Lifestreams: Organizing your Electronic Life*

Eric Freeman and **Scott Fertig**
Yale University
Department of Computer Science
New Haven, Connecticut, 06520

Abstract

The "Web," an attempt to organize the electronic world, has become the focus of a wide range of research activities from agents to information retrieval to virtual reality systems, and is a central focus of work presented at this symposium. Our work considers a more humble task: that of organizing our own electronic lives. We introduce a new metaphor, *Lifestreams*, for dynamically organizing a user's personal files, electronic mail, schedules, rolodex and financial data. In this paper we describe our prototype Lifestreams system and in the process cover most of the topics of this symposium, including indexing, information retrieval, user interfaces, multi-source integration and multimedia.

1 Introduction

The work of many participants in this symposium attempts to help users navigate the thicket known as the "Web". By contrast, Lifestreams is first and foremost an approach to organizing a user's personal files. We contend that today, just managing one's own electronic world can be a frustrating task for most computer users, requiring too many separate applications, too many file transfers and format translations, and the construction of organizational hierarchies that too quickly become obsolete. What is needed is a metaphor and system for organizing the electronic "bits of paper" we all so easily collect, whether they come to us in the form of electronic mail, downloaded images, pages gathered from the Web, or scheduling reminders.

Lifestreams is such a system. It uses a simple organizational metaphor, a time-ordered stream of documents, to replace conventional files and directories. *Stream filters* and software agents are used to organize, locate, summarize and monitor incoming information. Lifestreams subsumes many separate desktop applications to accomplish the most common communication, scheduling, and search and retrieval tasks; yet its machine-independent, client-server architecture is open so that users can continue to use the document types and viewers/editors they are accustomed to. In this paper we describe our initial efforts in defining and building Lifestreams; in the process we touch on the majority of research issues of the symposium including indexing, information retrieval, user interfaces, multi-source integration and multimedia. We begin by describing the Lifestreams metaphor and then in section 3 describe the architecture and performance of the current prototype. In section 4 we describe the way several common computer tasks are performed with Lifestreams. Section 5 discusses the search and indexing choices we made for Lifestreams, and how these relate to our current research.

2 What is Lifestreams?

Think of your *lifestream* as a diary of your electronic life; every document you create is stored in your lifestream, as are all the documents other people send you. Technically, a lifestream is a time-ordered stream of documents. The tail of your stream contains documents from the past, perhaps starting with your electronic birth certificate. Moving away from the tail, and toward the present, your stream contains more recent documents such as papers in progress or the latest electronic mail you've received—other documents, such as pictures, correspondence, bills, movies, voice mail, and possibly software, are stored in between. Moving beyond the present, and into the future, the stream contains documents you *will* need: reminders, your meeting schedule, and todo lists.

Users interact with Lifestreams via five operations: `new`, `clone`, `transfer`, `find` and `summary`. `New` and `clone` are used to create documents. `New` creates a unique document and adds it to your stream. `Clone` takes an existing document and creates a duplicate that is added to your stream. `Transfer` copies a document from your stream to someone else's stream. `Find` prompts the user for a search query, such as "all email I haven't responded to," or "all faxes I've sent to

*This work was partially supported by ASSERT grant F49620-92-J-0240.

Schwartz" and creates a *substream.*

Substreams, like virtual directories [Gifford *et al.*, 1991; Manber and Wu, 1993], present the user with a "view" of a document collection. In our case, this view contains all the documents that are relevant to our search query. Substreams differ from conventional directory systems in that, rather than placing documents into fixed, rigid directory structures, they create virtual organizations of documents from the stream. That is, documents aren't actually stored in the substream, rather, the substream is a temporary collection of documents that already exist on your stream. Given this, the collection of documents in two substreams may overlap. Moreover, substreams can be created and destroyed on the fly without affecting the organization provided by the stream or any other substreams. Substreams also have a dynamic nature. If you allow a substream to persist, it will collect new documents that match your search criteria as they are added to your stream. For example, a substream created with the query "find all documents created by other people" would subsume your mailbox and automatically collect mail as it arrives. Similarly, a substream created from "all electronic mail I haven't responded to" would act as a special mailbox that only contains unanswered mail.

