Lifestreams: A Storage Model for Personal Data

Eric Freeman and David Gelernter
Department of Computer Science
Yale University
New Haven, CT 06520

Abstract—
Conventional software systems, such as those based on the "desktop metaphor," are ill-equipped to manage the electronic information and events of the typical computer user. We introduce a new metaphor, Lifestreams, *for dynamically organizing a user's personal workspace. Lifestreams uses a simple organizational metaphor, a time-ordered stream of documents, as an underlying storage system. Stream filters are used to organize, monitor and summarize information for the user. Combined, they provide a system that subsumes many separate desktop applications. This paper describes the Lifestreams model and our prototype system.*

I. Introduction

INEXPERIENCED users are right to be confused by today's operating systems; they aren't well suited to most users needs and they require too many separate applications, too many file and format translations, the invention of too many pointless names and the construction of organizational hierarchies that too quickly become obsolete. They are built on ideas like named files (an invention of the 50s) in hierarchical directories (60s) that were brilliant when they were new but have long since become obsolete. Consider, for example, the "desktop metaphor," which attempts to simplify common file operations by presenting them in the familiar language of the paper-based world (paper documents as files, folders as directories, the trashcan for deletion). Although this metaphor has been successful to a point (one usually has to explain to a new user how the computer desktop is like a real desktop), its use has some unfortunate consequences: the paper-based model is a rather poor basis for organizing information[1] and constrains our choices in creating new information systems [15].

We've developed "Lifestreams" in an attempt to do better. Lifestreams, first proposed in [9] and described in [8], is a new model and system for managing personal electronic information. Lifestreams uses a *time-ordered stream* as a storage model and *stream filters* to organize, locate, summarize and monitor incoming information. Together, streams and filters provide a unified framework that subsumes many separate desktop applications to accomplish and handle personal communication, scheduling, and search and retrieval tasks. The prototype that exists today realizes many of the system's defining features and has allowed us to experiment with the model's key ideas. We've developed our system as a machine-independent, client-server architecture that is *open* so users can continue to use the document types, viewers and editors they are accustomed to. In this article we describe the model, our current implementation, its use in context, and the direction our system is headed.

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

[1] Where the state of the art is still a messy desktop.

II. The Idea

Lifestreams is based on the following observations:

1. *Storage should be transparent.* "Naming" a file as it is created and choosing a location for it is unneeded overhead. Names should be required only when users feel like inventing them and storage should be handled automatically by the system. When you grab a piece of paper and start writing, no-one demands that you bestow a name on the sheet or find it a storage location. Online, many filenames are not only pointless but useless for retrieval purposes. Storage locations are effective only as long as the user remembers them.

2. *Directories are inadequate as an organizing device.* Software is too faithful to the paper-based world; paper can't be in more than one place, but electronic documents can (or can appear to be). Conventional systems force users to store new information in fixed categories (namely, directories). But information should be organized as needed, not once and for all when it is created. Directories should be created *on demand* and documents should belong to as many of them as seems reasonable, or to none.

3. *Archiving should be automatic.* Data archiving is an area where electronic systems fail miserably compared to paper-based systems. Paper-based systems are first and foremost archiving systems, yet data archiving is difficult in conventional desktop systems. Often, users throw out old data rather than undertaking the task of archiving it (and remembering how to get it back). If software systems make archiving and retrieval more convenient, old information could be reused more often.

4. *The system should provide sophisticated logic for summarizing or compressing (and where appropriate, for picturing or animating) a large group of related documents of which the user wants a concise overview.* No matter how many documents fall into a given category, the system should be capable of summarizing the whole lot on a single screen. For some types of

documents, pictures or animations will be good vehicles for summaries.

5. *Computers should make "reminding" convenient.* Reminding is a critical function of computer-based systems [13][12], yet current systems supply little or no support for it. Users are forced either to use location on their graphical desktops as reminding cues or to use add-on applications such as calendar managers. We have argued that the former is a mere coping strategy (for lack of a better method) [6], while the latter could clearly be improved if operating systems helped.

6. *Personal data should be accessible anywhere and compatibility should be automatic.* Computers will eventually be used not as independent data storage devices, but as "viewports" to data stored and maintained on the Net. Users should be able to access their personal information worlds from any available platform—from a Unix machine at work, a Mac or PC at home, a PDA on the road, even a set-top box via cable. Data should be accessible everywhere, regardless of the viewing device.

