GNOME Storage: filesystem meets interaction design

Seth Nickell (snickell@redhat.com)

ABSTRACT

Despite its shaping influence on the interactive structures of desktop computing, human-centered design has yet to be seriously applied to the filesystem. GNOME Storage is an evolving design and implementation for a document store that addresses many problems stemming from traditional filesystems. Some of the problems are: document folders are overflowing with hundreds of files, there's a strong separation on desktop computers between desktop and application, there's a strong separation between local and remote resources, most applications do not support real-time collaboration using multiple computers, localizing basic filesystem structures is very hard, and applications use a save-buffer model rather than presenting documents as concrete objects. We discuss these as design goals and how they are solved in the Storage implementation, in addition to providing a theoretical framework for understanding the filesystem's role.

INTRODUCTION

The filesystem serves as the backbone of the desktop model and metaphor. It has played a critical role in desktop interfaces since the Xerox Star, and in command line interfaces long before that. Yet despite its importance to the interactive structures of the desktop, filesystem design has centered around supporting the technical needs of the operating system. Even on relatively human-centered systems such as the Apple Macintosh, the primary influence of design on the filesystem has been on the layout of the filesystem (and the minor concession of “long filenames”).

A number of problems have appeared that we attribute to this lack of human centered filesystem design. Some of the problems are: document folders are overflowing with hundreds of files, there's a strong separation on desktop computers between desktop and application, there's a strong separation between local and remote resources, most applications do not support real-time collaboration using multiple computers, localizing basic filesystem structures is very hard, and applications use a save-buffer model rather than presenting documents as concrete objects.

GNOME Storage represents the application of interaction design to the filesystem: in particular to the filesystem as a document store. We believe the conventional filesystem is an excellent design for low-level operating access and do not intend to supplant this. Instead, we provide a new layer intended soley to satisfy human interfaces.

We have identified three major roles that we believe a document store should fufill in order to provide a good desktop interface:

1. Orthogonal Persistence - Storage stores objects rather than "black box files". All information is equal (no metadata), and is accessible by association. While this is partly a technical detail, we believe it has a significant impact on the interface.
2. Object Reference - Storage relies on association through natural language phrases for object reference rather than the mix of recall and recognition used in hierarchical file browsing. As this is probably the most contentious aspect of Storage, we focus most of the paper on associative access for object reference.
3. Mediator (between different threads, computers, and people) - Storage is fully network transparent and provides mechanisms for multiple people and processes to simultaneously access documents.

We believe the Storage design solves the problems listed above by addressing deficiencies in the three major roles of the document store found in conventional filesystems.

ORTHOGONAL PERSISTENCE ROLE

Orthogonal persistence refers to objects (or systems of objects) that persist until such time as they are no longer accessible or relevant. Providing orthogonal persistence to applications and the operating system is often

considered the “primary” role of the filesystem.

Whereas most applications working with documents treat them (at least to some degree) as objects, conventional filesystems store information as an untyped stream of bytes. From a high level view, applications serialize the constituent objects of a document into a stream of bytes and write this out to disk. The resulting file is essentially a black box: readable only by applications with custom logic for interpreting the stream of bytes. While there are many common properties to document formats, every application has, in the past, solved problems for themselves, thus burying the common properties in an uninterpretable “block box” file. Extensible Markup Language (XML) and other structured formats represent an attempt to take many of these common properties. In the case of XML the common properties of “objects” with parent-child relationships and attributes were encoded. Additionally, XML schemas provided a standard for a more refined type definition, further decreasing the “black box” aspect of files. Finally, XML namespaces, combined with schemas, allowed XML “objects” to provide multiple inheritance: a single file could satisfy a number of different schemas / types.

But even with XML, the interface to the files still occur through a semantically impoverished stream interface. Storage provides objects to the GNOME desktop that behave like normal objects: you can set attributes, access children, etc. Additionally, Storage objects provide notification when attributes are changed. Thus multiple applications, or even computers, can access the same object at once operating as a model-view-control (MVC) system. Storage requires these technical features to support its interface goals, but the importance of providing good orthogonal persistence extends beyond “implementation details”.

