Lifestreams: Bigger than Elvis

Nicholas Carriero, Scott Fertig, Eric Freeman and David Gelernter

March 25, 1996

Introduction

Here's your compute-environment circa 2010: every document you've ever created or received stretches before you in a time-ordered stream, reaching from right now backwards to the date you were born. You can sit back and watch new documents arrive: they're plunked down at the head of the stream. You browse the stream by running your cursor down it—touch a document in the display and it pops out far enough for you to glance at its contents. You can go back in time, or go to the future and see what you're supposed to be doing next week or next decade. Your entire cyber-life is right there in front of you.

You might be in danger of being overwhelmed by all the documents on your screen—but you use "substreaming" to describe the documents you want, and everything else (temporarily) disappears. When you want to create a new document, you spend zero time deciding where to put it and what to name it. You press a button and a new element pops up at the head of the stream; you can put whatever you want inside. You can tune in your lifestream from any internet-connected computer: a Unix machine or PC at work, your Mac or set-top TV box at home, the generic computer at the supermarket, bank, hotel room or phone booth.

The cyberworld of 2010 is a collection of information sites. Your lifestream is your "personal site." A newspaper or TV station, a catalog company or tech-report cache are information sites too, and in the cyberutopia we have in mind, most such sites are stored in lifestreams. When a lifestream stores the electronic edition of a newspaper, today's

edition goes at the the streamhead with previous editions lined up behind. On a TV station lifestream, the head element holds the latest frame of sound-and-picture. You can grab and display each new image as it emerges in realtime or watch material broadcast earlier. In this worldview, the ignition key that fires up your Lifestreams dashboard (which will look like a credit card, presumably, and store your personal decryption and accounting information) is your key to the cyber-universe.

What makes this utopian picture utopian? Why is it better than today's cyberworld? Our goal in managing information is to make the time-you-spend-managing tend to zero as your "information power"—your capacity to find what you need and do what you want—tends to infinity. The Lifestreams system is promising because it makes electronic *housekeeping* fast and easy (you spend minimal time organizing and arranging things) and *retrieval power* is high (it's easy to find what you want and manipulate it effectively). Most important, the system promises large benefits from *integration*: it allows you to accomplish many tasks using the same small repertoire of techniques. A good tool—mathematical, philosophical, gardening—has the effect of bringing to the fore deep-lying similarities among superficially-different problems (e.g. planting tulips, preparing a perennial bed, digging up saplings, bopping raccoons). That is the Lifestream system's main goal.

Another Lifestreams property is more elusive but still important: the essence of a Lifestreams environment (as opposed to a Unix or conventional Macintosh environment, for example) can be captured in a simple mental picture. When you connect to your lifestream, the image you see on a Lifestreams-equipped Macintosh might be different from what you'd encounter on a PC or TV or hand-held digital assistant or ASCII terminal or 3-D virtual-reality goggles. (So far we have Lifestreams "viewports" for Unix machines running X windows, ASCII terminals and Apple's "Newton" PDA.) But it's the user's *mental* picture that counts, not the image on the screen. Because that mental picture and the operations that go with it are simple, readily grasped and readily held in mind, it will be easy (we believe) for users to adapt to a variety of Lifestreams dashboards, using the underlying mental picture as a guide. That adaptability in turn is a strong basis

for "universal access"—the idea that your Lifestream should be accessible from any net-connected machine anywhere regardless of hardware or operating system.

Lifestreams isn't a self-contained, hermetically-sealed system. It's designed to work closely with existing word processors and a wide range of popular applications, from finance managers to Web browsers. Today's browsers, finance managers et. al. are first-rate actors performing on cramped, junk-strewn stages. Lifestreams' goal is to provide them with a cleaned-up rational platform on which to cavort.

The Lifestreams system that is up and running today doesn't fully implement the model—it's still an active research project—but it goes a fair way towards the goal and already functions well as a personal information site. The fully-realized model envisions "Lifestream provider" companies working in tandem with software on the user's node, server-client style. The provider stores your lifestream securely and robustly and provides you with efficient universal access. ("Securely" will presumably require encryption. The private key with which you decode your stream might be stored on the few machines you rely on most, but to get access from a machine at the supermarket you will need to insert the key-card you carry in your wallet. The resulting system isn't airtight, but can probably be made at least as secure as the paper version of your diary.)

