Improving the Usability of the Hierarchical File System

GARY MARSDEN
Dept. of Computer Science, University of Cape Town, South Africa

and

DAVID E. CAIRNS
Dept. of Computer Science, University of Stirling, Stirling, Scotland, UK

Whether you are interested in improving the usability of Linux, Macintosh or Windows, there is one restriction you cannot escape – the hierarchical file storage system. The notion of files and folders has been with us for so long that it almost seems axiomatic. In this paper we look at the effects on users of forcing a hierarchical classification of files. We also consider how some of the resultant problems can be tackled with a new piece of file browsing software based on the ideas of relational database systems.

Categories and Subject Descriptors: H.1.2, H.2.0
General Terms: Design, Human Factors
Additional Key Words and Phrases: Human-computer interaction, databases, information retrieval, file storage, searching.

1. INTRODUCTION

Computer scientists like hierarchies. One has only to look at the proliferation of tree data structures [Cormen 2001] or immense class structure diagrams [Java 2003] to see that complex hierarchies do not intimidate the average computer scientist. Even the way files are arranged on a hard disk is a tree with branches (folders) and leaves (files). This scheme has been adopted by operating systems since the 1970s and is still with us. There have been two major changes since 1970, however:

1. The average disk space available to the user has recently been doubling every year – faster than Moore's law predicts [McCarthy 2002]. This means that hard disks now contain vast trees which require regular pruning and tending in order to stop them obscuring the data they contain. Remembering a file's position in a storage hierarchy of a 120 Gb disk would tax even the most organised of minds.
2. Computers are now used by people who are not familiar with hierarchical file systems and who have no idea how to manage large collections of data.

Whilst most advanced computer users have occasionally had to struggle through the hierarchy on their hard disk, this inconvenience does not seem to warrant redesigning how files and folders are stored and retrieved. However, anyone who has studied how application users store files realises that the file system is quite a large barrier to all but the most advanced users. The interface designers for Windows 95 realised that most users did not understand the folder/file hierarchy and those that did had a hard time making it work for their filing needs [Sullivan 1996]. The confusion of the hierarchy sadly did not prompt the Windows 95 team to redesign the file explorer, but instead created work-arounds such as the 'My Documents' folder and other application default folders. Whilst these compromises might work to a limited extent, their behaviour can confuse users – for example, when an application's default folder is changed, users believe that their files contained in the original default folder have been deleted.

Furthermore, it is not at all clear that a strict hierarchical classification of files is always appropriate [Dourish 2000]. Symbolic links, aliases, shortcuts and other patches have been added to most operating systems in an effort to allow users to make the same files visible from multiple places in the file hierarchy. (Of course, these mechanisms work differently in different operating systems). This kludge clearly points to some more systemic problem.

Dourish [2000] identifies the following problems in using a strict hierarchy:

- filing – often documents need to be classified in several different ways

Author Addresses:
G. Marsden, Dept. of Computer Science, University of Cape Town, Private Bag Rondebosch, 7707, South Africa. gaz@acm.org, www.hciguy.com
D. Cairns, Dept. of Computer Science, University of Stirling, Stirling, Scotland, UK, FK9 4LA. dec@cs.stir.ac.uk

- management – folders are used for organising like documents but are also used for system purposes such as sharing

For archiving purposes, however, hierarchies are an ideal mechanism. Whilst archiving is a facility programmers require, studies [Barreau 1995] show that most users rarely archive files. So the task that hierarchical file systems best support is one that most users do not need!

Therefore, we wish to investigate how to create file structures that will better support novice users filing their work on disk.

2. PREVIOUS WORK

One approach to managing data is to replace the hierarchical classification with a temporal classification. Projects such as "Life Streams" [Freeman 1997] and ScopeWare [Scope 2004] use time-based visualisations of files. Documents are shown in a continuous sequence ordered by when they are created. There is much to recommend these systems, but they rely on the user remembering when a document was created. We are seeking a more general system than this, based on no one particular attribute.

Attempts at creating alternative file stores include the 'Semantic Filing System'[Gifford 1991] which replaces the file store with a database of documents from which users can create their own virtual directories. Whilst this can overcome the problems of multiple classifications, it still requires users to have a knowledge of hierarchical directories.

Figure 1 *– Quan's system browsing for the files with attributes Microsoft AND software AND pirating.*

Another alternative for overcoming the problem of hierarchical mis-classification is proposed by Dourish et al [2000]. In their solution, they allow users to add their own attributes to files and create their own folders and hierarchies based on those attributes. Quan et al [2003] have built a similar personal filing system, allowing users to create their own hierarchical file store. This system also allows users to add their own attributes to files, allowing them to be classified in multiple ways. The system then visualises files in a hierarchy, not according to logical organisation on disk, but according to user assigned attributes (see Figure 1). The contents of the final folder thus represent the logical 'ANDing' of all the attributes specified by the folder names in the path to the resultant folder.