Some of these observations point to areas where software systems don't match the flexibility of paper-based systems. Others suggest areas where our software systems can do better. We will return to these observations (goals) throughout this paper, describing how Lifestreams realizes each of them. Our current work realizes a fair bit of goals 1, 2, 4, and 5, some of 3 and 6. But all these goals are central to the project.

III. The Model

A *lifestream* is a time-ordered stream of documents that functions as a diary of your electronic life; every document you create and every document other people send you is stored in your lifestream. The tail of your stream contains documents from the past (starting with your electronic birth certificate). Moving away from the tail and toward the present, your stream contains more recent documents — papers in progress or new electronic mail; other documents (pictures, correspondence, bills, movies, voice mail, software) are stored in between. Moving beyond the present and into the future, the stream contains documents you *will* need: reminders, calendar items, to-do lists.

In this section we describe Lifestreams in terms of its basic operations: `new`, `clone`, `transfer`, `find` and `summarize`. In the process we show how Lifestreams provides transparent storage, organization through directories on demand, and the ability to create overviews. We then examine the underlying time-based storage model and, in the process, show how Lifestreams accomplishes archiving and reminding in a natural way.

Document Creation and Storage

Users create documents by means of `new` and `clone`. `New` creates a new, empty document and adds it to your stream. `Clone` takes an existing document, creates a duplicate and adds it to your stream. Documents can also be created indirectly through `transfers`, which copy a document between streams. Creation is always "transparent" because documents, by default, are always added to the end of the stream and don't have names unless users want them to.

Directories on Demand

Lifestreams are organized on the fly with the `find` operation. `Find` prompts for a search query, such as "all email I haven't responded to," or "all faxes I've sent to Schwartz," and creates a *substream.*

Substreams, like virtual directories [10][14], present the user with a "view" of a document collection. The view contains all documents that are relevant to the search query. Substreams differ from conventional directory systems in that, rather than placing documents into fixed, rigid directory structures, they create virtual groups of documents from the stream. Documents aren't actually stored in them; a substream is a temporary collection of documents that already exist on the main stream. Substreams may overlap and can be created and destroyed on the fly without affecting the main stream or other substreams.

Substreams are dynamic. If you allow one to persist, it will collect new documents that match your search criteria as they arrive from the outside or as you create them. The result is a natural way of monitoring information— the substream acts not only as an organizational device, but as a filter for incoming information. 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.

Overviews

The last operation, `summarize`, takes a substream and compresses it into an overview document. The content of the overview document depends on the type of 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, the overview document may contain a chart displaying the historical performance of your securities and your net worth. If the substream contains a list of tasks you need to complete, the overview document might display a prioritized "to-do" list.

Chronology as a storage model

Given that we use substreams to organize documents, why bother with the underlying time-based ordering? For several reasons: time is a natural guide to experience; it is the attribute that comes closest to a universal skeleton-key for stored experience (Malone, for example, suggests the utility of time-based organization in his early studies [13].) The stream adds historical context to a document collection; all documents eventually become read-only (in the past, set in stone for history), and the stream preserves the order and method of their creation. Like a diary, a stream documents work, correspondence and transactions.

Although historical context can be crucial in an organizational setting [4], most current systems do little to track when, where, and why documents are created and deleted.

The three portions of the stream, *past*, *present*, and *future*, mirror information categorization in user studies [12], [2]. The "present" portion of the stream holds "working documents;" this is also typically where new documents are created and where incoming documents are placed. As documents age and newer documents are added, older documents recede from the user's view and are "archived" in the process (here we mean archiving in the conceptual sense; users don't have to worry about old information cluttering their desktops or getting in the way; if at some future point they need the archived information, it can be located with `find`).

The "future" portion of the stream allows documents to be created in the future. "Future creation" is a natural method of posting reminders and scheduling information. The system allows users to dial to the future and deposit a document there—say a meeting reminder. When the date arrives the reminder appears in the present.

IV. The Lifestreams Interface

Our research prototype consists of a client/server architecture that runs over the Internet. The server is the workhorse of the Lifestreams system and handles one or more streams—storing all stream documents and substreams. Each viewport is a client of the server and provides the user with an interface to the document collection. We believe the "look and feel" of the viewport interface will differ radically over the range of computing platforms, from set-top boxes to high-end workstations, but each viewport will support the basic operations.

We have currently implemented three client viewports: one using X Windows, one ASCII-based and one for the Newton PDA. The X Windows viewport provides a graphical interface and implements the full range of Lifestream functionalities; the ASCII interface also implements the full-range Lifestreams but with a mail-like interface; the Newton version implements a minimal stream-access method, given its lack of internal memory and low-bandwidth communications. In this paper we concentrate on the X windows viewport (information on the Newton version can be found in [7].)