Your own computer acts as a cache with respect to the Lifestream server. The server takes over management of your local disk and uses it to store up-to-date versions of as much of your current stream as will fit. (Up-to-date master copies of everything remain on the server.) If you're hard at work on your stream when a UPS truck arrives with your new computer, you can unplug your old machine and compost it, hook up your new one and (assuming the new one supports Lifestreams) continue working in your stream with no discernible blip, except for temporary access delays while the server refills your suddenly-empty cache.

Clearly the Lifestreams model, which puts the internet (or if you prefer the "information highway") at the center of the computing universe, requires important technological changes—most important, faster and higher-capacity network links. But those changes are coming re-

gardless. The interesting question is this: once we've got a high-speed information highway, what do we do with it? For one, we run Lifestreams.

Your lifestream as a personal cybersite

Every chunk of information (every document, email message, application transcript, rolodex card, appointment-calendar item...) is stored by this system in a single time-ordered stream. When you tune in, you see a stream of documents receding into the distance; farther away in imaginary space means farther back in time. When you create a new document or one arrives (via email, for example), you see a new document pop up at the streamhead. To create a new document, you can press the "new" button and get an empty box ready to fill, or clone an old document and get a new copy to alter as you choose. You don't need to name documents (although you can); documents are located by attribute and chronology. Attributes describe the type, point- and time-of-origin and other aspects of a document. We count every non-trivial word in a document as an attribute too, so we can do content-based searches. The *find* button creates a substream—you can ask for "all documents that mention Zeppelins or rigid airships," "all messages from Schwartz," "the last message from Schwartz" and so on. In response, the system shows you the portion of the stream consisting only of the documents you have specified, and deposits on the main stream a "calling card" document that gives you access to the new substream if you want to revisit it. Any substream can be "menu-fied"—displayed as a pull-down menu, for ready access to its contents; our current viewports menu-fy the substream of calling cards by default, so that users can browse readily among substreams.

A substream persists until you kill it. A newly-arriving Zeppelin-related document gets dumped in the main stream and also appears on every substream where it fits. If Schwartz sends another message, the single member of the "last message from Schwartz" substream changes.

In our current system you can *find* things based on keywords and attributes. Future versions will incorporate technology from our ongoing

expert database project; we intend for it to be possible to find documents on the basis of fuzzy, abstract or not-quite-accurate descriptions as well as exact ones.

When you press the "squish" button you get a summary of a substream. The type of summary depends on the type of information in the substream—textual for plain documents, graphs or pictures or animations for the appropriate more-specialized types. The "squish" button automatically invokes an appropriate *squish* for this substream (or offers you a choice of reasonable squishers). (Our prototype comes with a few different *squish*es built in, but the system is intended to accommodate custom add-on squishers as well. In some cases, highly complex and sophisticated squishers will be desirable. The Lifestream system's contribution isn't to say how these squishers should be built—rather to suggest *that* they be built, and to provide a uniform framework in which they can be installed. We anticipate a lively after-market in fancy squishers one day.) An "email message" is any document copied from one stream to another. You print a document by copying it to a printer stream. To run an application, you create a new document and run the program inside; output is stored and retrieved like any other document.

The stream has a future as well as a past. Appointments and calendar items are stored in the future, and become visible when their creation-times roll around or when you go to the future on purpose to look around. You can build forward-into-the-future substreams just as you build backward-into-the-past ones. A future-looking substream selects elements from "now" forward—it might show you all your appointments for next week, or every scheduled meeting in which you are likely to see Feinstein.

The stream, then, has three segments—past, present and future. Ordinarily the display shows only the present and (stretching out behind it) the past; the future is visible when you ask for it. (But we're considering a different default, in which the entire stream is visible in the form of two legs hinged together—the longer left leg showing the present and past exactly as the current display does, the right leg showing the future, with farther-away documents lying farther ahead in time.)

Documents in the "present" are writable. Farther back, in the "past," they have frozen into history and become read-only. Each user decides when the present ends and the past begins—at what point, in other words, documents freeze. One possibility (and the system default) is to freeze today's documents at the start of tomorrow. In that case you can work on a document all day, but to continue the next day you need to clone a new copy onto the streamhead. Or a user might postpone freezing for a week, or forever. (Users can override the controls as they choose. By default, Lifestreams saves every document and makes frozen documents unwritable. But you can delete a document or unfreeze it if you want.) The far-tail of the stream—for example, documents that are more than two years old—may disappear at the implementation's discretion into archival storage. The user specifies where the "far tail" begins, but the Lifestreams-provider will presumably set charges that depend on a user's willingness to have old material dumped into data warehouses. The implementation guarantees that the "header" of every document—the information that is available on the display when you browse but before you open a document—will always be available immediately. But if you journey to the remote past and open a document that has been archived, you may have to wait a while until it wends its way back to daylight.