In most development environments programmers implicitly perform much of design work for applications. There are often too few designers, and they often do not have the influence needed to control all interface aspects of the final artifact. As a result programmers inject their own conceptual models about the system into the interfaces they end up designing. Thus, when possible, it is beneficial to provide programmers with a design model that is similar (if more sophisticated) than the conceptual model you want imparted to users. In the non-ideal case of programmers doing design, we think of programmers as a medium through which the filesystem designer's design model will be filtered.

MEDIATOR ROLE

The filesystem has always served as a mediator between desktop/operating system and applications. While systems such as Microsoft Windows have largely given up on using only the filesystem for this role, also including a “Start” menu to launch applications, the core desktop interface still relies on the filesystem to launch applications relevant to documents. However, this is a very limited mediation role. We expand the role of the filesystem in mediation extensively.

Mediation between Small Tools

Objects in the store can have multiple types. For example, a paragraph in a word word processor is both “text” and “an alignable object”. Current functional desktops are constructed out of “factories”: large applications which are largely autonomous and provide large swaths of functionality for a few specific types of object. This differs substantially from the comparatively rich interfaces we find in real-world interaction where small “tools” can be used on objects depending on their properties. While command line systems such as Unix have succeeded in producing small sets of tools which can be used together in a multitude of ways, this requires substantial learning and does not translate well to graphical interfaces.

We do not propose a small tools interface for the desktop within the context of Storage. However, given the many possible benefits of a small tools based interface when compared to the dominant factory interface, we think it is important to remove as many barriers to a small tools interface as possible. Thus Storage objects can implement (and define) a variety of interfaces. Additionally, as mentioned previously, multiple processes can access objects simultaneously. Thus the stage is set for building an environment focused around tools rather than factories: Multiple tools can be simultaneously applied to objects w ith plastic types.

Collaboration

With traditional filesystems, applications that wish to include collaborative aspects must handle this themselves. Storage integrates the idea of collaboration into the document store itself. Objects, the constituent parts of documents, can be accessed simultaneously by multiple applications, even those running on other computers (Storage itself is network transparent). Thus storage must handle the “meta-design” for supporting good collaborative interfaces in applications that will be designed atop storage.

When other users work on a document, the interface in the application will be responsible for highlighting the region the other user is working on, but should not “lock” the region unless serious loss of data is likely to occur. It should then allow natural social “locking” of documents to occur by presenting methods for contacting the other user, perhaps represented as a little icon next to the aspect they are working on. Because applications operating as views over the model of the document store, changes made by the other application will be updated in the document in real time.

To provide a concrete example, a text editor or word processor might highlight “other user's work” at the paragraph level. It could show dotted lines to indicate the current viewable region of other users working on the document. Paragraphs somebody else is working on would be highlighted in a light colour, and an icon indicating that user would be to the right of the paragraph in the margin. Clicking on the icon would pop down a menu

Storage supports such application interface by providing transaction support, live feedback as document objects change, and API for getting information on other users accessing particular objects or nodes. Eventually we plan to have integration with network naming and information services such as LDAP.

OBJECT REFERENCE ROLE

Object reference refers to the process by which specific "objects" are denoted, typically within human conversation. The task of opening documents on a desktop computer system is a special case: the user is attempting to communicate to the computer the object they wish to work on. The dominant interface for object reference in current desktop systems is the heirarchical filesystem browser (e.g. MacOS Finder or Windows Explorer). These operate by allowing the user to constrain the set of denoted objects with each sub-folder selected, until the user chooses a specific document. Because documents can only exist in a single filesystem path[1] a heirarchical filesystem interface provides monotonic constraint by a single attribute (with each "folder" serving as a constraining attribute), resulting in a unique file.

Whereas database systems allowing constraint of a rich set of attributes have existed for a long time, early computer systems were too slow to use these as a reasonable mechanism for the Operating System to access files (human considerations were much less important when the computer cost more than its operator's salaries). Furthermore, the guaruntee of a unique file given perfect memory of a set of linear constraints (a filesystem path) was attractive for a computer's own internal use (because computers have very accurate memories, but have trouble dealing with uncertainty). While the basis of the heirarchical filesystem browser interface is historical, it has become entrenched as the dominant paradigm for object reference in computers. Unfortunately the number of documents stored in people's computers continues to increase as computers are used in more places for more things. We believe the heirarchical filesystem no longer adequately addressing the needs of object reference in human computer interaction.