Our last operation, `summary`, takes a substream and compresses it into an overview document. The content of the overview document is dependent on the type of the documents in the substream. For instance, if the substream contains the daily closing prices of all the stocks and mutual funds in your investment portfolio, then the overview document may contain a chart displaying the historical performance of your securities along with your net worth. On the other hand, if the substream contains a list of tasks you need to complete, the overview document might display a prioritized "to-do" list for you.

2.1 Why time-based ordering?

Given that we use substreams to organize our documents, why bother with the underlying time-based ordering? There are several compelling reasons: the stream adds historical context to our document collections as all documents eventually become read-only (in the past), set in stone for history, as is the order and method in which they were created. Like a diary of our electronic lives, streams document our work, correspondence, and transactions. This historical context can be crucial in an organizational setting [Cook, 1995] and notably, most current software systems do little to track when, where, and why documents are created and deleted, something that was taken for granted in the paper-based world.

The present portion of the stream acts as a workspace, holding "working documents"; this is also typically where new documents are created and where incoming documents are placed. Most newly created documents hang around in the present for some time before they become read-only and pushed off into the past, being automatically archived in the process.

The future portion of the stream is particularly interesting because it allows documents to be created in the future (unlike the paper-based world, computers can defy space and time). Allowing future creation gives us a natural method of posting reminders and scheduling information. How? Our system allows the user to dial to the future and deposit a new document, let us say a reminder for your Mom's birthday. When your Mom's birthday arrives the note appears in the present and reminds you. We will see how this actually happens in the user interface shortly. There are other uses for the future, for instance in maintaining a workgroup meeting schedule and scheduling agents. We will visit some of these, in passing, in later sections.

2.2 Embedded Computation: Agents

The term "agent" (personal agent, intelligent interface, personal assistant, knowbot, guide, etc.) has recently received a considerable amount of attention in the research community as well as the commercial world. In our system we use the term to describe any embedded computation that can be used to extend the functionality of Lifestreams. Lifestreams uses three kinds of embedded computations: personal agents, document agents and stream agents. Personal agents are typically attached to an user interface and can automate tasks or learn from the user's interactions with Lifestreams. Document agents live on documents and are spawned by various events (e.g., the first time a document is accessed). Stream agents are attached to streams and execute whenever the stream changes in some way (e.g., a new document appears on the stream). We explore agents in succeeding sections, seeing how they are implemented and how they can be used.

3 Our Research Prototype

Our research prototype consists of a client/server architecture that runs over the Internet. Although one might think a PC (and its file system) the natural place to first develop Lifestreams, we expect access to the Internet to be ubiquitous in the next decade and expect that a person's lifestream will be accessible from any computational platform capable of running Lifestreams clients, (*viewports*). Our server currently runs on UNIX platforms and is the workhorse of the Lifestreams system. Each server handles one or more streams—storing all stream documents, substreams, and agents. Each viewport is a client of the server, providing the user with an interface to the document collection. Viewports communicate with the server via remote procedure call and data is passed between the viewport and server in a machine independent form (with XDR). We believe the viewport interface will differ radically over the range of computing platforms, from set-top boxes to high-end workstations; however each viewport should provide the functionality of the basic operations. We have currently implemented two client viewports on the opposite ends of the computational spectrum: one for Sun workstations and one for the Newton PDA. The workstation version provides an interesting graphical interface and implements the full-range of Lifestream functionalities, while the Newton version implements a minimal method of accessing streams— given our lack of space, we concentrate on the UNIX version in this paper (information on the Newton version can be found in [Freeman, 1995].)

3.1 The Viewport

Our workstation viewport can be seen in figure 1. The viewport is implemented in C and Tcl/Tk, extended with our own command set. In the interface, we have attempted to preserve the stream-based metaphor. The user can browse her stream by scanning forward or backward over the documents via the scroll bar in the lower left-hand corner. Note that the scroll bar is annotated with the dates of the current range of visible documents on the top and the first and last dates in the stream on the bottom end points. The user may also slide the mouse pointer over the document representations to receive a "glance" view of the contents of each document. The glance view displays a few descriptive attributes such as the date and time of creation, a descriptive icon that represents the document content type, a summary line, who the message is to and from (if the document is a mail message), and the first couple of lines of the message. This last piece of information is currently supported only for text documents, although we would like to support other types (for example, thumbnail sketches for images).