Agents can troll down lifestreams, which have synchronization properties that are designed to make agent-building convenient. For example: an agent can cruise to the head of a stream and go to sleep after posting instructions that it be awakened whenever a new document arrives. Agents are important to many aspects of the system; we discuss some below. Custom squishers, custom agents and custom viewports are the main ways in which the system is designed to accommodate extensions and refinements.

The stream is organized by time because it is intended to function as an electronic diary. It's not just a file cabinet for information; it tracks your daily experience as it unfolds. Such a record is inherently useful—which is why people keep journals or diaries, or used to. For example Terry Cook, director of the Records Disposition Division at the National Archives of Canada, argues that the key to effective electronic record-keeping "lies in being able to determine,

sometimes long after the fact, not only the content but also the context of a record in question." (He has studied cases in which lack of context made electronic records useless.) Further, organizing documents chronologically makes it possible to use daily experience as a powerful retrieval guide. Mark Lansdale's experiments at the Loughborough University of Technology have shown that, as you would expect, "people remember chronological information about information." He cites examples: "My boss wants to see all the project reviews I have carried out over the last six months. The trouble is, they are filed under each of the individual projects. It will take me ages to work through and dig them all out." "Yes I remember that paper. It came at the same time as the product audit. I can't remember what happened to it, though."

> The underlying idea requires, furthermore, that every document at a personal site be stored in a single structure—so that you don't have to invest any time in correctly pigeonholing a document upon creation, and searches can take in your whole cyberworld at one shot. So we need a single master structure—and creation or arrival time is the one unambiguous, useful ordering attribute all documents have. (Thus we might have organized the master structure by "topic," but topic is hard to establish and a document might have several; we might have organized it by "length," but time is far more useful as a retrieval key; and so on.)

You don't search for documents only on the basis of creation time, of course. And for those who like conventional directories, the system can accommodate them by means of the substream mechanism.

Lifestreams in Action

§ Daily Business

Let's say you'd been working last night on the user manual for the new two-seater blimp coupé your company (Blimptronics Inc.) will be releasing soon. When you logged off, the blimp manual was at the

head of your stream. When you log on this morning you find five pieces of fresh email at the stream head. You glance at the new email by running the cursor down it—when you touch a stream document with your cursor, it slides out of line and you can see the beginning.

You decide to ignore the email and get on with the manual. You've set up your system to freeze documents overnight, so you reach back with your cursor to the manual and press "clone." A new copy appears at the streamhead. Names are optional; if you'd given the old version a name (say "blimp manual"), the clone inherits the same name. (In this case, typing "latest blimp manual" to *find* yields a one-element substream. The query "blimp manual" produces the multi-element substream holding every version since the start.)

Let's say you are about to get to work on the "valving helium" section and recall that your colleague Feinstein once sent you email on the topic. You press *find* and type "Feinstein and valving helium." The resulting substream has one element. You go to that document and, using your standard screen editor, copy a paragraph into your manual. Then you decide that it might be worth examining everything you've got on "valving helium"—mail, notes, even Web pages (we discuss them below)—so you type a new *find* request and get (let's say) a sixteen-element substream. (It includes the document you're working on, and Feinstein's message.)

You browse through the substream, selectively copy bits into your new document and return to your main stream. Later in the day you switch back to the "valving helium" substream, and it's three documents longer than before: helium valving being a hot topic these days at Blimptronics, of the messages that have accumulated during the day three mentioned it and were automatically trapped on the substream. (They appear on the main stream too.) To send a copy of the manual to your boss at the end of the day, you clone it to the streamhead, maybe add a note at the beginning, press "transfer," identify his lifestream and off it goes.

> In his *Psychology of Information Management*, Lansdale gives an example we cited earlier: the boss wants to see this year's project reviews, but the user has them filed un-

> der each individual project. In a conventional environment that would mean he'd created a separate directory for each project, and needs to visit each one and pull out the report. Under Lifestreams you can create substreams that function (more or less) as conventional directories: you point to a document and press "explicit substream." The system invents a unique id; it represents the id as an icon stamped on the document and on the calling card that gets created for this new substream. You can put other documents on the same substream by pointing to the icon and then a document: the icon gets stamped on the document. This is the natural way to group together (for example) all the separate files that make up a single longer paper. Explicit substreams are like conventional directories, except that you don't make up names for them, grabbing any member gives you access via icon to the rest, a document may be part of arbitrarily many of them, and they are shuffled in with the rest of the stream—thus are all searched together when you do a *find.*
>
> So: you could visit each project's substream individually and pull out the reviews. But you can also gather all the project reviews by means of a single *find* operation, *squish* them into a short summary and copy the reports and summary onto your boss's stream. He is likely to respond with a big raise on the spot.