Our X Windows viewport is shown in figure 1. The interface is based on a visual representation of the stream metaphor. Users can slide the mouse pointer over the document representations to "glance" at each document, or use the scroll bar in the lower left-hand corner to roll themselves back into the past.

Color and animation indicate important document features. A red border means "unseen" and a bold one means "writable"; open documents are offset to the side to indicate they are being edited. Incoming documents slide in from the left side, and newly created documents pop down from the top and push the stream backwards by one document into the past.

The user can view (or edit) a document by clicking on its graphical representation. We rely on external helper applications to view and edit documents, which speeds the learning process significantly for Lifestreams users — they can continue to use applications they are familiar with (such as emacs, xv, and ghostview) to create and view documents, while using Lifestreams to organize them and communicate.

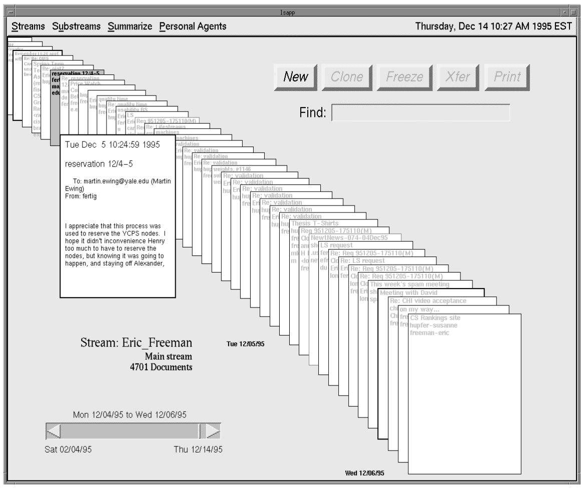

Fig. 1. The UNIX Viewport.

The interface prominently displays the primary system operations — `New`, `Clone`, `Xfer` (i.e., transfer), `Find`, `Summarize` and a few useful secondary operations (such as `Print` and `Freeze`) — as buttons and menus. The `New` button creates a new document and adds it to the stream. The `Clone` button duplicates an existing document and places the copy on the stream. The `Freeze` button makes a writable document read-only. `Xfer` first prompts the user for one or more mail addresses and then forwards a document. `Print` copies a selected document to a printer.[2] `Find` is supported through a text entry box that allows the user to enter a boolean search query, which results in a new substream being created and displayed.

Menus are used to select from streams or existing substreams, create summaries, initiate personal agents and change the clock. 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 contains a list of operations that can be performed on substreams (such as remove). The next 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 results in a nested set of menus. In this example the nested menus were created by first creating a substream "lifestreams and david" from the main stream and then creating two substreams from this substream, "scenarios" and "ben." Substream "scott" was created from

[2] This could 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.

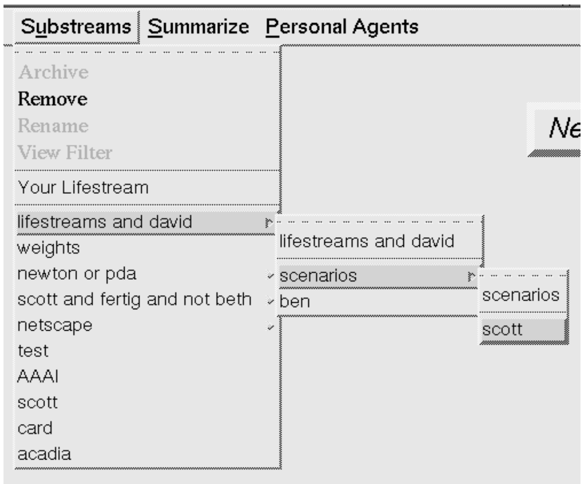

Fig. 2. Selecting a Substream.

the "scenarios" substream. Semantically this incremental substreaming amounts to a boolean **and** of each new query with the previous substream's query.

Figure 3 shows a list of possible summary types for this substream. Choosing any of these menu options creates a substream summary, and a new document containing the summary is placed on the stream. The `Personal Agents` menu lists a number of available agent types. (We discuss personal agents in passing in the next section. They can be added to the user interface in order to automate common tasks: see [8] for more information.)

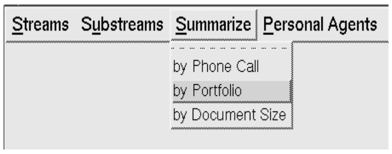

Fig. 3. The summarize menu item.

