form to find all mail in my database sent since February 1, 1986 from someone named Hagmann; note that for the Date field, DWIM selects date-range pattern matching, whereas it selects textual prefix pattern matching for the Sender field. The form in Figure 2-6 allows all mail sent to the RiverRats distribution list to be retrieved. The form in Figure 2-7 finds messages that were sent from someone whose name sounds like "Tearee", such as "Terry", and that contain the word "suggestions" somewhere in the message text.

2.3.2. Initiating a query and viewing the results

Initiating a query is done simply by clicking one of the two browse buttons in the menu of the query tool (Figure 2-4): Browse or BrowseToMsgSet. The operations differ in how they present the results of a query.

The Browse presentation mode is a lightweight mechanism for viewing the results of a query. A result appears in the window below the form window as five properties (called an item). Collectively these five properties uniquely identify a message. Mouse selecting an item opens a message window and displays the corresponding message text.

The BrowseToMsgSet operation moves the results of a query to a designated Walnut message-set. While slower than Browse (since it involves updating the database rather than simply displaying a message), this operation is useful for defining new categories, splitting a category, merging two categories, or even cleaning up the database. A common use for BrowseToMsgSet is cleaning up the Active message-set by specifying a query that moves all unwanted active mail to Deleted.

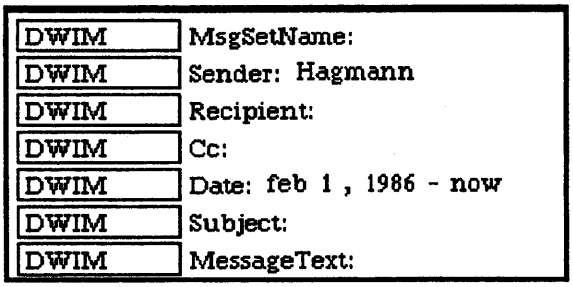

Figure 2-5. A form to find mail sent between February 1, 1986 and now, from someone named Hagmann.

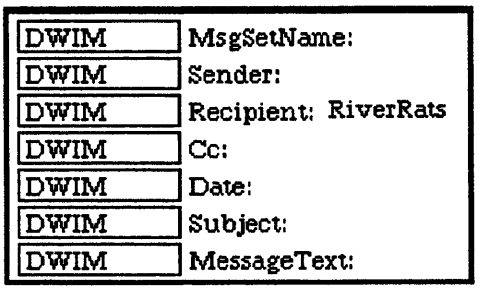

Figure 2-6. A form to find mail sent to the RiverRats distribution list.

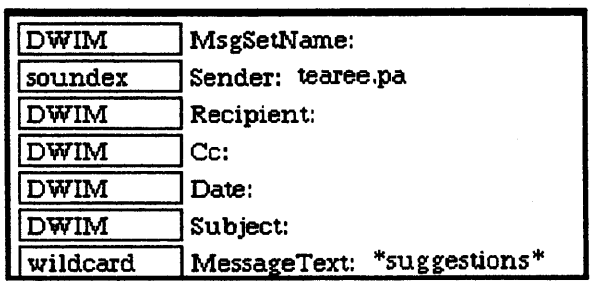

Figure 2-7. A form to find all mail sent from someone whose name sounds like 'Tearee' with the word "suggestions" occurring somewhere in the message.

2.4 Comparisons to Some Other Mail Systems

All mail systems have some basic similarities. In a typical mail system, each user is associated with a mailbox. A sender, assisted by a user agent, composes a message and identifies its recipients. The message is then deposited with the transport system which

	UNIX	DragonMail	LENS	Walnut
Support for categorizing mail	none	mail is grouped by conversations	semi-structured messages, rules, sub-typing	DBMS support, message fields
browser support	editor	equality matching on subject, date and sender	rules	wide variety of filters
conversation support	none	unique ID context kept	topic, ordered by time rcvd.	topic, ordered by time sent
shared mail support	NetNews	same as public mail	ANYONE servers	public mail database archive
time to process new mail	very fast	fast	depends on number of rules	slow
fast access paths	no	no	no	yes
support for removing unwanted msgs	none	space deallocated when conversation is terminated	rules can be defined for filtering	query operation aids cleanup, Expunge operation

Table 2-1. A comparison of mail reading systems.

forwards it to each recipient's mailbox. Tools are typically provided for categorizing messages after they have been received, usually into predefined file-names called folders.