Color and animation are used to indicate important features of documents. The borders of unread documents are colored red (or any user defined color), the borders of writable documents are made thicker, and open documents are offset to the side to indicate they are being edited. Incoming documents slide in from the left side of the user interface to alert the user and newly created documents pop down from the top of the interface and push the stream backwards by one document.

The user can view (or edit) a document by clicking on its representation in the stream. Rather than committing to a document model (e.g., ATK, PDF, ASCII, HTML, Microsoft Word, etc.) we have instead chosen an open architecture based on MIME types [Borenstein and Freed, 1992]. Similar to many Web browsers we rely on external helper applications to view and edit documents; this lessens the learning curve significantly for Lifestreams users, as they can continue to use the applications they are familiar with (such as emacs, xv, and ghostview) when working with documents, while at the same time using Lifestreams for its organization and

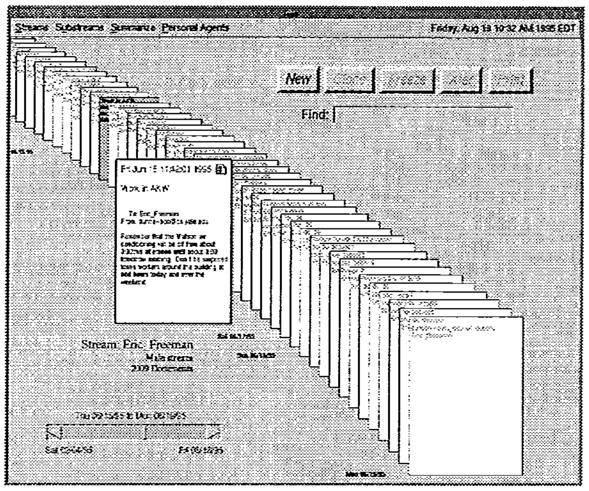

Streams Substreams Summarize Personal Agents
Friday, Aug 18 10:32 AM 1995 EDT
New
Find:
Stream: Eric_Freeman
Main stream
2009 Documents

Figure 1: The UNIX Viewport.

communication capabilities.

The interface prominently displays the primary system operations - **New**, **Clone**, **Freeze**, **Xfer** (i.e., transfer), **Find** and **Summary**, in addition to a few other pragmatic operations we commonly use (such as **Print** and **Freeze**) - as buttons and menus. The **New** button creates a new document and places it on the stream. The **Clone** button duplicates an existing document and places the copy on the stream. Documents are selected through a mouse button. The **Freeze** button is a convenient method for making a writable document read-only. The **Xfer** button first prompts the user for Lifestreams or Internet mail addresses and then forwards a copy of the document. **Print** copies a selected document to a printer[1]. **Find** is supported through a text entry box that allows the user to enter a boolean search query; it results in a new substream being created and displayed[2].

Menus are used to select from streams or existing substreams, create summaries, initiate personal agents, and to change the "time" of the viewport. The **Streams** menu allows the user to select from a list of locally available streams. Figure 2 shows the **Substreams** menu; the menu is divided into three sections. The first section contains a list of operations that can be performed on substreams (such as remove). The next section contains one menu entry labeled "Your Lifestream," and focuses the display on your entire Lifestream (i.e., all of your documents). The last section lists all of your substreams. Note that substreams can be created in an incremental fashion that result in a nested set of menus. In this example the nested menus displayed were created by

[1] This can easily be implemented by transferring all documents to a printer stream, where a stream agent forwards each new document to the appropriate printer. Our implementation, however, uses conventional methods of transferring documents to the printer.

[2] A new version, in the works, uses a more sophisticated dialog box to allow for searches on particular document fields.