For your next trick, let's say you need to retrieve information about a particular real-estate deal. The information was collected by a former co-worker who can't help you find it because he has since departed the firm. In a non-Lifestreams world, you face the complex task of find'ing and grep'ing your way through an alien directory structure. Under Lifestreams, you focus your viewport on your former co-worker's stream (assuming you have the necessary software permissions) and type the appropriate query (say "Castle Grande and Web Hubbell"). For your purposes, any pre-existing directory structure is completely transparent.

Simple enough, but how do you point your viewport at someone

else's stream? You call up a substream on your display by clicking on its calling card; you get access to someone else's stream in the same way. Each lifestream is represented by its own calling card. (You aren't likely to give many people unlimited access to your lifestream, but the system allows you to grant limited partial access.) You can store copies of your co-workers' calling cards on your stream, just as you might keep their addresses or phone numbers.

Suppose, though, that you don't have the calling card for a stream you want to investigate. We already have a way of finding things—namely (surprise!) *find*. Ordinarily *find* rummages through your own stream, but you can make it rummage through calling cards for everyone else's stream instead—through the calling cards of every other stream in the world that is willing to be visible to outsiders. (You can, similarly, confine your dog to the inside of your house or set him loose to investigate the outside of everyone else's.)

If you want to refocus your display on Conchita Feinstein's stream but don't have her calling card, you can let *find* search the world for "Conchita Feinstein," starting with "nearby" lifestreams and foraging further and further afield until it reaches Outer Mongolia (or whatever site is most distant from yours as your particular installation sees things). The result will be a substream consisting of calling cards for every Conchita Feinstein in the world—unless you see what you want and terminate *find* in mid-search, as you ordinarily would. (The calling card for your stream carries whatever information you are willing to release; Conchita's might read "Conchita Feinstein, VP for Mooring Masts, Blimptronics Inc., Teaneck NJ.")

§ Scheduling and Reminders

The "future" segment of your stream is as an appointment calendar. If you have four meetings and a doctor's appointment next Wednesday, the "next Wednesday" segment of your stream has five entries. When the time stamped on a future document (say "Wednesday the 27th, 10AM") arrives, the document materializes on your stream as if it were newly-arriving email. You can see the future by resetting the system clock, via the same operation that takes you on browsing missions back

to the past. You can also (as we've said) build a forward-looking substream, in the same way you would a backward-looking one. If you build a "rest of today" substream at the start of each day, all your remaining appointments are trapped on that stream. If new events are scheduled during the day (perhaps by your meeting-maker agent) for later the same day, they are trapped in the substream also.

Most people use their computers to help manage their calendars, but when they need to generate reminders they often fall back on leaving files on the desktop or sending themselves email—*ad hoc* methods that are plainly less than ideal. There *are* plenty of time and calendar management software packages on the market, but the advantage of the Lifestreams system is *integration.* When you need to generate reminders or update your calendar, you rely on exactly the same operations and software tools and the same underlying model as you do in the course of normal document and email handling. The time-ordered stream with its idea of "right now" plus the substream mechanism are all you need to manage your files, mail and calendar. Integrating many capabilities under one model allows you, also, the flexibility to blur or eliminate unneeded distinctions. Calendar items automatically turn into stored documents (migrate automatically with the passage of time, that is, from your stream's "future" to its "past"). If you need a record of all billable hours on the Castle Grande case, you can select and squish a substream in the same way you always do. And you can squish the future as well—to get a synopsis, for example, of all your appointments for next week.

§ Integration and Money Management

Lifestreams isn't a money-management package, but it is a basis for squeezing more value out of existing packages. To write checks under Lifestreams you might, for example, pop a new document on your stream and stamp it with the attribute "check"; a check agent slides down your stream and prints the checks out or carries out the transfer electronically. The check agent isn't part of Lifestreams; Lifestreams merely provides the integrated framework within which the

agent can do its stuff. The same techniques that allow you to substream your files or appointments allows you now, also, to find all the checks you've written this year to Phil's Zeppelins. If you want to track the whole story of this year's business with Phil's, you might need to see email and phone records and appointments too— Lifestreams-style "integration" means that you can trap them all in a single substream. In our current system, for example, squishing the substream created by an appropriate *find* command yields a chart displaying the historical performance of each security in a user's portfolio.