Finally, Lifestreams always displays the time in the upper right hand corner of the interface. This time display also acts as a menu (figure 4) that allows the user to set the viewport time to the future or past via a calendar-based dialog box (figure 5). Imagine a cursor always pointing to the position in the stream such that all documents beyond that point towards the head have a future timestamp and all documents before it, towards the tail, have a past timestamp. As time progresses this cursor moves forward towards the head; as it slips past "future" documents they are added to the visible part of the stream, just as if new mail had arrived.

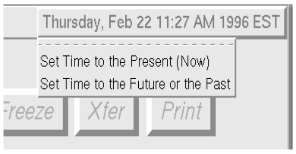

Fig. 4. Altering time in Lifestreams.

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

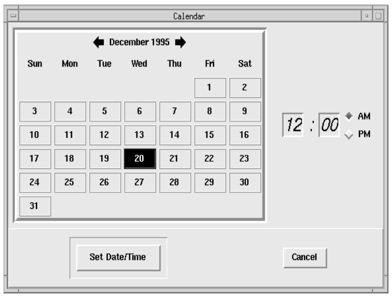

Fig. 5. Setting a time.

V. Lifestreams in Practice

We now look at examples of how Lifestreams can be used to accommodate common computer tasks, such as communication, creating reminders, managing scheduling, tracking contacts, and managing personal finances (to name a few). While a detailed description of how Lifestreams is used could fill a paper in itself, we will attempt to convey a sense of how the system is used through a handful of examples.

Sending and receiving Email; Automatic reminders

Using email in Lifestreams is not much different from what users are already accustomed to. To send a message, the user creates a new document (by clicking on the **New** button) and then composes 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 he "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.

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 in figure 6 appears.

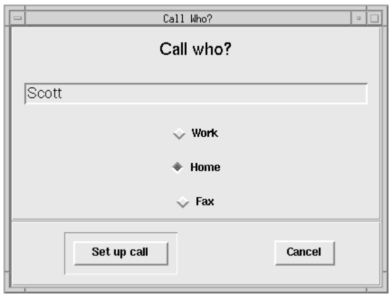

Fig. 6. The phone call agent.

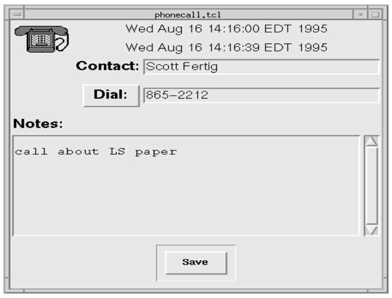

Fig. 7. Phone record, automatically filled in by the agent.

The user types in the name of the callee; the agent then searches the current stream for a business card with that name and, if found, creates and fills in the appropriate entries of a phone call record as seen in figure 7 (this functionality is similar to the use of the personal assistant on the Newton platform).

The user can then later use the Lifestreams **summarize** operation to summarize over the phone calls. This results in 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/Java 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.

Personal Finances

Online commerce is quickly becoming commonplace; a large number of users already track their checking accounts, savings, investments, and budgets with applications such as Quicken. The types of records and documents used in these applications — 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 stock quote service that forwards the daily closing prices of a fictional portfolio to our lifestreams at the end of every business day.[3] 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, giving the user a method of calculating his assets on a specific day. But what if the user wants "higher-level" view of his portfolio over time? This is where **summarize** can be used. The user first selects a substream containing his stock quote documents, and then he selects the "summarize by portfolio" menu item. This compresses the data into a single chart of historical data having summarized over the portfolio documents in the substream. We present the result in figure 8.

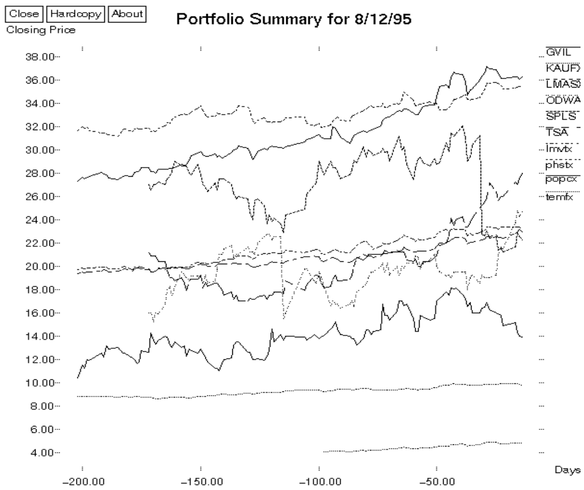

Fig. 8. The portfolio summary.