The primary alternative to heirarchical object reference systems in use today is a "search" method. Search, however apt a technical description of what the computer does, is not an accurate description of object reference from a human view. While one might argue that when attributes are constrained people "search" the space of objects they know about to find an object matching the constraint, this is a computer science reduction of human cognition. We propose that object reference interfaces should be modelled after a similar (and implementationally identical) idea: that of

[1] Actually, this is not quite true. "Shortcuts" and "aliases" do allow a document to exist in multiple filesystem paths, but this feature does not characterize the normal mode of filesystem usage.

referencing a known object from a known space. "John, get my stapler" is a fundamentally different query from "John, find a paper on mountain chimps". The former is object reference, the latter is "search".

As an alternative, we propose an interface and set of techniques based on an extension of the "search interface", but designed around the idea of association. We believe this system provides a basis for object reference that:

- Is more useful with large numbers of documents
- Is faster for people to use
- Requires less cognitive load, particularly on the human memory
- Closely mirrors object reference in human conversations

Some of the techniques proposed for the interface are technically difficult and require reimplimentation of basic system constructions, but they are by no means impossible. We have written a prototype called Storage that implements most of the techniques described in this paper. Where a technique has not been implemented in our prototype, we mention it explicitly.

The Storage Object Reference Interface

A brief overview of Storage is useful as a reference for the rest of the paper. Storage's interface is a simple "search" box which allows input of phrases in human language (currently English, but the system is extensible to other languages). Search results are filtered in realtime as additional words are typed, providing fast refinement of search queries and easy recovery when an object reference is over constrained.

1. Search results update as you type. This allows quick exploration, which is especially important given the natrual "walk up use" problem for a search interface[2].
2. Movies, images, and PDFs (more in the future) are shown in a small preview form to allow for a visual search (people are much faster at visual search than reading), and to provide increased information scent. Relevant attributes of the document in question are also provided in the search results.

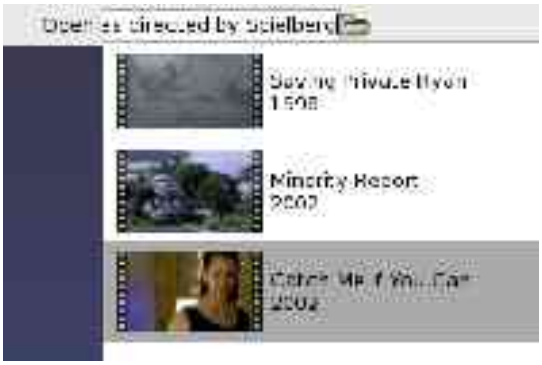

Figure 1Access to "movies directed by Spielberg"

3. Searches can be dragged to the desktop as folders. This allows for some of the recognition advantages of folders, but not inside a fixed single category heirarchy. A search folder about "Music by U2" will always contain all the music by U2 stored on the computer. This ability allows "experts" to optimize their behavior on common (for them) searches so they do not have to type them frequently.

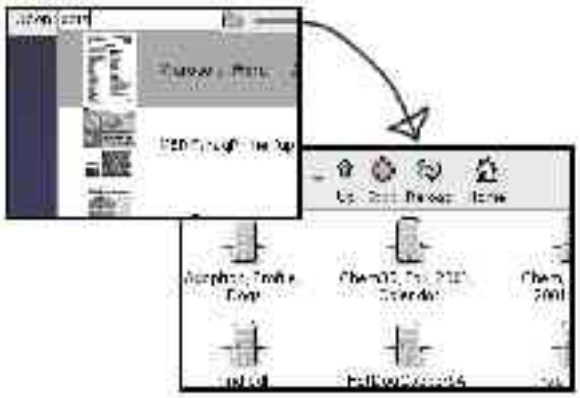

Figure 2A search for "PDFs" can be dragged to the desktop

[2] al systems are harder to find specific items in, they provide more information scent when learning what files are available on a system

We plan to compute categorizations when the number of results exceeds 10 or so, and display those to allow for a combination of information scent and association when searching. Unfortunately there was not time to implement this feature. This may remove the problem of exploration (as compared to the heirarchical filesystem) since a search with no constraints will produce a high level categorization of the documents on the computer.