Table 2-1 attempts to summarize the differences between four mail systems: UNIX Mail [11], Dragonmail [5], Information Lens [9], and Walnut. The mail system commonly used in UNIX is chosen as a representative of conventional mail readers. Dragonmail provides additional support for managing conversations. The Information Lens (sometimes referred to simply as LENS) uses semi-structured message types and rules for filtering incoming mail.

In terms of support for categorizing mail, LENS appears to be the most versatile. In LENS, a rule's predicate can be based on the message type or the contents of the message header. A rule's action can be, for example, to store the message in a given folder. Using the type hierarchy, a wide variety of message inter-relationships can be represented.

For browsing, the LENS user can define a browsing rule such as "if mail from Joe then display". The Walnut user, on the other hand, has a more convenient fill-in-the-form browsing interface and can choose from a wide variety of predefined filter types.

Support for conversation browsing is also important. LENS and Walnut interpret the message's subject field as a conversation ID, and then display these message in a linear temporal order (by time received or by time sent). This approach has two problems: (1) multiple conversations may be associated with a single topic and (2) displaying a conversation in date order may be misleading since it gives the user no idea of the context in which the message was written, that is, which messages were read to instigate a given reply. Dragonmail solves this problem by associating a unique ID with each conversation. Users browse conversations not in temporal order, but in the context in which they were written (a conversation is represented by a directed acyclic graph).

The various systems also differ in how they support public mail (mail that is to be widely read). Many mail systems use distribution lists that are maintained by the mail transport service [1]; these allow client applications to treat public mail exactly like private mail. Unfortunately, maintaining public mail recipients using distribution lists is wasteful of space since each recipient receives a private copy of the message. For this reason, systems like NetNews [8] store public mail in a logically centralized repository called a bulletin board. Walnut supports both bulletin boards (public databases) and distribution lists. LENS offers a compromise solution: the ANYONE server. The ANYONE server functions as an intermediary between the sender and potential recipient. Users register interests (via rules) with the ANYONE server. In contrast to distribution lists, "interests" can be very detailed (as the interests are rules) and the technique scales better (as mail is forwarded at a more local level, there is less network bandwidth consumed).

Conventional mail systems provide little or no support for one very important electronic-mail operation: removing junk mail. The LENS system's rules provide some help for the problem. In Walnut, we have discovered that queries invoked with the BrowseToMsgSet operation can be used quite effectively to remove junk mail from one's Active message set.

In many ways, the file-based mail systems described above suffer from performance and usability limitations. Browsing is often inconvenient or impossible. Retrieval is slow due to the lack of fast access paths. And data consistency is not guaranteed. Many of these deficiencies are what motivated us to build Walnut, an electronic mail system layered over a DBMS.

3. The Walnut Implementation

3.1 The First Schema

In the first implementation of Walnut, the mapping from conceptual model to entity-relationship schema was fairly simple. This original schema is presented in a high-level language in Figure 3-1. Both messages and message-sets are represented as entities (represented by the two domains: Message and MsgSet). A single relation, MessageSetToMessage, serves to associates messages with message-sets. Additionally, there is an index on the concatenation of the first two fields of the MessageSetToMessage relation.

```
Domain Message

Domain MsgSet

Relation MessageSetToMessage
    msgSet: MsgSet
    date: time
    message: Message

Index on Relation MessageSetToMessage
  [msgSet, date]
```

Figure 3-1. The first Walnut schema.

```
Domain Message

Domain MsgSet

Relation MessageSetToMessage
    msgSet: MsgSet
    date: time
    sender: text
    recipient: text
    cc: text
    subject: text
    message: Message

Index on Relation MessageSetToMessage
  [msgSet, date]

Index on Relation MessageSetToMessage
  [date]

Index on Relation MessageSetToMessage
  [sender, date]

Index on Relation MessageSetToMessage
  [recipient, date]

Index on Relation MessageSetToMessage
  [cc, date]

Index on Relation MessageSetToMessage
  [subject, date]
```

Figure 3-2. The revised Walnut schema.

3.2 The Revised Schema

To fully support the mail browser, the Walnut schema was later revised, as in Figure 3-2. This new schema exposes far more of the semantics associated with mail. Properties that were previous left uninterpreted in the mail message are parsed and

exposed in the schema. Indices were defined on each of the five new fields: date, sender, recipient, cc, and subject. All of these indices are keyed on the concatenation of some message attribute with the message's date.