§ Integration and your Rolodex

Let's say you acquired Phil Schwartz's phone number at some point, but can't recall where or when: it might be stuck in email or an electronic bill or phone notes, or maybe you located it in the phonebook and copied it onto a "rolodex card" stream document. To find it, you could make a Phil Schwartz substream. But the fact that Lifestreams puts your entire info-life in one place makes it convenient to turn an agent loose on the whole bundle. Your phone-number agent (we've built one for our prototype system) ranges backward through the stream, looking for likely candidates—phone numbers appearing in the textual vicinity of Phil's name. The time-ordering makes it easy to guess which numbers are most recent. So the agent gathers candidate phone numbers and presents you with a choice: work? home? fax? beeper? It will dial the call for you too, if you like.

You run the agent as you would any application: pop a new document on the stream and run the agent inside. The agent treats the document as a transcript window for user communication. If you tell it to dial, it pops a phonecall document on the stream, which serves as a record of the call and can store your notes about the conversation.

§ Integration and the Web

When you locate an interesting Web site and want to find your way back easily, you create a "bookmark" that points there. But it's easy to accumulate an unwieldy supply of bookmarks. Many software companies have acknowledged that fact by developing bookmark manager applications. Lifestreams, again, supports bookmark and Web-browsing management within an integrated framework. Our prototype works like this: a Web agent watches your Web browser and shadows it. Each time you create a new bookmark in the browser, the agent creates a new URL document on your stream and puts a copy of the new bookmark inside. When you open a URL document in Lifestreams, your Web browser opens and shows you the referenced page—just as your word processor is deployed when you open a text document. The URL document itself stores a copy of the page as it looked when the bookmark was created. Your list of bookmarks (or hotlist) is simply your "bookmarks" substream.

Integration means that substreams can trap Web pages along with other document types, and that passing around URLs is easy. To send someone a bookmark in a conventional environment, you copy it from the browser to the mailer; the recipient copies it in the reverse direction and then uses the browser to open it. In Lifestreams, you simply copy the URL document to the recipient's stream. When he opens it, a copy of his web browser fires up and shows him the page.

In sum: under Lifestreams, the same operation that makes a directory also creates a mailbox, hotlist, phone log or appointment calendar. In each case, pressing "squish" creates a synopsis. Browsing documents in search of a file or mail boxes in search of a letter or next week's appointments to get a feel for your schedule are three instances of the same operation. If you know how to locate the file you want (by creating a substream via *find*), you know how to set up an agent to filter your mail (incoming mail gets tossed automatically into the appropriate substreams). When you return to your machine you can tell at a glance how much mail has collected, and check through it at the sweep of a mouse. You can reorder your entire information world in a few keystrokes. Our prototype realizes all of these capacities.

> Lifestreams is also a good platform for properties that would be desirable in any software context but make particular sense here. Archiving to and retrieving from bulk storage ought to be automatic in every system, but Lifestreams is a strong basis for automatic archiving because of its well-defined browsing model and the attributes list that it associates with every document. When the document goes, its browsable header and attribute list remain behind, to point the user in its direction when it's needed. Your personal file-space ought to be accessible from any net-connected computer running any operating system. But Lifestreams is a particularly good basis for supporting this kind of universal, machine-independent access, because of its small operation set and simple underlying model. Thus the Macintosh environment (for example, by way of contrast) is defined by windows and menus working together with the hierarchical directory structure and the applications in a particular way; you can't change the windows and menus and still keep the Mac environment, because look and feel *is* the Mac environment. Lifestreams on the other hand substitutes a simple mental picture for a particular image on the screen. The stream and its basic operations are easily visualized, and the essence of the system is the user's mental imagery and not the details of the display.

Beyond the Personal Cybersite

The Lifestreams system has big aspirations. It provides a simple, powerful way to maneuver around any environment based on time-ordered data. That capacity is potentially useful all over the place.

