

Information Programming for Personal User Interfaces

Stephen Farrell Volkert Jürgens Christopher S. Campbell Paul P. Maglio

IBM Almaden Research Center
650 Harry Rd.
San Jose, California

{sfarrell,volkert,ccampbel,pmaglio}@almaden.ibm.com

ABSTRACT

With widespread access to e-mail, the world-wide web, and many other information sources, people now use computers more for managing information than for managing applications. To support how people naturally and routinely organize information, computers ought to be able to reflect the categories, relationships, and cues that people rely on when thinking about and remembering facts. Toward this end, we created an *Information Programming Toolkit (IPtk)* layer that sits between user applications and the information sources with which the user interacts (e.g., file system, databases, email, web, and so on), collecting application-independent properties, indexing documents along many different dimensions of these properties to create a personal record of information use, and providing convenient means for information access. The IPtk enables the development of smart user interfaces that automatically tailor information to a user's history and context of information use; for instance, displaying a list of documents related to the current document in time or content (rather than simply those that were opened recently). Moreover, the IPtk enables the development of user interfaces that look nothing at all like today's user interfaces; for instance, saving and retrieving quickly jotted notes without having to explicitly name them and then recall their names. In this paper, we explore the notion of information-based computing in some depth, describe our general architecture for supporting it, and detail our prototype implementation.

INTRODUCTION

With recent increases in network bandwidth, computational power, and data storage, people are now confronted with vast amounts of information. Scores of email messages, web pages, and files comprise merely a fraction of the information that might capture a typical user's attention at any time. Most of the time, disparate information sources are stored and accessed by disparate applications (e.g., an email program for email, a web browser for web pages, a document editor for text documents). Yet this organization—

information compartmentalized by specific applications—reflects more historical development than carefully considered user needs. In attempting to recall information that has been seen before, users typically remember bits and pieces, for instance, where and when they saw something, who they were with, or a few key words or phrases. That is, human memory for facts and episodes is not organized solely by application use, but indexes also include history, context, and content, among others [26]. It follows that to support human memory, computer memory ought to mirror the categories, relationships, and cues that people naturally use when thinking about and remembering information. Given modern computational power and storage capacity, users no longer need to be forced to think in terms of applications alone. By providing a uniform layer for information storage and access between applications and existing information sources (e.g., file system, databases, email, web, etc.), we developed an information infrastructure that enables computer systems to categorize information in more human terms, enabling storage and access along a variety of dimensions in an application-independent way. Our infrastructure, or *Information Programming Toolkit (IPtk)*, provides a programming layer between low-level information storage and access (e.g., email, files, web pages) and high-level user concerns (e.g., history, context, content) that enables the creation of smart, personalized, and adaptive user interfaces.

Smart, Personal, Adaptive Interfaces

The benefits of an information- and context-aware substrate become apparent when considering how present user interfaces fail to meet user needs. Suppose Bill has stored some facts and figures for today's meeting with Maria somewhere on his computer. He had to record the information quickly a few days ago and now does not remember exactly where he put it. This meeting information might be anywhere in the file system or it might be associated with any number of applications. Meeting information might be stored in the email application's inbox, outbox, or elsewhere in that application's folder hierarchy. Meeting information might also be found in the calendar application's database, on web-pages, or in a text file. Of course, each application might have its own search function to speed up the process finding information. To use these application-specific search functions, however, Bill must ignore the most significant cues he can remember about the appointment. Say he remembers what he is looking for as "The info for the meeting with Maria that I read when Rob was in my office—I think it was around Friday." Do any of the applications know about

Maria? Or when the information about the meeting was last opened or read? Or when Rob was in the office? In fact, for standard sorts of text search functions, the only useful information Bob has is the string “Maria,” which might be found in the text about the meeting information (and also in many other places as well). Bill’s problem of finding this relevant information would be much easier if he could simply express his query in the terms he remembers.

In the same way, storing newly created information can be as troublesome as finding old information. Most systems force users to attach certain kinds of properties to files or documents, such as a filename, file type, and location in the file system. These properties are not always useful for later finding or retrieving information, and typically adds a great deal of “file management” overhead. Suppose that Sydney works with large numbers of text reports that are related to various projects. She is careful to organize this information using a well thought out file naming scheme and a unique directory for each project. Every time someone sends her a new version of a file, Sydney must rename it to match her convention. Not only that, but the number of projects has grown so much that Sydney must now search through the all the directories every time she needs to locate a file. Finally, because some reports are associated with multiple projects, Sydney puts the reports in the “most central” project folder, resulting in lost reports and revision control problems. All these problems would be mitigated if Sydney could simply attach properties to the documents to reflect the project, and then could query the system by project, date, or version without worrying about names and directories.