3.3 Selecting An Access Path

Due to the lack of a sophisticated query optimizer in our DBMS and the complexity of optimizing queries with a variety of pattern matching filters, the Walnut query tool was forced to perform its own optimization (in this case, index selection). Initially, we used a simple heuristic for index selection: find the field in the form with the longest character string and use this field's index. The reasoning was that prefix pattern matching is most commonly used and that the longest character string will most likely provide the fewest entries in an index.

This strategy had worked well for a similar browser used to query databases of names and phone numbers. It did not work well for Walnut databases since:

- A mail form contains dates as well as text.
- Walnut text fields are not as uniformly distributed as, for example, phone numbers.

Moreover, a mail database is typically much larger than our phone database, so poor index selection can add greatly to the cost of a query.

We ultimately chose to implement a more intelligent optimization strategy. The new strategy takes into account both the types of pattern matching filters being used and estimated selectivity factors. Date selectivity is estimated by the size of a date range and the average number of messages stored per day. Entity selectivity is computed by the number of tuples that reference a particular entity.

4. Evaluating the System: Usage Patterns, Schema Design, and Performance Considerations

4.1 User feedback

Based on an informal sampling of user opinion, the second release of Walnut was a qualified success. Users reported that manual message-set categorization was usually unnecessary (automatic message categorization sufficed) and retrieval of old messages was fast, considerably faster than displaying a message-set.

We informally probed users as to how we might improve the Walnut data model. A few suggested we enhance the schema, for example, to expose keyword information. Others suggested we simplify the data-model (who needs "cc"?) or at least allow indices to be declared on a per-user basis. This lack of consensus reinforced something we knew all along: we needed a more formal way to evaluate how the system was being used.

Sending out a questionnaire was one option. This is a widespread method for understanding how users work with complex systems (or user-interfaces). But the information we sought was basic enough (how often did users fill in this field? what did they fill it with?) that we could get all our information simply by analyzing a log of user's queries.

Many of the new features that have been proposed to improve query processing come at the expense of mail update operations. To better assess the tradeoffs involved, we decided to examine both how users pose queries and the performance impact (and storage cost) of past and proposed schema designs on mail update operations.

4.2 Insights into the DBMS Burden

4.2.1 Alternate schemas

Complicating the mail schema places additional burden on new mail retrieval and expunge operations. Quantifying this burden is helpful. For this reason, we compared four mail schemas in two important ways. The first comparison is based on storage required by a 1082 message database, whose characteristics are described in Table 4-1. The second comparison is the time required to read 45 new messages into the database.

Descriptions of the four alternative schemas that we tried are included below:

The **Old** Schema is the first Walnut schema summarized in Figure 3-1.

The **Current** Schema is the revised schema summarized in Figure 3-2. There are six indices, all concatenated with dates.

The **Normalized** Schema better exposes the semantics of the "recipient" field and "cc" field to the user. Mailbox names (those present in the recipient, cc, and send fields), as well as the subject field, are represented as entities rather than uninterpreted strings. This provides for more intelligent browsing

Size of database in messages: 1082

Number of subject entities in database: 850
Number of address entities in database: 899
Number of keyword entities in database: 1547

986 keywords occur once
297 keywords occur twice
108 keywords occur three times
156 keywords occur more than three times

Table 4-1. Characteristics of benchmark database.

Schema	Size (kilobytes)	NewMail (seconds)
Old	1590	18.3
Current	3082	28.8
Normalized	3338	41.4
KeyWords	4618	52.8

Table 4-2. Comparisons of alternative schemas.

(users could now ask for all mail where they were specified as a recipient, without needing to specify a wildcard filter).

The **KeyWord** Schema is an attempt to assess the potential burden of keyword processing on new mail. Keywords are represented as entities within the schema, and are extracted from the subject field using a reasonable stop-word list. (Note: This schema was built on the normalized schema.)

4.2.2 Observations

We expected a significant storage savings from the current schema to the normalized schema since unique ID's would be replacing many of the long text fields. Table 4-1 supports our intuition. Compression of sender, cc, and recipients fields to address entities results in almost a 4:1 storage savings (1082 messages * 3 address entities per message : 899 address entities). The other domains accounted for far less savings. From the meager compression of subject field to subject entities, we can conclude that there aren't many conversations in the benchmark database. Keyword entities also did not offer much opportunity for compression, we see in Table 4-1 that the vast majority of keywords occur only once.