PRIOR WORK

Interest in alternatives to the traditional file system interface seem to have died out in the late 1980's, possibly as a consequence of the seeming impossibility of shifting to anything else. Even in this period there is a surprisingly small amount of published work. Systems like the "Semantic Filesystem" (Gifford et al) were aimed at augmenting not just the human interface, but the entire filesystem interface. For example, in O'Toole it is proposed that a semantic filesystem be constructed that has no "where" based naming but instead relies soley on file attributes and properties. This proposal is for the basic filesystem, not an extra layer. It is not surprising that these proposals never caught on, because there performance was substantially lower than "where" based naming (on a period Microvax), and because the filesystem was shared by the Operating System where high performance was desirable. While the "information store" component of Storage is amazingly similar to the semantic filesystem explored by Gifford and O'Toole[3], Storage has four critical differences:

1. Storage does not aim to replace the filesystem for the Operating System usage it was originally designed for (and works well for).
2. Storage's search interface is based on Natural Language rather than requiring learning of a query language
3. Storage does not merely index information, it stores it in a fundamentally decomposed format. This allows the entire body of semantic information in a file to be exposed, whereas the SFS was clearly designed to expose a small subset of information in the file (perhaps a dozen attributes on each file)
4. While Storage does provide a compatibility virtual filesystem layer, it also provides a clean direct API with major benefits for application developers.

A more modern system along similar lines is the Haystack environment from MIT (best exemplified for our purposes in Adar 1998). Adar, like Storage, distinguishes between the object reference task (what she describes as accessing personal files, books, etc) and the search task (accessing information from a large corpus where there is no knowledge of specific items), and like Storage chooses to focus on object reference. Adar proposes that the distinction between the two is that of structured search and unstructured search, but that both are present to varying degrees in most queries. To this end, Haystack provides multiple "search" means, akin to a full-text search (like Google) and an associative property search (like the SFS). While Adar's paper uses natural language queries to illustrate the problems of structured (e.g. Storage) v. unstructured search (e.g. Google), it appears that their intention for an actual implemention (and what is seen in Haystack) is a sort of unstructured query language combined with data mining.

On a more theoretical note, Fisher et al focus on the problem of searching through poorly structured "information spaces" using Usenet archives. Their chart on "information lifecycle" details errors and failures people experience while trying to access information. It provides an interesting (and richer) alternative to the interface / computer knowledge distinction discussed later in this paper, though much of the chart seems related primarily to their domain. Like Storage, Fisher et al underscore the importance of "real time" feedback as to a query / operation's success in order to allow iterative refinement and reformulation. In the end, they settle on a combination of a structured browser (not surprising since Usenet is a space where search is "unstructured" rather than "structured, using Adar's terminology) and an interactive query building and refinement tool.

[3] The SFS and Storage's information store are similar right down to, amazingly enough, using a virtual filesystem to provide backward compatibility with existing applications. History does indeed repeat itself.

If it is possible, still less seems to have published addressing the issues of natural language queries from an HCI perspective. There is a great body of literature on both natural language parsing, and even a substantial "SQL NL query parsing" body of work, but both provide primarily techniques for improving parsing, not discussions of issues and interface. One paper that was somewhat helpful in developing ideas about feedback and error recovery in natural language systems, though primarily inspired by their discussion rather than directly addressed, was Tennant et al (1983). However, the paper is sufficiently dated that linguistic parsing has improved so substantially that many things discussed are non-issues. I would suggest that part of the reason for a lack of work in this area is that most natural language interfaces are sufficiently poor in understanding human language that they are not of interest to HCI professionals[4]. We are leveraging very recent work in HPSG grammars (http://hpsg.stanford.edu), combined with the insight that noun phrases with constrained verb heads present substantially less ambiguity than open ended parsing to achieve reasonable linguistic results.

The Heirarchical Filesystem Has Already Failed

A primary goal of the computer's object reference system is to allow people to quickly access their information. Unfortunately, most people have too many items in their "Documents" folder to be able to quickly access their information. In a survey of 24 adult's computers we found that 19 of them had primary document folders with more than 150 files. This is well above the number of textual items that can be searched in a single visual survey, and icons are only of limited use because there are typically only a few types of documents in the folder. In problems of visual search, even when all the information is presented on the screen at once (not possible in this case), and the items are more visually distinct, visual search time tends to increase linearly with the number of items.