The weakness in both these solutions (and others like [Jones 1986]) is that they rely on users taking the time to explicitly assign meta-data to files. Whilst this may work in a corporate environment (which is what Dourish is interested in), or in the forced environment of a laboratory user study (as in Quan's case), observational studies of personal filing show that users are unlikely to use metadata. Even when the tagging is something as convenient as voice tagging, users are not likely to make the time investment required to assign metadata to files [Rodden 2003].

So, whilst it may not be wise to build a system that explicitly requires users to assign metadata, there is much merit in allowing users to view their file store according to some attribute of the files being viewed. For example, studies of email users show that email is often stored in a single list and rearranged on a given attribute for a given task (e.g. sort the list in terms of receive date in order to find all emails received in the previous month) [Whittaker 1996].

Dynamically rearranging documents in this way is not possible within a hierarchical file system – files are always arranged according to the static hierarchy. We therefore need to find a model that gives the user the flexibility to restructure their file store according to the task in hand.

We believe that such a solution is to be found in a branch of computer science dedicated to fast information retrieval – namely, database design.

3. DATABASES

Originally database information was stored in flat files. The flat files could not model the relationships inherent in the information, so the hierarchical database model was created [Date 1993]. Here information was stored in strict

hierarchies allowing the modelling of one-to-many relationships. However, this was not powerful enough to model all situations, so the model was adapted by using 'link' records which created symbolic links across hierarchies.

Effectively, this is the point we are now at with hierarchical file stores; strict hierarchies kludged with link files. However, database design has managed to move beyond this point and hierarchical and network databases were eventually replaced by the much more powerful relational database model.

The power of the relational model lies in its ability to dynamically model the relationships between the data – there are no fixed structures, as with the hierarchical model. Depending on the application at hand, new relationships can be built into the data model and old ones discarded as they become irrelevant.

So should we then ask users to learn SQL, normalisation and store their data in relational databases? Of course not. What we do want, however, is to ape the ability of the relational database to model the data in a way which best suits the task at hand. Therefore, we believe that the operating system file browser be changed to interact with the file store as dynamically as one can interact with data stored in a relational database. It is not important to the user how the files are stored on disk (performance constraints aside) but the user should be free to choose how those files are viewed and retrieved. (BeOS [Giampolo 1998] is an example of an operating system which stores its files one way – in a relational database – but presents them to the user as a normal hierarchical structure).

3.1 Design Inspiration

Our previous work has shown that using computer science algorithms as an interface design guide can result in new forms of intuitive interfaces [Marsden 2002]. We are therefore interested in seeing what functionality from relational databases can be used in the new system.

For our purposes, all files may be viewed as tuples in a single relation with attributes such as 'Name', 'Date created', 'Size' etc. Relational algebra gives us a set of canonical operators which we can use to manipulate relations, namely: restrict, project, product, join, divide, union, intersection and difference. Of these, product, join and divide can be ignored as they deal with data modification – we are concerned only with data manipulation. Also of interest are data manipulation commands from standard SQL not covered by relational algebra. These commands would be 'order-by' and 'group-by'.

To summarise, our system must present files to a user in a way that lets them perform the following operations in the simplest way possible: restrict, project, union, intersection, difference, group-by and order-by. For ideas in how to present these functions, we shall look at previous attempts in file store visualisation research.

4. VISUALISATIONS OF FILE STORES

One way many researchers have sought to improve the hierarchical file store is to improve the way the structure is presented to the user. The value of visualisations in presenting file stores to novice users cannot be over emphasised. Observational studies [Barreau 1995] show that, regardless of operating system, users prefer to search for files visually (often taking longer) than try to recall the filename they allocated in the first place. The result of this behaviour is that the 'Find file' search facility is used only as a last resort.

Text based systems such as FlexView [Schaffer 1993], and graphical visualisations such as treemaps, cone trees and hyperbolic trees [Card 1999] all provide good visualisations of hierarchies, but are built on the premise that the underlying structure is meaningful to the user. As we have discussed already, this is not always the case. Considering systems which visualise relational databases are also problematic, as they tend to be designed for database designers or other domain experts.

Returning to the study of email users [Whittaker 1996], one visualisation of documents which does work well is that found in most email clients – a chronologically ordered list of received messages. By clicking on the different headings (see Figure 2), users can re-order the list dynamically. Although it is hardly an advanced visualisation, it is effective and serves as the starting point for out system.