Table 4-2 confounds our intuition. Note the database size increase (we expected a decrease) from the current schema to the normalized schema. For this reason, we checked to see exactly how the database pages were being allocated and discovered two things:

(1) Both the "to" and "cc" field of a message may contain many names. And to represent a one-to-many relationship between a message and its recipients (or its cc's), a new relation **R** must be created to maintain this association. For each message containing N names in a given list (cc or recipient), N tuples must be added to **R**.

(2) The underlying DBMS maintains indices on all entity names, whether or not they were declared. Thus, for many fields, we were forced to pay twice for indices; for example, indices were kept on both subject and (subject, date).

The last column in Table 4-2 gives the time required to add 45 new mail messages to the database. As expected, this cost increases with the complexity of the schema.

4.2.3 Suggestions

Indexing on each keyword in the subject of a message, is expensive. Alternative solutions have been proposed:

- *Only keyword process mail that will be referenced in the future.* That is, index a mail message by its keywords not when it enters Active but when it is moved to a stable message-set. This saves processing since a considerable amount of mail is deleted immediately as it comes in from Active; deleted mail can justifiably not be keyword processed.

- *Perform keyword processing as a batch operation.* Keyword processing can be performed as an off-line operation, just like Expunge.

- *Try alternative access paths.* Signatures [7] may be preferable to indices, especially given their minimal space consumption.

4.3 Insights into User's Queries

4.3.1 Usage Information

Understanding how people use the browser is helpful in many ways. Unused fields can be identified and removed from the schema. Candidate fields for further refinement can be exposed. Common fill-ins for a form can suggest user-interface speed-ups and prefetching information. Overall, we can assess the

TYPE	Frequency	Cumulative
(date,sender)	14%	14%
(sender)	9%	23%
(subject)	8%	31%
(subject, date)	7%	38%
(sender,msgset)	7%	45%
(subject,sender)	5%	50%
(date,sender,msgset)	5%	55%
(subject,msgset)	4%	59%
(subject,date,sender)	4%	63%
(subject,date)	4%	67%

Table 4-3. Frequency of Query Types.

Sender	Subject	Date	Recipient	MsgSet	MsgText	Cc
9%	8%	2%	2%	<1%	<1%	0%

Table 4-4. Frequency of One-Fill-Ins.

Sender	Date	MsgSet	Subject	Recipient	Cc	MsgText
59%	44%	44%	40%	23%	15%	3%

Table 4-5. Frequency that a Form Contains a Given Field.

tradeoffs between facilitating query processing and complicating mail update.

Shortly after the new version of Walnut was released, we gathered usage information by analyzing logs of users' queries. During the testing session, 713 queries were run over 50 days by 18 users. Discounting weekdays, this amounts to 20 queries per day. Four users accounted for over half of these queries.

4.3.2 General Query Types

One way to distinguish amongst query types is according to whether (or not) a given field in the form is used. Table 4-3 lists the nine most frequent queries. Referring to Table 4-3, we see that ten query types (out of 128) account for 67% of the queries. Note that the majority of types are two-fill-ins (6) with 2 three-fill-ins and 2 one-fill-ins. Filling in two entries, it appears, almost always provides an adequate filter.

Table 4-4 gives the measured frequencies when only a single field in the query form is used, while Table 4-5 gives the frequencies that a given field appears in a form. One apparent inconsistency is that MsgSet and Date occur with low frequency in Table 4-4, and with high frequency in Table 4-5. The former is easily explained: The Walnut control user-interface (Figure 2-1) provides message-set buttons that enable a user to easily open a message-set window (as in Figure 2-2) on a given message-set; these operations are not accounted for in Table 4-4. As for why date is never a lone fill-in, we suspect temporal associations are rarely made without additional knowledge.

According to the gathered usage data, the recipient and cc fields rarely are filled in by themselves or even with date. For this reason, both these indices could be dropped, at least for the private mail databases. (Although we have no statistics on public mail database browsing, we suspect that public mail browsers would likely interrogate the cc and recipient fields a great deal.)

4.3.3 Use of the Date slot

We were interested in how people temporally browse. In particular, how long is the interval when a date range is specified, and how far back in time is the start?

Figure 4-1 gives the number of observed queries for different length intervals of days. By far, the majority of date intervals are from one to two days.