The problem is that, in addition to creating their information, the traditional filesystem requires the user to perform the extra task of structuring their information if they wish to realize the benefits of categorization[5]. Common modes of use are to either organize many files at once on a periodic basis, or to organize individual files immediately upon creation. Periodic mass-organization tends to produce a situation where people keep pushing off organization, so more cluttered documents get added, so the problem becomes larger, etc. The result is that the person's most relevant documents (those which they have worked on most recently) are mired in a cluttered mess save a couple weeks after a semi-annual "spring cleaning". Organization upon creation is much more effective, but is infrequently practiced[6].

The "paradox" of the active user might be generalized as a tendency of people to focus on short rather than long term goals. In the context of learning this is manifest in a desire to move on from reading documentation, presuming this actually improves their performance in the long run, to actually doing. But the problem is more general than presented in Carrol & Rosson: any situation where people choose to cut corners (with negative long term implications) in order to more quickly realize their short term goals might be characterized as involving the paradox of the active user. It is precisely this tendency which causes the traditional heirarchical filesystem to fail so stupendously. The "organize my files" goal never takes precedence over the more highly valued immediate goals (usually) involved in using a computer: to get something done. This in spite of the fact that organizing files frequently would probably improve overall production. Consequently, unmanagable numbers of documents acrue in people's primary computer work space.

[4] And on the more theoretical side there is little discussion, I suspect, because it is so obvious that a natural language interface is desirable, and will be heavily explored once they are technically feasible.

[5] For their documents, system files having of course been categorized by a software developer prior to receiving the computer.

[6] Even when practiced, the task is not easy. There is a reason "Information Architects" are hired to organize menus, web sites, etc. Organization by a single heirarchy is a very difficult problem because objects frequently cut across attribute classes.

Recall, Recognition, Association

"Search" operations are usually given a second class status in computer interfaces. They are presented as a fallback mechanism when the user has failed to locate a file using some sort of structured format. This can be seen, for example, in the three clicks required to search for files in Windows (or the Macintosh) compared with single clicks to browse "My Documents", and even more prominently in the amount of visibility given to the respective features (search features tend to be buried). Users respond to this and form conceptual models that mirror the system image simply because the overall interface more readily affords browsing. Users begin to conceive of "search" as a strategy to be employed only when other strategies have failed to achieve the goal of locating a file[7].

It is not uncommon to encounter a cognitive justification of the heirarchical filesystem. The argument goes that search operations require recall, because you are not presented with a list of choices to choose from you must remember attributes of the file, whereas "file browser" interfaces require recognition. Because it is a well established cognitive phenomenuum that people can recognize a greater number of items far faster than they can recall the same, it is presumed that people will be faster and will be able to handle a larger number of files with a file browser interface. I suspect that these arguments are spuriously transferred from a context where they are valid: justifying that the file browser interface requires a lower cognitive load than a command line interface. Why are they not valid in relation to search? Firstly, because search is more accurately characterized as an associative task not a recall task. Secondly, because heirarchical browsing is not a purely "recognition" based task and in fact requires substantial (and difficult) recall.

In a typical command line interface people must memorize specific keys (paths) in order to access the information they want. These keys will, in good cases, have some relation to the desired object ("/doc" for example has some relation to "the 3rd quarter financials spreadsheet"), but the specifics of the key must be accurately recalled. This is a slow and error-prone task. At the other extreme, if you allow people to denote an object by any attribute they find relevant people are extremely efficient (more so than even recognition). This is because in order to have a goal of "open file" in mind the person has some characterization of the file in their head. This characterization can search as the key in an association process. Transferring this abstract characterization into a linguistic form that can be expressed to the computer or another person can take some work, but is generally fairly easy.

Associative reference is the reference system naturally employed in human conversation, and is extremely fast and easy[8]. "My roommate's desk" is an associative reference to the object that might be heirarchically denoted as "Earth.US.CA.Stanford.Kairos House.Room 109.Inner Room.Desk". Notice the higher information density of association as compared to heirarchy as well as how much easier the former is to generate. Association is a technique that scales much better to "real world" situations in which an uncountable (in the literal sense, not mathematical) number of objects and categorizations are available. In these systems the branching factor of a single attribute constraint is sufficiently low that the number of attribute constraint operations required to produce a unique result balloons. The depth of the heirarchical structure is directly related to how difficult it is to navigate (how much recall is required), as will be shown below. Consequently, a heirarchical structure becomes comparatively less desirable as the number of objects in the total space increases.