Consider TV. There's universal agreement that TV will change radically in the near future; the question is, will it be well or poorly integrated with the internet-based cyberworld? Everyone agrees that a TV station ought to be an archive and not just a real-time data feed. On Tuesday at 10 you ought to be able to watch program material originating on Tuesday at 10 *or* at any previous moment. Clearly, too,

you ought to be able to bring the power of your desktop computer to bear on your TV, searching through archived material, posting agents to trap what you want, stripping out commercials. But what kind of data structure should an "archival" TV station live in? How do you dial yourself backward in time? How do you search for what you want? How is programming information integrated with program material?

The match to Lifestreams is obvious. A lifestream is a network-accessible data archive for time-ordered information, which is just the sort of structure a TV station needs. If a lifestream (in other words) is an infinite pile of cards, you can draw little pictures on each one, thumb the edges and get a flip-book. Edge-thumbing is performed by an agent you hang on the the end of the stream with instructions to take action whenever a new element shows up. (Our current system assigns new documents to the proper substreams by means of such an agent). On a TV stream, an agent would grab each new element and throw it up as an image on your display. When a stream is ordinarily escorted by a particular agent its calling-card can say so, and the agent can be fired-up automatically when you tune in the stream—the way an appropriate viewer application is fired-up when you open individual documents.

Program information in the form of text documents would be interspersed with images (the TV agent ignores interspersed text.) The "future" part of a TV stream functions as a program guide: the same operation (create a future-looking substream) that shows you your appointment calendar for next week will also work, when you apply it to a TV stream, to show you next week's programming. The same operation that dials you back to last April to scrounge around for a report can also take you back to last April to watch a movie you missed. The TV agent can respond to commands like "fast forward" or "search for" or "rewind" in the intuitive way, toodling blithely forward and backward over the stream like a speedy locomotive on a high-quality branch line. (The TV-stream brings up an issue we won't discuss except in passing: a lifestream is a recursive structure. A stream element can itself be an entire stream. Thus "the 7 o'clock news" might be a stream element; peer inside and you see another stream, the succession of frames that capture the program.)

Your TV set in this worldview consists of Lifestreams running on a computer with a biggish display and the appropriate agent. After you unpack it and plug it in, you need to find some interesting stations; you might point *find* at the outside world and type "NBC." You would get back a stream of calling cards for "NBC" streams, and you could tune in an NBC station by clicking on its calling card in the regular way. (The same tricks work, obviously, for digital sound. You can seek out a particular recording of the *Waldstein* sonata or Pea-Brain and the Rap Romeos the same way you look for NBC, pay for your own copy of the stream or rent it by the hour, and set your music agent cruising. The music stream might have program notes or video mixed in. Your favorite-substreams menu might now include email, the report you're working on, C-Span, email from some particular person, a local radio station and your favorite recording of the *Missa Solemnis.* You mouse among them in the regular way.)

The consumer electronics company Thomson recently announced "Genius Theatre," a big TV with a built-in computer and a wireless, remote-control keyboard. "An enterprising move to blur the distinctions between the PC and TV," according to *International Design* magazine's March-April 1996 issue. But those distinctions are likely to blur a good deal more convincingly in a Lifestreams world.

A lifestream is a natural way to organize a certain sort of Web site. Suppose you're a catalog company selling a bunch of different products. The conventional view of a "catalog" is a collection of product blurbs that must be updated (by and large) simultaneously; to change a blurb, you bring out a new catalog. A Lifestreams catalog, on the other hand, is a stream of blurbs. To update a listing, you add a new blurb to the end of the stream. Everything else stays the same.

What good does the time-ordering do? Granted, obsolete product listings (e.g. for last year's ZippoBlimp 230, which has since been superseded by the pricier but more powerful 250) will rarely be of interest and will probably be deleted from the publicly-accessible stream view. But the Lifestreams structure means that you can search the catalog using the same operations you depend on at your personal site—if you're interested only in ZeppoSweaters and Hydrogen Leak Detectors, you can create the appropriate substream, then either browse it or (if

you're in a hurry) squish it. And you can post an agent at the end of a catalog stream to send you new product blurbs as they appear.

The same organizing principles make sense at (for example) a Web site holding a government department's or university research group's technical reports, or a candidate's press releases, or each new edition of a newspaper or magazine—except that, in *these* cases, the historical archive remains interesting in its entirety. Each "Webstream" has a future as well as a past, where upcoming events can be posted. (The "Ada Project," a popular Yale web site for women in computer science developed by graduate students Elisabeth Freeman and Susanne Hupfer, uses Lifestreams to provide web access to announcements, news articles and other web links. Users of the Webstream see a time-ordered list of documents and can search the stream in the ways we've discussed.)