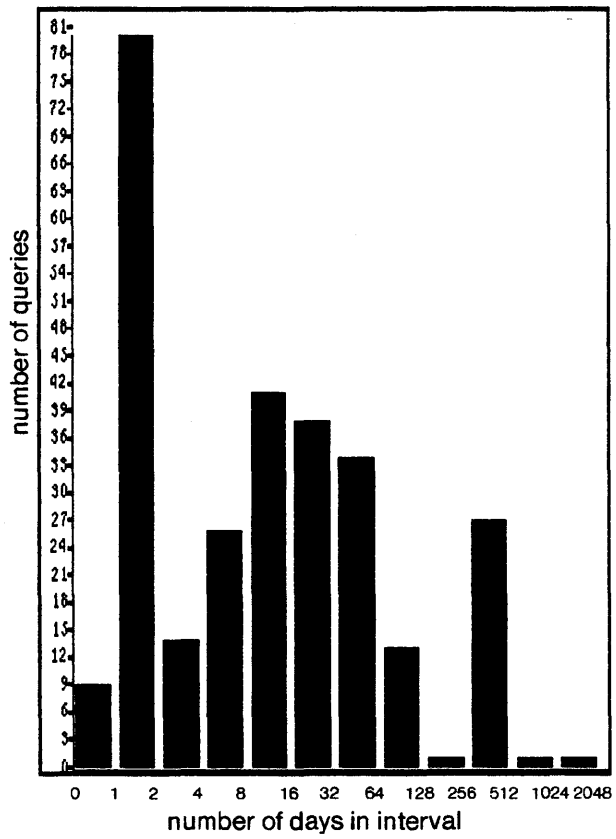

Figure 4-1. Histogram of Date Intervals.

The (relatively) large number of queries with a time interval of 256 - 512 days can be explained by the large number that start on either January 1 or "one year ago".

Figure 4-2 clearly shows the mail users' bias towards querying about recent events. Nearly one half of all queries that included a date range specified a "start time" of less than a week ago. For this reason, we believe mail systems should prefetch recent mail on start-up and cache messages to reflect the temporal bias.

Our experience has also shown that users tend to specify date ranges in a very systematic way, one that could likely be exploited by a better user-interface design. Ranges such as "the beginning of this [temporal unit] to now" accounted for the majority of user's ranges. Allowing the user to mouse select a range (from a pre-defined menu) would be preferable to forcing the user to type in these common ranges as he must do in Walnut.

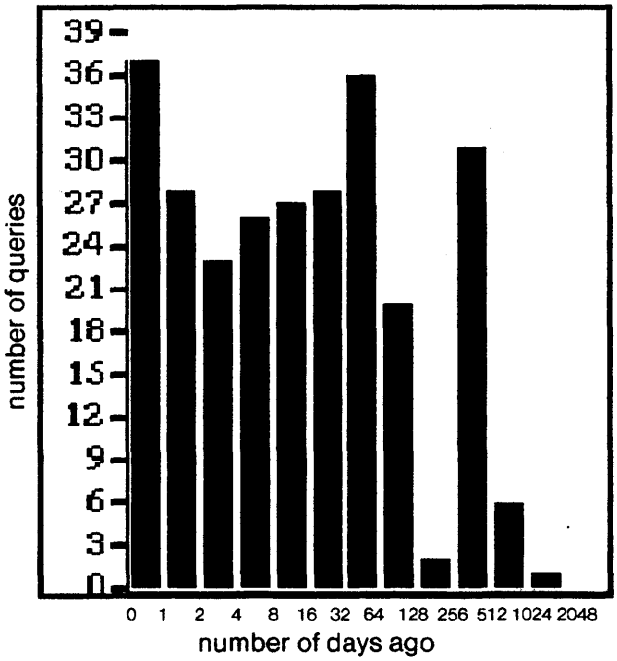

Figure 4-2. Histogram of Query Start Dates.

4.3.4 Use of the Subject Slot

When a Walnut user responds to a mail message with subject field "foo", the system fills in the subject field of his response with "re: foo". Thus, conversation browsing on message "foo" is done by first browsing for subject "foo" and then for subject "re: foo".

We found that the mail system has rarely been used for conversational browsing (at least not in the way we described). When the subject slot was filled in, it was prefixed by "re" only 6% of the time. If nothing else, this leads us to question the importance of specialized support for conversations.

Users browse for keywords in the subject field by using wildcard queries. We found that, when the subject slot was filled in, 31% of the time it was prefixed by a wildcard. Processing these queries is slow. For this reason, we believe keyword support is justified and keywords should be interpreted by the DBMS. The next Walnut release will incorporate keywords.