Novice computer users face a daunting task trying to manage information when they do not know which applications are used to work with which types of information or where information ought to be stored in the file system. Suppose Ted is trying to find an image file in which he was creating a picture to be used in a presentation. As a novice user, he does not remember the name of the file he saved, and he is not familiar with the default directory into which the application saves files. Without knowledge of standard image formats (e.g., BMP, JPEG, TIFF), he cannot simply search the file system by file extension. Ted is forced to seek out an experienced user who knows how to find files using a set of search heuristics to locate the image. For instance, the experienced might ask, “What application were you using to create the picture?” If the answer is Photoshop, all files with the extension “PSD” can be located and ordered by creation date, which might in fact locate the file. Yet storing files and information with the right properties would give Ted and other novice users a more natural way to manage information; for example, he could simply search for “the last image file I worked on.”

By capturing the context of events surrounding the use of some piece of information, users can rely on many different cues to help them locate and manage information. Suppose David has a hard-disk full of music files (e.g., MP3 format) that are stored in a single, huge directory. He wants to listen to a song he heard on Monday, but does not know the title or band. The only information he can remember is that the song played after the song “Memories Can’t Wait” and before he started replying to email. Because he listened to

more than 50 songs on Monday, simply searching for “songs I listened to on Monday” would not be very productive. In this case, he would need to get the list of the 50 songs and then listen to all those he did not recognize to determine if it is the correct one. By storing events as well as file properties, the system can effectively use the cues David remembers to retrieve the songs heard on Monday after “Memories Can’t Wait” and before he started working with the email application. This sort of query would likely produce in a manageable list of songs for David to sort through. Storing the information in the context of events provides powerful hooks for finding information, allowing people to use episodic memory to pin-point buried data.

Automatically adding all sorts of properties that can be dynamically indexed allows users to find and retrieve information in *unexpected ways*, that is, given connections or associations that could not be predicted prior to information retrieval and may have arisen due to chance. Suppose Gail needs to find an article on “new display technology” that she read over six months ago for a project proposal that is due by the end of the day. It has been so long that she cannot remember the title, author, or the name of the display technology that the article discussed. Searching by keyword will be useless because she does not know the area well enough to formulate good keywords. The only thing she remembers is that on the day she read the article, Joan came into her office very upset about a personal problem. Because “people in the office” is part of the context that Gail’s system collects, she can quickly locate all articles read more than six months ago when Joan was in her office. At the time, the context of “Joan in my office” appeared to be merely circumstantial, hardly the sort of context information that could matter. As we have seen, however, Joan’s visit provided a salient event in Susan’s memory and the needed association for reaching the article.

In general, users want to...

- think about information rather than applications
- annotate and relate information
- rely on history of information use
- enter information into system easily
- search for information in natural terms
- browse information along various dimensions
- find information that is related to other information

How to Support User Memory

In the physical world, people manage personal information by collecting, collating, and manipulating physical objects such as paper, folders, books, notes, desks, shelves, and cabinets. Arranging information objects in physical space (e.g., books standing on a bookshelf, papers piled on a desk, files alphabetized in folders) is a highly effective means for maintaining certain relationships among information and for reminding people of those relationships [25]. Spatial location provides powerful cues for indexing human memory, as do other aspects of context, such as time, activity, and emotional state (for an overview, see [1]). In trying to recall

information, people routinely reconstruct previous settings, for instance, to try to determine where and when something was seen last. When organizing information to aid in finding it later, people often rely on conventions, such as alphabetizing by author's last name or piling notes with phone numbers by the phone. The key to retrieving information from memory is to *set up* memory for the information using indices that are likely to be helpful when retrieving it. This sort of elaboration to support retrieval might be done deliberately and can develop as an automatic skill with practice (e.g., [15]).

Information storage in current computer systems provide limited support for human memory. For a file system, for instance, location in the directory hierarchy is paramount, and this tends to reflect conventional organization (e.g., UNIX's `/etc` directory is conventionally used for configuration data) or arbitrary organization (the `goat/hacks` directory might contain code for the "goat" project). In addition to hierarchies, file systems often provide certain file properties (such as creator and access date), links allowing one object to appear in several places in the hierarchy, and text search. Each of these gives users additional means to index and access files, adding to the cues people can use to retrieve information. Some graphical user interfaces (GUIs) that rely on the desktop metaphor (e.g., Macintosh, Microsoft Windows) let users position files and other information spatially on the screen, adding spatial relations to the set of cues users can rely on, though there is some dispute over whether this sort of two-dimensional and three-dimensional spatial information provides effective memory cues (cf. [22, 30, 9]). For file systems and desktop GUIs alike, users manually organize information, whether in hierarchies or in spatial positions on the desktop, and manually retrieve information, whether by searching by text or category or spatial location.