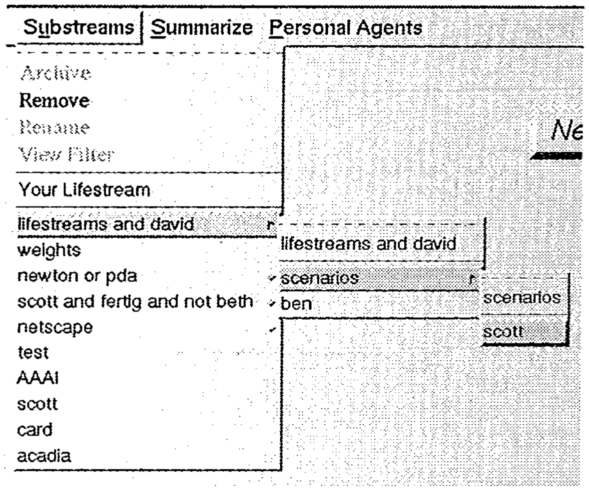


Figure 2: Selecting a Substream.

first creating a substream "lifestreams and david" from the main stream and then creating two substreams from this substream called "scenarios" and "ben." Likewise the substream "scott" was created from the "scenarios" substream. Semantically this incremental substreaming amounts to a boolean and of each new query with the previous substream's query (and so on recursively).

In figure 3, we find a list of possible summary types for this substream. Choosing any of these menu options creates a summary of the substream and a new document containing the summary is placed on the stream. Similarly the `Personal Agents` menu lists a number of agent types that can be executed. We will give examples of both summaries and agents in section 4.

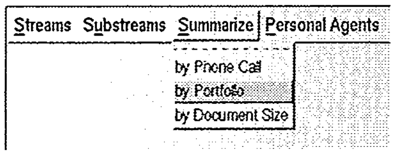


Figure 3: The summary menu item.

Finally, Lifestreams always displays the time in the upper right hand corner of the interface. This time display also acts as a menu, which allows the user to set the viewport time to the future or past via a calendar-based dialog box. To understand the effect of setting the time, imagine a pointer that always points to the position in the stream such that all documents towards the head of the stream have a future timestamp and all documents towards the tail have a timestamp from the past. As time progresses this pointer is moved towards the head.

The effect of setting the time to the future or past is to temporarily reset the pointer to a fixed position designated by the user. Normally the user interface displays all documents from the past up to the pointer. Setting the pointer to the future allows the user to see documents in the future part of the stream. Likewise, creating a document in this mode (i.e., with the pointer in the future) will result in a document with a future timestamp. Once the user is finished in the future or the past, she can reset to the present by selecting the "Set time to present" menu option in the time menu.

3.2 Supporting Agents

As we have discussed, Lifestreams supports three types of agents: personal agents, document agents and stream agents. All three agent types are specified in Tcl/Tk and have access to Lifestreams through our extended command set. Personal agents "live" in the user interface; users can define their own arbitrary personal agents and attach them to the user interface. Document agents are loosely based on enabled-mail [Borenstein and Rose, 1993] and are stored in a document attribute until they are spawned. Document agents can be spawned on three different events: (1) a document's arrival at a stream, (2) when it is read for the first time, and (3) any time it is read. Stream agents are attached to the stream data structure in the server and are spawned any time a document is added to the stream. We have chosen these initial events for document and stream agents; of course others are possible. We will return to agents in the next section and show an example of each type.

3.3 Performance

Our current prototype runs on a Sparcstation 10 and can handle three to four simultaneous users with a few thousand documents per stream. As of this writing, we are making architecture changes to the server and are also in the process of porting it to run on a faster IBM PowerPC workstation. When done, we expect the server to be able to support five to eight users with five to ten thousand documents per stream. The next logical step will involve load balancing users across a network of workstations, each workstation supporting the request of one or more users at a given time.

4 Common Tasks: Using Lifestreams

We have claimed that Lifestreams helps to simplify the most common computer tasks, such as communication, creating reminders, managing scheduling, tracking contacts, and managing personal finances (to name a few), and subsumes many separate, incompatible applications in the process. While a detailed description of how Lifestreams accomplishes this feat could fill a paper in itself, we will attempt to convey a sense of how the system is used through a handful of examples. Additional motivation for our claim can be found in the concluding remarks.

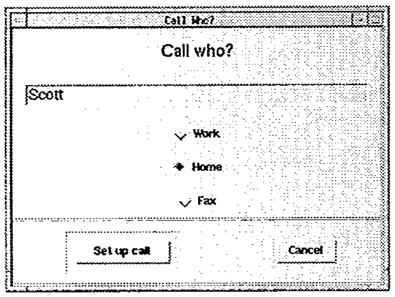
4.1 Sending and receiving Email; Automatic reminders

Using Email in Lifestreams is simple and not much different from what users are already accustomed to. To send a message, one creates a new document (by clicking on the **New** button) and then composing the message using a favorite editor. The message can then be sent with a push of the **Xfer** button. Similarly, existing documents are easily forwarded to other users, or documents can be cloned and replied to. While all mail messages (incoming and outgoing) are intermixed with other documents in the stream, the user can easily create a mailbox by substreaming on documents created by other users; or, users can take this one step further and create substreams that contain a subset of the mailbox substream, such as "all mail from Bob", or "all mail I haven't responded to."