Finally, heirarchical filesystems require substantial recall tasks in order to perform. A

---

[7] People having acquired this "knowledge" actually presents a serious problem of negative transfer for projects where a good search interface is presented. People have learned that searching for files is a fallback operation and will not perform it even when it is pushed as the preferred interface (and is more effective). We have had to avoid the use of the word "search" in our interface (we call it "open", which is perhaps more appropriate since it is a reference system not a search system anyway) in order to minimize this transfer.

[8] Associative recall so basic that some neurological theories such as spreading activation explain memory as a fundamentally associative process wherein groups of neurons are related to properties, ideas, and concepts.

sketchy walkthrough of the cognitive process involved in acquiring a hypothetical object illustrate the problem well: The goal is to find the spreadsheet containing "my tax information". We are presented with a typical Windows desktop interface. First we must recall (easily) that this is a document and can be found in the documents folderand must scan the desktop to find this folder (could be easy or hard depending on the number of items). Probably we execute this procedure without accessing the declarative knowledge that it is a document (it is simply part of the procedure that we open the documents folder). Opening the folder we find 40 items inside. Are we presented with a recall or a recognition task here? If we were merely asked to report whether we had seen a folder before (as most experiments on recall v. recognition do), it would be a recognition task. But we must remember whichfolder the document was in, which may be a particularly difficult task if it could be in multiple folders (for example if we see both "Spreadsheet" and "Financials" folders it is not immediately clear which is correct). This is hard and requires recall of a specific piece of information. Unlike the command line you do not need to memorize the exact information, you can recognize it from a visual search, but you still must recall the information. Increasing levels of heirarchy make the task substantially more difficult, requiring ever increasing amounts of recall.

In summary: search, in terms of unconstrained association, presents a lower cognition burden than filesystem heirarchy which requires both recognition and recall. In addition, association more readily scales to denoting a large number of objects. Why then are association/search interfaces not prevalant?

How does the computer understand association?

The problem is that we have specified unconstrained association. This is clearly not reasonable. For unconstrained association to be maintained, for example, I should be able to walk up to a new computer and ask for "my favorite file on this computer". The computer obviously cannot know what my favorite file is because it knows nothing about me. The problem is that the computer's knowledge base, reasoning, and interpretation "skills" are limited. There are two important facets to the problem: providing a good interface for inputing constraints, and providing an implementation that has access to information about a large number of associations. Can the set of associations an interface + implementation provide for sufficiently unconstrained association as to perform better than a heirarchical system[9]?

Traditional solutions address the "interface" problem by constraining input. shows a fairly typical interface of this variety, searching for something akin to "Files edited yesterday". Searching for information other than the filename (which is typically not a very friendly string, even when people name documents themselves), requires telling the computer what type of constraint you will be adding, and then adding the specific information to constrain by. This requires alternating between the keyboard and mouse, searching through lists of constraint types, and learning how to work within the set of constraints provided to achieve flexible searches. For example, "files edited yesterday" as shown in required two clicks, searching through a list of 15+ items, another click switching to the keyboard and typing, and then switching back to the mouse to click "Find". Also, only a few attributes such as date and size can be searched over (the common sets of search constraints primarily having been determined by what information operating systems 20 years ago required). This is a problem of providing the computer a sufficient number of associations.

A good alternative is to provide a purely typing based interface. There are two basic approaches: to try and understand human language phrases or to provide a standard "search query format". The latter removes the problem of interpretation, but by shifting a learning burden onto the user. The experimental mail client available from Bloomba (http://www.bloomba.com) is one

[9] The heirarchical filesystem is, in a sense, single constraint association. But this makes it easier to understand and control as a tool, an aspect I suspect would make somebody like Nielsen favour it. The worst system is one that has a small set of invisible constraints (and consequently is neither expressive nor controllable). This danger is exposed by moving away from a heirarchical filesystem and must be carefully avoided.

example of such an interface. Once people have learned the query format they can quickly and accurately (if not always easily until they are fluent with the query syntax, and even then new types of searches may cause problems) type expressive associations. The problem of the active user once again rears its ugly head: people do not want to spend the time to learn a search query format even if it is helpful, and will bottom out on the learning curve once they have learned/found a small set of search constraints that perform just well enough to get something done (in a sense, this is satisficing in the learning context, learning just well enough to meet a basic threshold). This can be seen in even the most basic search query formats, such as the logical operators provided by some search engines. Most people do not use them, even when search engines still promoted them[10].