The potential for computational systems to support a user's memory for information was perhaps first described by Vannevar Bush [7]. Bush's pre-computer vision of a "Memex" was a kind of desk that could store a vast amount of information and allow its user to annotate and associate the information at will, making retrieval easier by providing arbitrary and customizable indices. The development of hypertext systems for this purpose (e.g., [14, 20]) enable linking and annotating information in the context of personal information use. The general notion of hypertext led to the development of systems for creating creating and publishing documentation (e.g., [31]), and ultimately to the development of the world-wide web [6], which is a specific approach to linking information on a global scale. In most cases, hypertext systems require explicit action on the part of the content creator to establish the links or indices for retrieval.

From a cognitive perspective, supporting a user's ability to retrieve information that has been seen before means capturing information in context, associating information with context, and associating information with other information. In short, the key is to relate information to anything that the user might consider relevant so that later it can be accessed given almost any reasonable cue. To support document retrieval, file systems provide indices such as file names, directory structure, access date, and text search. In this way, file systems are rigid, giving users a

few fixed means for access. By contrast, hypertext systems are much more flexible, providing indices along arbitrary, user-defined dimensions that link documents together. These schemes privilege documents over information activities (cf. [2, 10]). Of course, approaches that mix document use and other user activities are also possible. For instance, "Semantic File Systems" can induce hierarchical structure and other properties from document contents, effectively organizing documents for access [19]. In the same way, the "Presto" system can automatically extract properties from documents, enabling connections among documents to simplify access and other document management actions [12, 13]. "Lifestreams" structures a user's information space in terms of history of use, meaning that access depends on time and sequential order [16, 18]. Whereas Lifestreams focuses on time, the "Forget-me-not" system incorporates both time and other context information to create a personalized sequence of episodes or activities that can be accessed by time [23]. "Time-Machine Computing" provides the same sort of function, but emphasizes the ability to reinstate previous workspaces [27]. The "Remembrance Agent" collects document contents and certain context information (e.g., physical location), indexing these for later retrieval given similar content and context [29, 28]. In all such systems, certain properties are extracted from documents, certain context information is derived from sensors or other means, and specific processes relate these together for to support specific information tasks. No single system covers all sorts of properties, context, and retrieval—yet this is exactly what is needed to best support users' memory for information.

INFORMATION PROGRAMMING TOOLKIT

The IPtk provides a tool for programmers to take account of arbitrary properties and context, to index information along any possible dimension, and to access information given whatever cues make sense. That is, the IPtk is not constrained to use certain properties or context information, or to present a user interface for accessing information in any specific way. Conceptually, the IPtk is a programming layer between user-level applications (such as document editors, web browsers, email readers) and all of the user's local and network information sources (such as file systems, the web, email servers). The IPtk provides a uniform interface that takes advantage of a user's history to personalize the user's experience [4, 24]. In fact, one application driving our development of the IPtk was a "Personal Library", a single point of contact between the user and his or her history of information use. Because the IPtk can automatically index information in many different ways, the user can rely on a Personal Library to retrieve information when it is needed—regardless of how the information is remembered. For example, a user might recall an important email message as "the one I received from Fred before my trip to Acme Inc. one morning," a paper written as "the one that referred to that paper by Einstein," or a quick note as "the note I took while talking to Barney on the phone." By standing between the user and information use, deriving properties and making connections among these, the IPtk can easily support exactly this kind of personalized and smart user interface.