[3] Many similar stock quote services already exist of the Internet.

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 employ 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 his electronic life (and in a convenient and searchable location).

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 a 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" [11], 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 his bookmarks substream.

VI. Status and Future Directions

Lifestreams is up and running on our local computing environment (a collection of SunOS, Solaris, and AIX workstations) and supporting a few local users. We plan to expand this number as the software matures. Our initial implementation efforts have focused on providing a "proof of concept" for the Lifestreams model. Based on feedback from the initial users we judge the experiment to have been a success — users find the system valuable and don't want to give it up. Given the nature of the system our work has proceeded on many fronts, including user interface design, system integration, indexing and retrieval, agent technologies, network access, security, and performance issues. The goals of our first system have been modest; each server to support three to four simultaneous users with stream sizes on the order of 5,000 documents (perhaps a year or two of documents for the average user). As of this writing, we are making architecture changes that should allow lifestreams of 20,000 documents; that may well be the limit for our current architecture.

We are now considering design changes that will make the system more scalable; previous work in the information retrieval and database communities is encouraging. Lifestreams incorporates ideas from both disciplines; but focuses on personal storage rather than centralized data collections.

Lifestreams already incorporates many current information retrieval techniques. Substreaming is efficiently implemented using an inverse index of the document collection (maintained by the server). We've seen no real performance problems with respect to retrieval and, given the very large indices that are being used on the Internet, we believe our retrieval scheme should scale to large document collections.

Lifestreams also has much in common with database systems. Substreams are related to a "views" in relational databases [5]. "Triggers" are related to future documents in Lifestreams (in Lifestreams, the trigger occurs when the creation date of the document slides into the past). There are also connections between Lifestreams and temporal databases [18], temporal logic [1], and sequence database systems [16] where time and/or logical sequences play a crucial role in the system.

Our most immediate gain potential from database work is in efficient handling of large document collections. Currently, both the client and server keep in core the records of the entire document collection when a user views his whole Lifestream. We need to borrow database technology for large collections. There are HCI problems to solve here too. Since no user can look at 10,000 documents at once and discern any usable information,[4] it doesn't make sense to give users an entire document collection at once. A more reasonable approach would be to use "cursors" to allow the user to view segments of the document collection, and to load in more segments as needed.

Our last problem area involves multi-user access to a Lifestream. Our current implementation only provides a single-threaded server (and thus a single point of access). While we've found server performance reasonable for a small number of users, clearly a multi-server and multi-threaded approach present a more scalable architecture. Our previous work in parallel computing has explored these areas [3], but there is work still to be done in the integration of these techniques with basic information retrieval and database functionality.

VII. Acknowledgments

The authors wish to thank Scott Fertig, Michael Franklin, Elisabeth Freeman, and Susanne Hupfer for their comments and suggestions on drafts of this paper.

[4] Although recent work by Shneiderman [17] suggests how one might, in principle, do so. His techniques still need work, through, to scale beyond small databases of information.

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] Deborah Barreau and Bonnie A. Nardi. Finding and reminding: File organization from the desktop. In *SIGCHI Bulletin*. SIGCHI, July 1995.

[3] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, April 1989.

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

[5] C. J. Date. *Database Systems*. Addison-Wesley, 1986.

[6] Scott Fertig, Eric Freeman, and David Gelernter. Finding and reminding reconsidered. In *SIGCHI Bulletin*, volume 28, January 1996.

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

[8] Eric T. Freeman and Scott J. Fertig. Lifestreams: Organizing your electronic life. In *AAAI Fall Symposium: AI Applications in Knowledge Navigation and Retrieval*, November 1995. Cambridge, MA.

[9] David Gelernter. The cyber-road not taken. *The Washington Post*, April 1994.

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

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

[12] M. Lansdale. The psychology of personal information management. *Applied Ergonomics*, March 1988.

[13] Thomas W. Malone. How do people organize their desks? Implications for the design of office information systems. *ACM Transactions on Office Systems*, 1(1):99–112, January 1983.

[14] 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.

[15] Theodor Nelson. The right way to think about software design. In *The Art of Human-Computer Interface Design (Ed.) Brenda Laurel*, 1990.

[16] Praveen Seshadr, Miron Livny, and Raghu Ramakrishnan. Sequence query processing. In *ACM SIGMOD Conference on Data Management*, 1984.

[17] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, pages 70–77, November 1994.

[18] Richard T. Snodgrass. Temporal databases - status and research directions. *SIGMOD Record*, 19(4):83–89, 1990.