We have already mentioned how users can dial to the future, depositing documents that act as reminders. A user can also send mail that will arrive in the future. If she "dials" to the future before writing a message, then when the message is transferred it won't appear on recipients' streams until either that time arrives or they happen to dial their viewports to the set creation date. In the present, the document will be in the stream data structure but the viewport won't show it. We use this ability to send mail to the future to post reminders to others about important meetings, department talks, etc. By appearing "just-in-time" and not requiring the user to switch to yet another application, these reminders are more effective than those included in a separate calendar or scheduling utility program.

4.2 Tracking contacts, Making a phone call

There are a number of contact managers on the market that store electronic business cards, the date and time of contacts, and time spent on tasks for billing purposes. Our research prototype currently supports an electronic business card document type as well as a "phone call record" document for noting the date and time of phone contacts. In addition we have automated much of the task of creating a phone call record through a personal agent. The personal agent is automatically attached to the personal agent menu, so anytime we want to make a call we choose "Make Phonecall" from the personal agent menu. The agent is spawned and the dialog box is figure 4 appears.

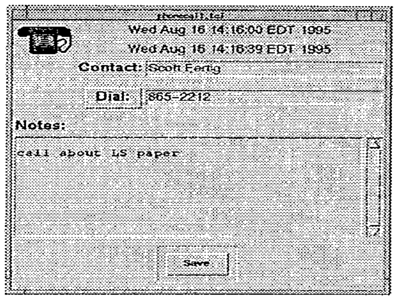


Scott

Work [✓ Home] Fax

Figure 4: The phone call agent.

The user types in the name of the callee, the agent then searches the current stream for a business card with



Wed Aug 16 14:16:00 EDT 1995
Wed Aug 16 14:16:39 EDT 1995
Contact: Scott Fertig
Dial: 865-2212
Notes:
call about LS paper

Figure 5: Phone record, automatically filled in by the agent.

that name and, if found, creates and fills in the appropriate entries of a phone call record as seen in figure 5 (this functionality is similar to the use of the personal assistant on the Newton platform).

The user can then later use the Lifestreams `summary` operation to summarize over the phone calls. This result is a report as shown below:

```
WHO           ON                          AT            ABOUT
--------------------------------------------------------------------
Scott Fertig  Tue Aug  1 12:05  EDT 1995  432-6433      Port to PPC
Ward Mullins  Tue Aug  1 11:57  EDT 1995  415 224-1912  Tcl/Jave discussion
Beth Freeman  Tue Aug  1 10:22  EDT 1995  432-1287      insurance
```

This could be extended to subsume the functionality of a time manager (and we are in the process of doing this). Time managers generally track the billable hours a professional spends on one or more projects. In Lifestreams this is easily accomplished by creating a timecard that marks the starting and ending time of each task (these timecards are just thrown onto the stream as they are used). Then, before each billing period, the stream is summarized by the timecards, resulting in a detailed billing statement for each contract.

4.3 Personal Finances

While online commerce will become commonplace in the next decade, millions already track their checking accounts, savings, investments, and budgets with their computers, with applications such as Quicken. The types of records and documents used in applications such as Quicken — electronic checks, deposits, securities transactions, reports — can be conveniently stored and generated by Lifestreams. We have just begun to explore using Lifestreams to manage personal finances, having implemented a fictional service that forwards the daily closing prices of a fictional portfolio to our Lifestreams at the end of every business day. These documents are simple ASCII documents as shown below.

```
Quote-O-Matic Stock Service for 5/16/95

GVIL    14.00
LMASX   20.84
ODWA    18.50
SPLS    27.12
TSA     19.25
lmvtx   21.41
```

The document lists each stock and mutual fund along with its closing price. The user can then perform a summary on his "portfolio" substream and is presented with the chart of historical data (having summarized over the history of the portfolio documents in the substream) in figure 6.