As a toolkit, the IPtk provides abstractions that enable operations on information and information use at a level some-

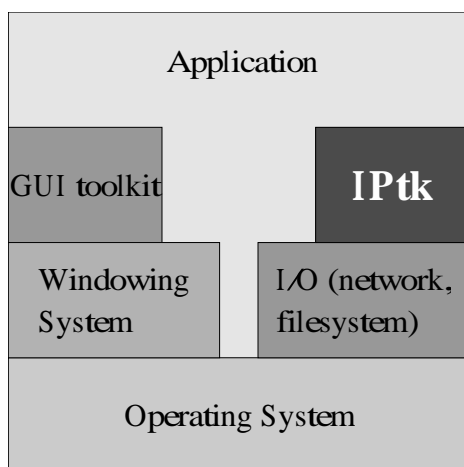


Figure 1: The Information Programming Toolkit provides an abstraction layer between low-level I/O and application programs, just as GUI toolkits often provide an abstraction layer between user interface programming and low-level drawing operations.

what higher than operating system input and output, similar to the way a GUI toolkit might provide abstractions for developing user interfaces rather than for drawing on the screen (see Figure 1). The analogy to GUI programming also reminds us of the “Context Toolkit”, which relies on GUI-like widgets as programming abstractions for context [11]. The programming interface in the IPtk is not so closely tied to specific metaphors of GUI programming.

The fundamental construct in the IPtk is the *event*, which can be used to capture document contents, document use, and any other context information that might be available. To set up easy access along all dimensions contained by events, the fundamental storage process in the IPtk is *promiscuous indexing*, which actively stores information contained by events in as many ways as possible. To access information, the fundamental retrieval process in the IPtk is akin to a database *query* run over all applicable indices. We now discuss each of these fundamental concepts in turn.

Events

Each document or piece of data the system encounters is packaged together with a timestamp to create an event. The event is implemented as a compound document comprised of a document containing at least the URL and timestamp, and optionally one or more payload documents. For example, a web page might be represented by a set of three documents including the document containing the URL and timestamp, a document representing the HTTP header, and an HTML document. Applications built with the IPtk can construct events that are indexed whenever it is appropriate. Events are also what are returned when the application submits a query. Some legacy applications can be extended to create events, others require the use of intermediaries to detect the access of documents. Once introduced to the system, the event is given a unique identifier number (*EID*). The system indexes each document within the event by the EID, thereby

ensuring that the relationship between those documents is preserved.

In addition to encapsulating documents introduced to the system, events can represent information that is not stored as a document. For example, information such as keystrokes, movement of people in and out of an office, and changes in the weather can be encoded as events. The more kinds of events that are indexed by the system, the more different types of cues the user can rely on for retrieval.

Promiscuous Indexing

Typical information management systems index documents using a pre-defined, static set of features. By contrast, our approach is to index promiscuously, using many diverse (and possibly highly specialized) document and event features in concert. Because people often rely on idiosyncratic cues to recall information, the utility of a system intended to support human-like retrieval depends on its ability to capture as many dimensions of information as possible. For applications, indices might be generic (e.g., based on document creation time) or highly specific (e.g., based on whether Java code contains inner classes, or implements a particular interface). Some users might find it helpful for the system to pay attention to context information, such as whether the office door is open, whether the laptop or the desktop computer was being used, or whether a colleague was in the office. The power of promiscuous indexing is apparent when the user recalls only a few features of what he or she is looking for or a few aspects of the context it was seen in, and can express these through a single query to find it.

Though there might be many different kinds of indices, we have found it helpful to think about several specific categories of them.

Physical properties of documents, such as size.

Semantic properties of documents, such as title.

Domain-specific properties of documents, such as inheritance relationships for object-oriented code.

Associations between documents, such as hyperlinks.

Clustering documents, such as semantic grouping.

Search features such as regular expression search, text search, and search for image or sound content.

External properties about the context outside of the computer, such as physical location, presence of people or objects, and weather.

Querying

Existing operating system infrastructures expect applications to provide a full path or URL to a file. The task of finding that object is either not provided at all, or is supported by a “Open File” dialog box and possibly a facility for storing a list of recently opened documents. By providing the powerful abstraction of querying on various attributes, the IPtk enables applications to rely on information-awareness. Moreover, it can be helpful to build reusable components that encapsulate certain search functions that are useful across applications. For example, the “Open File” dialog

could be modified to support sophisticated query forms. The “recently opened documents” function could show documents opened with compatible applications. These enhanced “Open File” search and “recent documents” features can be thought of as “widgets” of the IPTk. Others widgets might list a set of documents related to the one currently open, or list the documents that pertain to a scheduled meeting, or a particular person being met. Still others might provide a time line of the modifications of a particular document, or might support browsing of related documents along many different dimensions. Because these are information-management tasks—rather than tasks specific to a particular application—it makes sense that they be available in a toolkit at a layer between applications and I/O systems.

ARCHITECTURE & IMPLEMENTATION

Our Java-based implementation of the IPTk was conceived as a background or daemon process that is informed of various information and sensor events. Intermediaries are used to generate information events when documents are viewed by legacy applications such as web browsers. Other applications can be extended to send events to the daemon. We also created new applications, particularly for gathering sensor data, that are written directly with our toolkit. In addition to a system for monitoring information and environmental events, the daemon contains a query processor for quick attribute-based event retrieval.