For these reasons, we chose to provide a natural language interface. How do you make the limitations of the natural language search visible in this context (since this is not unconstrained association)? We believe the key is good feedback on interpretation of phrases (by retranslating the interpretation back to human language) and which parts of the phrase were not understood (by underlining in red words/constructs that were not understood). These features are not yet visible in the Storage interface due to time constraints, but the underlying architecture for them is in place. There are general patterns to the sort of phrases the computer is good at recognizing and those it is not. We hope that this feedback will increase the length of the learning curve before people hit an asymptote, because they will be sub-consciously learning to speak the dialect of English most intelligable to the natural language parser (for example, it does much better with more formal written English than with slang, tends to work better with "movies that were directed by..." than "movies directed by...", doesn't work at all with sarcasm, etc, etc). Of course, it goes without saying that we also focused intently on making the natural language parser handle as much and as complex natural phrases as possible.

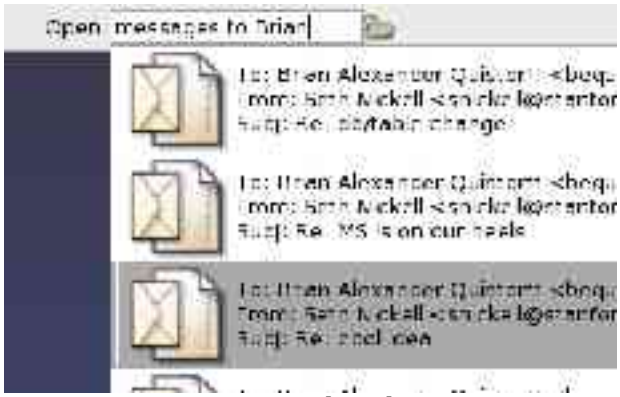

Figure 3Access to e-mail in the store

In order to handle the large number of types of constraints people may want to search for, we have situated the search interface within a rich "information store", where objects and documents are represented in a format that is transparent to the operating system and desktop (it is akin to an XML store). When files are brought into the store, they are decomposed into a structured format, and large amounts of human-sensible metadata about them is extracted, using auxilary information sources when available. For example, importing a movie into the information store breaks the movie's internal metadata apart to determine the date, author, and title, as well as the type, length, width, height, etc. This information is then used to leverage additional information from the Internet Movie Database (imdb.org) about the director, actors, etc. This sort of information is critical to allowing people to express their goals as associations rather than performing a "recall" task, because it increases the space of associations the computer can operate in[11].

CONCLUSION

Storage provides an attractive alternative to heirarchical filesystem interfaces for object reference that is faster to use, scales to large information stores better, and requires lower cogntitive load to use. While Storage's interface is similar to search, it is focused on the task of object reference which produces important

[10] Of course, techical people do often use these search features, often to great effect, but they can transfer a great deal of knowledge from programming and logic domains to make the learning process nearly non-existant

[11] There are many other cognitive and design issues relevant to the information store itself, such as allowing for direct editng of files in realtime rather than buffering, but we will not go into them here

interface differences. Unfortunateley, this results in Storage being weak in the areas of exploration of "unknown" information (a major problem for walk-up use). Planned agumentation with a category refinement system may produce a hybrid that addresses both object reference and exploration.

REFERENCES

1. Eytan Adar. Hybrid-search and Storage of Semi-strucutred Information. Master's thesis, MIT, Department of Electrical Engineering and Computer Science, May 1998.
2. Haystack Project Home. Website, http://haystack.lcs.mit.edu/, accessed May 2003.
3. Fischer G, and Stevens C. Information Acess in Complex Poorly Structured Information Spaces. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1991).
4. Gifford D, Jouvelot P, Sheldon M, and O'Toole J. Semantic File Systems. In Proceedings of the Symposium on Operating Systems Principles, October 1991.
5. O'Toole J, Gifford D. Names should mean What, not Where. Accessed from http://citeseer.nj.nec.com/540075.html.
6. Tenant H, H.R, Ross M, and Thompson C. Usable Natural Language Interfaces through Menu-Based Natural Language Understanding. In Proceedsings of the ACM Conference on Human Factors in Computing Systems, pages 154-160. 1983.