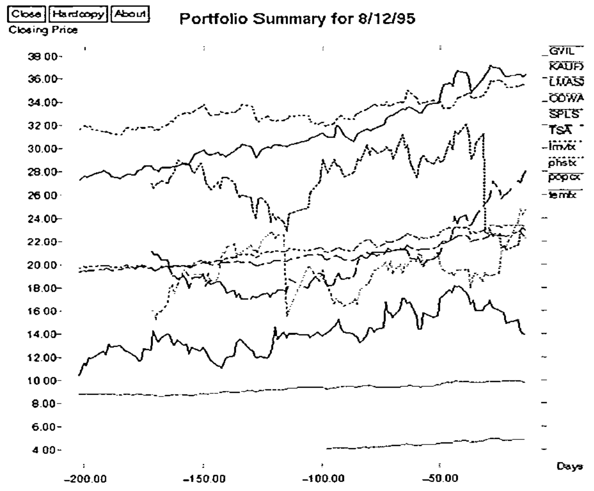


Figure 6: The portfolio summary.

This is just the beginning. A user could easily migrate his checking account to Lifestreams so that each check written creates a record on his stream. Some of these checks would be electronic checks sent to companies with an online presence; others transcribed from written checks (just as many people already do with Quicken). The user could then employs a personal agent to help balance his checkbook. At year's end he runs a tax summary which squishes the financial information in his stream down to a form 1040, which could then be shipped electronically to the IRS.

Lifestreams could help with budgeting, tracking expenditures, etc. Of course, many of these capabilities are already available in products like Quicken; it is worth pointing out, however, that Lifestreams contains *everything* a person deals with in their electronic life.

4.4 Interacting with the world: Web bookmarks

All of our previous examples made use of information stored in Lifestreams. We have also found Lifestreams quite useful for managing information outside of the system. For example, our research group found it difficult to keep track of our own Web bookmarks and inconvenient to pass interesting bookmarks to one another. This was usually accomplished by copying an URL from a Web browser to an email message, which the recipient would copy from email back to their own browser and add as a bookmark. We were able to solve both of these problems with Lifestreams.

We developed a system similar to "warm lists" [Klark and Manber, 1995], whereby a daemon watches each user's bookmark file, and each time a new bookmark is added the same bookmark is added to Lifestreams as a new "URL document." The effect of opening a URL document in Lifestreams is that our web browser comes to the foreground and attempts to connect to the URL. In this way we can use Lifestreams to create a bookmark substream while at the same time making the data in the bookmarks readily available to any other searches we might make on our stream.

Passing URLs around is trivial. We merely copy the URL document to another user's stream (a one step process) and the URL is automatically included in their bookmarks substream.

5 Current Research

Lifestreams is up and running while still being actively developed. User feedback is driving the development of new summarize capabilities and personal agents. We are now tackling the problem of enabling agents, and Lifestreams in general, to identify relevant documents automatically. We can loosely separate this into two research subgoals: Learning user profiles based on past interactions and context; and identifying documents that "match" a profile and therefore are likely to meet an expected or current need. As of yet we have nothing to say about the first problem. We discuss current work on Lifestreams in relation to the second after making clear our assumptions.

5.1 A few words about indexing

The Lifestream model assumes that documents are never deleted. They can be moved to archival storage or hidden from view, but at some level remain accessible. Therefore a Lifestream implementation must provide efficient and context-sensitive searching capabilities so that users can find what they are looking for. Too often, finding the documents we need in our semi-structured file systems is uncomfortably similar to searching a packed hall closet, or dimly-lit garage, for an urgently needed object. We know that what we are looking for is there but don't know how to find it. We asked ourselves, "Is it possible to index a document so that it is easily found two months, or two years, into the future, when it's needed in a possibly quite different context?"

We think the answer is yes, but a necessary (although not sufficient) condition is that documents be as richly indexed as possible. For example, text documents in Lifestreams are indexed by every word (modulo a standard stop-list.) We assume it is better, not just faster, for the indexing algorithm to leave indices unweighted at the time of creation. No particular index is flagged as more important than any other since retrieval context is difficult or impossible to anticipate. We believe that creating complex structured indices is unnecessary as long as the server supports retrieval by arbitrary combinations of simple indices efficiently. It is the responsibility of the retrieval software to find the right combination, possibly through an iterative process of successive approximation.