The goal of our design is to provide an extensible query substrate to applications for managing information. We want the system to be able to be enhanced to support different binary formats, sources and kinds of data, to pay attention to new features of documents, context, and other data, and to support innovative information-driven user interfaces. To meet these goals of modularity we have divided our system up into different functional units, each of which is extendable with pluggable code modules.

Format Interpreter

One subsystem is responsible for interpreting application-specific data found in documents. Components in this subsystem can answer questions about documents, and are responsible for the semantic mapping. For example, a Java source code format interpreter might determine that the title of a document is the fully-qualified name of the main class; an HTML interpreter might determine that the title is the value between the TITLE tags in the HEAD section of the document. The author of a format-interpreter module is responsible for mapping the set of known attributes onto a specific data format. Format interpreters are similar to *transducers* [19].

The format interpreter module is also responsible for recognizing a document that it can decipher. When a new document arrives, an instance of the format interpreter is attached to the document with a specificity score. In this way, many format interpreters can be attached to a document, but the most specific takes precedence in the case of conflicts. For example, an HTML document might also be recognized as an ASCII text document, but these different format interpreters would have different answers to some questions, such as the number of words in the document, as HTML tags would not be interpreted as words by the

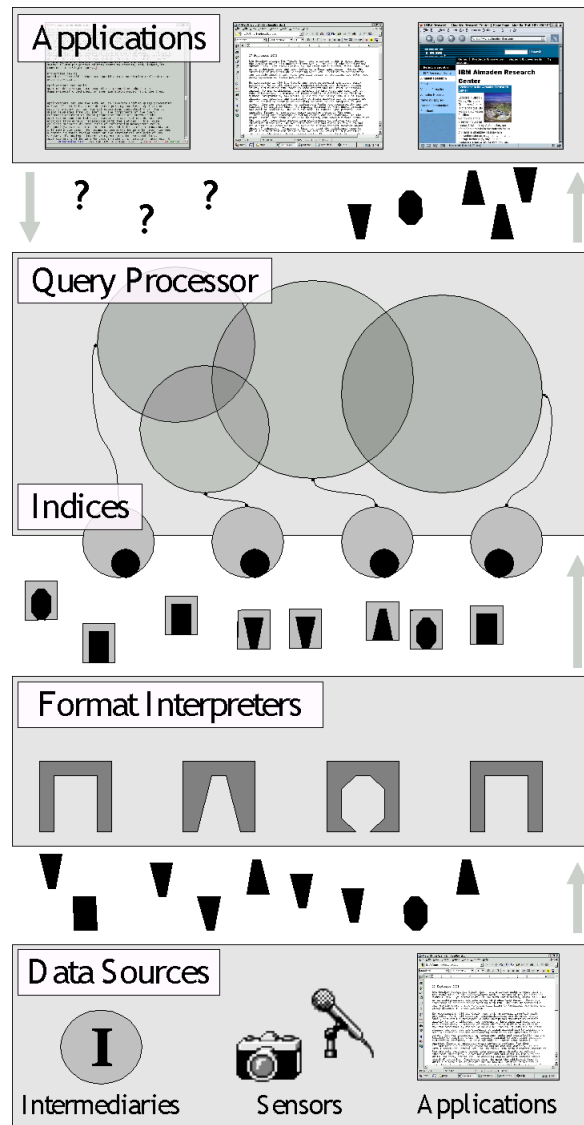


Figure 2: The system is comprised of four functional elements: (a) sources where data enters the system, (b) interpreters or transducers that extract meta data from documents, (c) processes for indexing and querying, and (d) new and legacy user interfaces and applications that can now be made information-aware. Documents and events, represented here by variously shaped blocks, are delivered to the format interpreters by the information sources. These might originate in intermediaries that monitor data streams, including web and other legacy applications, sensors that provide events about the environment, and IPTk-enabled applications. One or more format interpreters bind to the documents, providing common interfaces that make their properties visible to the indices. Applications pose queries to the query processor, which performs set operations on the data set and returns the matching set of documents.

HTML format interpreter. In this case, the value returned by the HTML format interpreter would be used because it had attached to the document with greater specificity.