Suppose you and another Lifestreams user want to chat online in Unix "talk" style. One approach: you spawn a new Lifestream, and you each tune it in. (The streams we've discussed so far have been long-lasting, but lifestreams are merely data structures and can be created and destroyed dynamically as the need arises.) To make a comment, you pop a document on the stream and put the comment inside. Of course the Lifestreams-based conversation (versus the Unix talk version) supports any number of participants, and allows chatters to join in asynchronously: a third party might add new comments to a conversation that took place yesterday, assuming the original participants allowed their "conversational lifestream" to hang around.

But notice, now, that there is no fundamental distinction between this sort of software conversation and a network bulletin board. Suppose we store bulletin boards in lifestreams: users once again acquire the power to browse, search and squish the same way they do at home.

Given "chatstreams," it's easy to see how phone calls could be stored on lifestreams too. Each party to the call runs a "phone agent" that translates speech into data packets and tosses them on a phonecall stream, while concurrently grabbing each successive packet from the stream and turning it back into sound. To phone Schwartz I would spawn a stream plus an agent, then send Schwartz the stream's calling card. If he wants to take the call he opens the stream, his own phone agent fires up and he's in business. Phone ringing is handled by

an agent that sits on the end of your stream and makes noise when a call-request document shows up. If you are in frequent phone contact with some party, you can allow the phonestream you share with him to persist; either of you can place a call (or leave a message) by re-opening the shared stream.

Again, integration yields benefits. To find out what calls I've missed when I return to my machine, I build a substream of call-requests. (Callers who failed to reach me can leave answering-machine-type messages on the phonestream they had intended to hold the conversation.) Lifestreams now provides a complete filing system for phone calls and phone messages. An agent can forward my calls by copying an incoming call-request document to some other stream. If you want to identify yourself in hopes of convincing someone to take your call, you can type a message into your call-request document before shipping it off. Setting up a conference call is trivial: you send your request to many streams instead of one. Because your TV is also your phone it can pause a program when you get a call, and pick up where you left off when the call is over.

It's widely agreed that services like phones, TV and internet access will be unified in the future; there's nothing new in that prediction. But merely to make the prediction begs the important questions: will these services be *integrated?* How? How will users negotiate the new communications landscape? Will ninety percent of them make do with two percent of the system's power, because they can't figure out how to get the rest to work? Will the communications landscape become a slap-dash, complex pastiche along the lines of today's computing landscape? Or do we stand a chance of achieving power and simplicity and integration—in other words, elegance?

In Sum...

If you master the Lifestreams dashboard, you know in essence how to deal with files and calendars, mail and phonelogs and hotlists—and how to work your fancy new digital TV set, how to deal with a wide range of bulletin boards and Web sites, how to set up a chat room or a conference call online (just spawn a stream and hand out calling cards). And these

capacities are delivered in a context where you no longer worry about managing your file and disk space, no longer care what base operating system a machine is running and can dive into your private information world at any gas station, karaoke bar or McDonald's in the country.

There's a certain class of computer user who enjoys learning new systems, puzzling out new ways to get things done and lugging around portable computers. For those of us who aren't in it, the Lifestreams system is a promising way to cut down on the time-wasting, brain-wasting overhead of operating in cyberspace.

Lifestreams in context

Because Lifestreams undertakes to perform such a wide variety of functions, the number of systems it resembles in one way or another is gigantic. Relevant points of comparison include file indexing systems, information retrieval and filtering systems, corporate document and archiving systems, personal information handlers, time and contact managers, workflow systems and financial managers, among others. The relationship between Lifestreams and these other applications is an important topic and we have surveyed it at length elsewhere; here is a brief overview.

Note first that, so far as we are concerned, the Lifestreams system's appeal isn't a matter of its being able to carry out new kinds of information processing that other systems can't. It depends instead on the versatility and simplicity of the Lifestreams tool-set and underlying model. Among competing tool-sets whose capacities are (ultimately) equivalent, the choice comes down to subjective, aesthetic judgements. That doesn't mean there are no useful experiments to run; we are in the process of running some simple ones designed to gauge a user's productivity under Lifestreams versus a conventional Unix environment with "standard" tools. We have to confess, though, that even if the experiments fail to turn up any measurable productivity gains, we are extremely unlikely to stop using Lifestreams ourselves. We have tried to explain why certain users might find the system attractive independent of any measurable performance gains. Some key issues are

well-summarized by Andrew Corradini, Manager of Internet Services for Phoenix FiberLink, a Western U.S. data and telecommunications provider (he is an observer of the system rather than a user):

> I have so much stuff coming in my "InBox" daily, whether it's incoming e-mail, snailmail, phone messages, articles, or what-have-you; that there's not really time to organize it all. Rather, as you quite convincingly point out, I'd rather just STORE it all (since storage is cheap!) and access only what I want when I want to access it.