5. Conclusions

Based on our experiences, electronic mail will represent a considerable percentage of communication in the "office-of-the-future". Our mail system, Walnut, is one of the first to exploit database management technology. Though it manipulates an entity-relationship database [4], the particulars of the data model are not important. Conventional relational database management systems would have served us as well.

The Walnut case-study offers compelling evidence for layering an electronic-mail application over a database management system, especially given the features now available in commercial relational systems (features that we did without). The majority come equipped with fourth-generation languages (for building forms-based user-interfaces), triggers (for implementing a reminder feature) and long strings (for efficiently storing message text in the database).

Our investigation has revealed that users classify mail in a very systematic and structured manner. For this reason, we question the value of certain approaches to personal mail browsing. Keyword search on the body of a message may not be worth the additional burden; consider that MsgText was never a lone fill-in for a browser form and that less than 3% of Walnut mail queries filled in MsgText. Contrast this usage with keyword browsing in the subject field (described earlier). Our observation seems to dovetail with the favorable results reported by Salton [10] for an automatic indexing system that uses only abstracts (not full text). However, keep in mind that our usage information was gathered from a relatively small set of novice users; we hope to repeat our experiments now that our users are more numerous and sophisticated.

Given a mail browsing facility, ad hoc mail categorization (like message-set or folder) may be less necessary. Consider that almost 70% of all queries that included message-set were filled in with Active or Deleted (and the vast majority of these were two fill-ins). In fact, some Walnut users now report that they only work with three (logical) message-sets: Active, Deleted and Archived.

We have gained important insights into how mail users pose queries, and under what conditions the boon of additional functionality (and complicating the schema) may justify compromising update performance (or increasing storage costs). Insofar as the DBMS we used is not unusual and the model of mail we chose is very simple, we believe our insights will be of interest to both researchers and implementors.

In the future, we plan to release a new version of Walnut, based on the keyword-conceptual model discussed in section 4.2.1 and using insights gleaned from examining users' queries. More intelligent filtering will also be used, especially as it relates to mail addresses and keywords.

6. Acknowledgements

Jim Donahue deserves much of the credit for Walnut. Carl Hauser and Polle Zellweger provided valuable comments on early versions of this paper. Margaret Butler gave tremendous help on the final revisions of the paper.

7. References

[1] Birrell, A., Levin, R., Needham, R., and Schroeder, M. Grapevine: An exercise in distributed computing. *Communications of the ACM*, April 1982, pp. 260-274.

[2] Brotz, D.K. Laurel Manual. Tech Report CSL-81-6, Xerox Palo Alto Research Center, Palo Alto, Calif., May 1981.

[3] Cattell, R. G. G. Design and Implementation of a Relationship-Entity-Datum Data Model. Tech Report CSL-83-4, Xerox Palo Alto Research Center, Palo Alto, Calif., May 1983.

[4] Chen, P. The entity-relationship model – toward a unified view of data. *ACM Transactions on Databases*, March 1976, pp. 9-36.

[5] Comer, D. and Peterson, L. Conversation-based mail. *ACM Transactions on Computer Systems*, November 1986, pp. 299-319.

[6] Donahue, J. and Orr, W.-S. Walnut: Storing electronic mail in a database. Tech Report CSL-85-9, Xerox Palo Alto Research Center, Palo Alto, Calif., April 1986.

[7] Faloutsos, C. and Christodoulakis, S. Description and performance analysis of signature file methods for office filing. *ACM Transactions on Office Information Systems*, July 1987, pp. 237-257.

[8] Horton, M. How to read the network news. 4.3 Berkeley Software Distribution, April 1988.

[9] Malone, T., Grant, K., Turbak, F., Brobst, S., and Cohen, M. Intelligent information-sharing systems. *Communications of the ACM*, June 1987, pp. 390-407.

[10] Salton, G. Another look at automatic text-retrieval systems. *Communications of the ACM*, June 1986, pp. 648-656.

[11] Shoens, K. Mail Reference Manual. UNIX Programmer's Manual. 4.1 Berkeley Software Distribution, vol. 2, Berkeley, Calif., 1979.

[12] Swinehart, D. C., Zellweger, P. T., Beach, R. J., and Hagmann, R. B. A structural view of the Cedar programming environment. *ACM Transactions on Programming Languages and Systems*, October 1986, pp. 419-490.