Indexing

The indexing subsystem is responsible for returning a set of previously accessed documents that meet some feature criteria. In order to prepare to answer this question expediently, index components are given an opportunity to examine all documents added to the system. These components do not need to know about the details of one binary format or another—rather, they pose a question in abstract terms and the format interpreters attached to the document are consulted for an answer in specificity order. Some indices might pay attention to temporal information, some to text, others to document URLs, and still others to associations made by users between documents.

Properties

The property subsystem is responsible for mediating between the interpreting, indexing and querying subsystems. Our system has a core vocabulary of well known properties (or a “schema”) that is used to characterize documents. Properties are represented as classes that implement the “Property” interface. Some core properties include time of creation, MIME type, size, and value as ASCII text. The key to extensibility of the system is the extensibility of the property vocabulary. Core properties are highly generic and can be shared by many applications. Other properties, such as those a developer might add to interpret his or her own source code commenting style, are highly specialized and probably cannot be shared by many applications. Though there will always be tension between interoperability and expressiveness, our system allows properties of both sorts to coexist and to be combined in a single query.

Querying

Applications can use the IPtk to leverage unified query processing across all indices and document parsing modules. Applications specify a query as familiar set operations, conceptually similar to query languages such as SQL. The query criteria are given by partially specified Properties that all returned documents will match. For example, a query criteria of a TimeRangeProperty with only a start specified will return all documents with a time property after that start time. We have found that many applications demand sophisticated features (such as results that are sorted, grouped, or partial) to allow, for example, a query of all documents modified in the last day sorted in temporal order and grouped by MIME type.

APPLICATIONS

As programmers, we wanted the IPtk to provide a set of tools and services for developing smart, personal, and adaptive user interfaces. As users, we wanted a *Personal Library* that we could each rely on as a personal information repository, supporting application-independent information storage and access, and supporting natural information retrieval. In fact, the IPtk enabled us to implement large pieces of a prototype Personal Library that we have been effectively using for several months. In addition, the IPtk

also enabled us to develop a new style of user interface for handling information, the *Information Landscape*.

Personal Library

Our Personal Library can track the use of many information sources, including email, calendar, world-wide web, and file system. To do this, several helper applications serve as intermediaries between user applications and low-level methods of information access, delivering events to the IPtk. In the case of email, for instance, we created a number of intermediaries for different email servers, including Lotus Notes and IMAP, to track email usage. In the case of the web, we relied on simple WBI plugins [21] to monitor web pages viewed and send information access events to the IPtk. In the case of the file system, we used several approaches to capturing information access events, one was to add file-save hooks to applications that would allow this (e.g., EMACS), and another was to modify a SAMBA server to monitor file access. In addition to helper applications for discovering information events in ordinary computer use, we also created separate applications that extracted context information from sensors in the environment, such as using Radio Frequency Identifier tags (RFIDs) to track people in the office. Moreover, Personal Library users could easily save notes or text in the system without naming or otherwise locating them by dropping text onto a special desktop icon, and information in the system could be manually annotated and linked together using a simple GUI.

A variety of format interpreters were written to extract properties from the events delivered by the applications, and several indices were created to provide convenient access to information given the properties. In particular, we created formats and indices for properties including time, key words, MIME types, document associations, names (URIs), and people. These indices enable users to query the system to find documents using combinations of any of these properties, for instance, “the MS-Word documents I looked at yesterday,” or “all text files containing the the word 'Java' that I saw when Sydney was in my office.”

Simple queries over the indices are just one user interface to the information stored by the system. We created several additional views onto these data to allow users to browse through their histories of information use. One view presented all information events on a calendar, making it easy to browse through what was done on previous days or weeks. These calendar views could be filtered by any query to have the calendar display, for instance, all email messages written to a certain person. Another view presented all stored information as a file system (implemented using a SAMBA server). With this view, any of the properties could be used to determine hierarchical relationships for browsing.

Information Landscape

In a sense, the Personal Library prototype connects the IPtk to familiar applications and ways of interacting with computers. Yet our information-programming approach lets us also create novel interfaces for information management. The Information Landscape was our first experiment along these lines. We mapped the user’s history of information use into an easily navigable space using *semantic zooming* [5] to create a of bird’s eye view of a user’s electronic world.

The position of documents and events in this space is determined by grouping along the indices we developed for the Personal Library, namely time, people, MIME type, and so on. The meaning of zooming in this case depends upon the index along which documents are grouped; for instance, if documents are grouped by time, zooming changes the time scale of the view. Another feature of the IPtk we exploit is the ability to register for events that match a query so that new documents automatically appear in the proper place on the Information Landscape. In addition, special landmarks located on the landscape implement live queries. Matching documents cluster around the query landmarks, changing as the information in the system changes. Finally, in addition to representing a view of the user's information space, the Information Landscape allows the user to link documents together and to add text annotations.