5.2 Refining the search for relevant documents

We designed Lifestreams with a particular philosophical bent: That while documents should be indexed by unweighted keys, retrieval happens in a context, and this context can be exploited to weigh terms (indices) dynamically. This extends the model first proposed by Fertig and Gelernter [Fertig and Gelernter, 1991] to support the automatic calculation of retrieval cue *evocativeness*. The concept of feature evocativeness has a long history in AI, going back at least to the expert system Internist-1 [Miller *et al.*, 1982]. But whereas the authors of Internist-1 manually assigned weights to features, the Lifestreams server will empirically and temporarily assign weights. It will use statistics calculated from all the documents stored by the server as well as a particular query context to perform feature weighting, and then use these weights to refine the search. Here is how it might work. Suppose you start with a boolean search predicate:

find `adaptive parall*` and `MPP` or `piranha` and `MPP`

It yields a substream of documents. The `summarize` operation can come up with a set of words that are "characteristic" – that occur in "most" of the documents in this substream. These words (w1, w2...) can then be treated as potential new search cues. Each of the commonly occurring terms can then be weighted by its evocativeness, that is, by how many documents it indexes and whether there is substantial overlap with the documents already retrieved. Terms are defined as evocative if the degree of overlap is large yet the total number in the retrieved set small. The most evocative terms can then be added to the search predicate in the form "w1 or w2 or ...". This process would correspond to executing:

find `adaptive parall*` and `MPP` or `piranha` and `MPP` and `etc`

The "etc" would mean "and other documents along these lines." This can be viewed as a form of matching documents to a user's "profile", where a profile is rather simply defined as one specific query context. Once this capability is in hand we can investigate methods of learning such profiles from users' interactions with the system.

6 Conclusion

Computer users must deal with a far greater quantity and range of electronic objects in their work and personal lives than ever before. New systems must support the management of these objects efficiently and transparently. The success of Lotus Notes, and the transition in operating systems toward a document-centric model (as seen in OLE and OpenDoc) exemplifies this; documents are now the sun around which applications revolve. Lifestreams takes this one step further: It embodies in software the stream of electronic events already facing the user, and brings in applications (filters, summaries, agents, etc.) as needed. Lifestreams anticipates the time when bit storage is provided by service utilities, and users will be able to view their data from anywhere. We believe Lifestreams replaces the rigid storage system — a left-over artifact from early computer operating systems — with a more fluid, and natural system that reflects the way users work.

References

[Borenstein and Freed, 1992] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of internet message bodies, June 1992.

[Borenstein and Rose, 1993] Nathaniel Borenstein and Marshall T. Rose. MIME extensions for mail-enabled applications: application/Safe-Tcl and multipart/enabled-mail, Nov 1993.

[Cook, 1995] Terry Cook. Do you know where your data are? In *Technology Review*. MIT, January 1995.

[Fertig and Gelernter, 1991] Scott J. Fertig and David H. Gelernter. FGP: A software architecture for acquiring knowledge from cases. In *Proc. of the International Joint Conference on Artificial Intelligence*, August 1991.

[Freeman, 1995] Eric Freeman. Lifestreams for the Newton. *PDA Developer*, 3(4):42–45, July/August 1995.

[Gifford *et al.*, 1991] David K. Gifford, Pierre Jouvelot, Mark Sheldon, and James O'Toole. Semantic file systems. In *13th ACM Symposium on Operating Systems Principles*, October 1991.

[Klark and Manber, 1995] Paul Klark and Udi Manber. Developing a personal internet assistant. In *ED-MEDIA '95 World conference on educational multimedia and hypermedia*, June 1995.

[Manber and Wu, 1993] Udi Manber and Sun Wu. Glimpse: A tool to search through entire file systems. Technical Report 093-34, Department of Computer Science, The Univesity of Arizona, October 1993.

[Miller *et al.*, 1982] R.A. Miller, H. Pople, and J.D. Meyers. Internist-i, an experimental computer-based diagnostic consultant for general internal medicine. *New England Journal of Medicine*, 307(8):468–476, August 1982.