Corradini believes that "Lifestreams is a fundamental paradigm shift..."

It's worth recalling the formal studies that have shown the Macintosh environment to be more productive, in well-defined ways, than various PC environments. Those studies were interesting—and yet, in a sense, they were doubly irrelevant. Macintosh enthusiasts didn't need them; their enthusiasm tends to be based on the machine's subjective appeal. Nor have the studies been sufficient to put the Macintosh over in business environments dominated by the PC. Subjective factors dominate in that case also.

Lotus Notes is today's dominant "workgroup" application. Both Lifestreams and Notes maintain document databases and allow custom viewing and filtering of any database (via a "view" in the case of Notes). In both systems email is tightly integrated with document handling. But Notes is intended mainly as a corporate document system, and its databases are optimized to the needs of general audiences as opposed to single owners. Notes databases are statically configured, and database filters are created not by typical employees but by system administrators. Although Lifestreams supports collaborative work, a single-user "stand-alone" Lifestreams system makes perfect sense; stand-alone Notes makes less sense.

Systems like the MIT Semantic File System, Manber and Wu's "Glimpse" and various versions of the Macintosh *find* application allow files to be retrieved on content rather than name or location. Content-based document retrieval is central to Lifestreams also, but in a way that is integrated with document creation, time management and communication as we've described.

Lansdales' "Memoirs" system used time and content-based search as a basis for organizing personal information on Macintosh systems. Ben Schneiderman and Catherine Plaisant's "Lifelines" is a system for producing graphical, timeline summaries of historical information; their techniques are potentially valuable in the Lifestreams context.

Lots of other systems bear comparison to Lifestreams; Eric Freeman discusses them in his dissertation.

Conclusions

Big predictions are a risky business at best, and most people wisely avoid making any. But life would be less interesting if no-one ever went out on a limb. We are about to go out on one.

The internet, the information highway and associated software (like the Web and Java) have yet to find their killer app. Web browsing itself is sometimes spoken of in those terms, but it's unlikely to drive the development of a network environment substantially more capable than today's—and it is unclear where the ultimate commercial payoff lies in a system whose main function, in business terms, is to make advertisements and product descriptions available to people who choose to ferret them out. Advertisement-deprivation is not a problem many Americans struggle with, whether they are habitual Web browsers or not. Web shopping is a promising area. Yet we tend to believe that, in a period as enlightened as ours, in which anyone with a stack of catalogs and a credit card can shop up a storm without ever stirring from his Barca-lounger ... net shopping won't change many lives. Video on demand has significant business potential. But demand video could be delivered by a more limited and specialized infrastructure than a full-blown info-highway, and the whole prospect raises an interesting question. How many families will spring for the incremental cost of the required network services, versus the far-more-modest requirements of an expanded cable-TV world offering (say) 500 movie channels instead of five? The answer, some industry insiders believe, is "not many."

So here's our prediction: Lifestreams (or something else with a different name and realization but the same purpose and basically the

same design) is *the* information highway killer app. It offers something new, powerful and different, a service that will pay off daily in real life and can't be realized (in all its glory) in any other way; a service that needs the information highway and milks it for all it's got. The big commercial opportunity won't be selling the base Lifestreams system; it will be renting out Lifestreams service, and selling the custom agents, squishers and viewports that support fancy special-purpose environments for libraries and hospitals, fertilizer factories, supermarkets, film buffs, *et. al.* (You read it here first!)

Lifestreams is a three-phase project. We are near the end of the first phase. In phase one we build a stable, robust implementation for Unix workstations; servers handle a modest number of clients (less than ten), and provide an interface between the Lifestreams world and the surrounding internet (making arriving email appear on lifestreams, for example). Our intention at the end of this phase is to distribute the Unix implementation to interested users via internet. In phase two we port the system to Windows and Mac and add some important refinements (for example in attribute-handling and *find*); in phase three, we develop full-blown utility servers. The system will be valuable, we think, even if it never gets anywhere near the ambitious TV and phone integration we've proposed. In any case we can't help believing that an integrated, elegant communications landscape would be of value to nearly everyone, and that Lifestreams is a strong basis for achieving one.