Related Work

As mentioned previously, several research systems have aimed at similar goals and had similar approaches. Perhaps the closest work is Xerox's Presto [12, 13], which is a toolkit for building applications around "placeless documents". Like our IPtk, Presto is meant to enable task-based information access rather than the usual location-based or application-based information access (cf. [3, 17]). As a toolkit, Presto provides many of the same features as the IPtk (e.g., extendable means for pulling properties out of documents, and for indexing these), and enables many of the same applications (e.g., their Vista interface corresponds closely to our Information Landscape). Yet Presto focuses squarely on tasks involving documents, whereas the IPtk is more general: document-based tasks are a single category of the information and context events the IPtk handles. Because Presto is concerned mainly with indexing document properties, other sorts of context information (such as "when Bill was in my office") cannot be used to cue access to information. Thus, the IPtk is a generalization of the Presto approach in which arbitrary properties can be defined and used to support user memory for information.

Considerable work has gone into understanding how to integrate heterogeneous data sources into a single query space, and how to export these databases as semantic file systems (e.g., TSIMMIS [8] and SFS [19]). These projects focus primarily on solving two problems that we also encountered: (a) creating a unified query space across heterogeneous data, which includes the problem of finding a mapping for data from one schema to another; and (b) mapping such a database into the file system name space. Though the query processing in our IPtk could certainly be enhanced by some of the techniques and tools developed by these projects, we have chosen to focus on different problems. In creating a toolkit for developing personal interfaces that take account of a user's history of information use, we found no real need to add constraints among heterogeneous data sources or to rigorously define mappings between data schemas, as some false positives and false negatives are acceptable to users.

From a user interface perspective, another related project is Xerox's Forget-me-not system, which continually records a user's history of information activities to capture properties that might be useful for information retrieval [23]. The sort of "episodic memory" that Forget-me-not creates is meant

to support the way user's would normally recall past information actions in terms of document properties and external context information as well. In this way, Forget-me-not is a particular application of our IPtk approach that incorporates a specific set of events, properties, and query types.

Lifestreams privileges time as the primary means for indexing previously seen information [18]. Because it relies on a particular aspect of context to support user memory and information access, Lifestreams can be easily implemented as an application of the IPtk. Similarly, the Remembrance Agent [29] privileges text-based properties and context (and a few others [28]) for indexing and retrieval. Like Lifestreams, the Remembrance Agent can be easily implemented using our IPtk, as it too relies on only a few of properties, indices, and types of queries for supporting user memory and information access.

The approaches reviewed here all focus on specific applications, context, and technical issues to help users manage and organize information. Our approach is to support users' information needs more generally by providing flexible, dynamic indexing of multiple document and event properties. In this way, the IPtk is a cognitive approach that enables the development of smart user interfaces—interfaces that tailor information to a user's history of information use. The approach is complete enough to support recall of information in unexpected and idiosyncratic ways, to help novices users intuitively manage information, and to simplify recall by allowing natural specification of retrieval cues.

CONCLUSION

Though computer use has become much more information intensive over the past few years, user interfaces and systems for organizing information have failed to keep up with the sheer amount data users routinely confront. Because users naturally recall information in context and by association to other information, we developed a high level information programming toolkit that support the development of applications that allow users to refer to documents by semantic structure, relationships, and context. Promiscuous indexing—the liberal association of data along many dimensions—can anticipate the kinds of relationships that may be helpful for recall in the future. Our query processing system and application toolkit enables legacy and new applications to leverage the information- and context-aware subsystem, enabling users to access information in natural and intuitive terms. Insofar as computers are used to view and exchange information, we see applications that help associate, annotate, search, browse and discover relations between documents becoming the normal mode of interaction in user interfaces.

REFERENCES

1. Baddeley, A. *Human memory: Theory and practice*. Allyn and Bacon, Boston, MA, 1990.
2. Bannon, L., Cypher, A., Greenspan, S., and Monty, M. L. Evaluation and analysis of users' activity organization. In *Proceedings of CHI '83* (Boston, MA, 1983), pp. 54–57.
3. Barreau, D., and Nardi, B. Finding and reminding: File organization from the desktop. *ACM SIGCHI*

- Bulletin 27*, 3 (1995).
4. Barrett, R., Maglio, P. P., and Kellem, D. C. How to personalize the web. In *Proceedings of CHI '97* (Atlanta, GA, 1997), ACM Press.
 5. Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., and Furnas, G. W. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing* 7, 1 (1996), 3–32.
 6. Berners-Lee, T., Calliau, R., Luotonen, A., Frystyk-Neilsen, H., and Secret, A. The world-wide web. *Communications of the ACM* 37 (1994).
 7. Bush, V. As we may think. *Atlantic Monthly* (1945).
 8. Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. D., and Widom, J. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan* (Tokyo, Japan, 1994), pp. 7–18.
 9. Cockburn, A., and McKenzie, B. 3D or not 3D? evaluating the effect of the third dimension in a document management system. In *Proceedings of CHI 2001* (Seattle, WA, 2001), ACM Press, pp. 434–441.
 10. Cypher, A. The structure of users' activities. In *User centered system design*, D. A. Norman and S. W. Draper, Eds. LEA, Hillsdale, NJ, 1986, pp. 243–263.
 11. Dey, A. K., Abowd, G. D., and Salber, D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction* 16 (2001).
 12. Dourish, P., Edwards, W. K., LaMarca, A., and Salisbury, M. Presto: an experimental architecture for fluid interactive document spaces. *ACM Transactions on Computer-Human Interaction* 6, 2 (1999), 133–161.
 13. Dourish, P., Edwards, W. K., LaMarca, A., and Salisbury, M. Using properties for uniform interaction in the presto document system. In *Proceedings of UIST '99* (1999), pp. 55–64.
 14. Englebart, J., and English, A. A research center for augmenting human intellect. In *Proceedings of the Fall Joint Conference* (1968).
 15. Ericsson, A. K., and Polson, P. G. An experimental analysis of a memory skill. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 14 (1988), 305–316.
 16. Fertig, S., and Freeman, E. Lifestreams: Organizing your electronic life. In *AAAI Fall Symposium, AI Applications in Knowledge Navigation and Retrieval* (Cambridge, MA, 1995).
 17. Fertig, S., Freeman, E., and Gelernter, D. Finding and reminding' reconsidered. *ACM SIGCHI Bulletin* 28 (1996).
 18. Fertig, S., Freeman, E., and Gelertner, D. Lifestreams: And alternative to the desktop metaphor. In *Conference Companion to CHI '96* (Vancouver, BC, 1996), ACM Press, pp. 410–411.
 19. Gifford, D. K., Jouvelot, P., Sheldon, M. A., and O'Toole, J. W. Semantic file systems. In *13th ACM Symposium on Operating Systems Principles* (1991).
 20. Goodman, D. *The Complete HyperCard Handbook*. Bantam Books, New York, 1987.
 21. IBM. Web Intermediaries. Available as <http://www.almaden.ibm.com/cs/wbi/>.
 22. Jones, W. P., and Dumis, S. T. The spatial metaphor for user interfaces: Experimental tests of reference by location versus reference by name. *ACM Transactions on Office Information Systems* 4, 1 (1986), 42–63.
 23. Lamming, M., and Flynn, M. Forget-me-not: Intimate computing in support of human memory. In *Proceedings of FRIEND21* (1994).
 24. Maglio, P. P., and Barrett, R. How to build modeling agents to support web searchers. In *Proceedings of the Sixth International Conference on User Modeling* (Sardinia, Italy, 1997), Springer Wien.
 25. Malone, T. W. How do people organize their desks? implications for the design of office automation systems. *ACM Transactions on Office Information Systems* 1, 1 (1983), 99–112.
 26. Neisser, U. *Memory observed*. Freeman, San Francisco, 1982.
 27. Rekimoto, J. Time-machine computing: A time-centric approach for the information environment. In *Proceedings of UIST '99* (1999), pp. 45–54.
 28. Rhodes, B. J. The wearable remembrance agent: A system for augmented memory. In *Proceedings of The First International Symposium on Wearable Computers, ISWC '97* (Cambridge, Mass., USA, 1997), pp. 123–128.
 29. Rhodes, B. J., and Starner, T. Remembrance Agent: A continuously running automated information retrieval system. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi Agent Technology* (1996), pp. 487–495.
 30. Robertson, G., Czerwinski, M., Larson, K., Robbins, D. C., Thiel, D., and van Dantzich, M. Data mountain: Using spatial memory for document management. In *Proceedings of UIST '98* (San Francisco, CA, 1998), ACM Press, pp. 153–162.
 31. Walker, J. H. Document examiner: Delivery interface for hypertext documents. In *Proceedings of Hypertext '87* (Chapel Hill, NC, 1987), pp. 307–323.