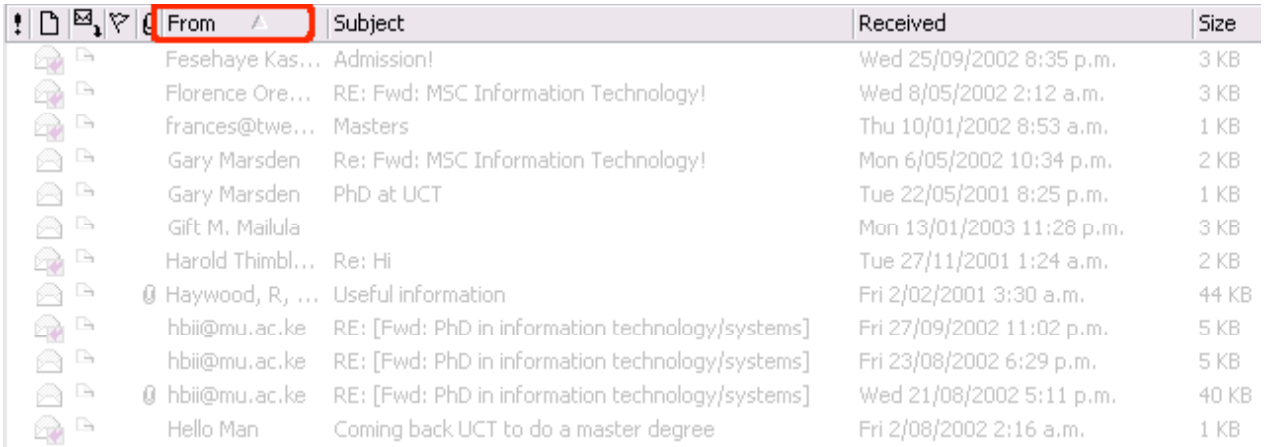

Figure 2 *– Here, messages are arranged in alphabetical order according to who sent them. Clicking on any other heading will re-order email according to the values in that column.*

5. SYSTEM DESIGN

5.1 Lessons from Email

Considering the email client visualisation in relational database terms, it is essentially the visualisation of a relation where each email is a tuple.

If we reflect for a moment on the attributes of the email relation, we see that (just as with other relations) some of the columns contain static attributes (sender) and some contain computed attributes (read/unread).

Using the 'View' menu, users can perform a project operation by specifying the columns to be viewed. The restrict operation is available as spam filters, reducing the set of tuples to those the system considers worthwhile. Clicking on a column heading is a neat way of implementing the 'order-by' function. The other SQL function to consider, 'group-by', is provided in the sense that the user can group items visually once ordered. The union, intersection and difference operators are not provided, presumably because the system designers considered them too complex for users to understand – they represent a higher level functionality, whereby simple query results may be combined to form more complex queries. Translating these ideas to a file browsing system becomes problematic, however, due to the size of the file store.

5.2 Moving to file stores.

Current file browsers support a similar attribute view of files, but only in a per-folder basis. In Figure 3, we can see a Macintosh OS-X Finder window, where files are ordered (ascending and descending) by selecting the desired column heading. The settings dialog for the window also supports project (for column headings) and restrict (e.g. system files can be made invisible). (It also includes some presentation and performance settings which shall be discussed in section 5.4).

If we were to visualise all files in this list, as in the email client, the list would be too long to afford rapid scanning and, amongst other things, we would loose the ability to perform a visual 'group-by'. Turning back to the literature on visualisation, we find a solution to help us deal with this type of information overload. Ben Shneiderman's 'Information Visualization Mantra' is 'Overview, zoom & filter, details-on-demand' [Shneiderman 2003]. Paraphrased, this means that when faced with too much information, we need to provide some overview, which can be zoomed down to the specific details we require. Fortunately, the 'group-by' function was designed for providing overviews of data and shall be used as the starting point in our system's visualisation.

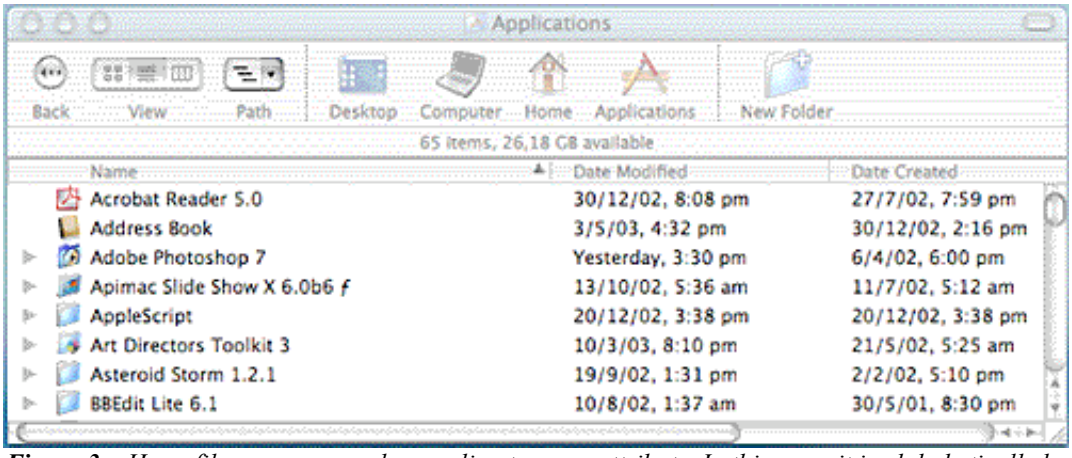

Figure 3 *– Here, files are arranged according to some attribute. In this case, it is alphabetically by name of the file.*

5.3 Typical Queries

Figure 4 shows the visualisation system in action. The top-left pane shows the different ways of visualising the file store – in this case, the 'Accessed', 'Size" and 'Created' are visible, but more options exist beneath. The bottom left pane is a history mechanism showing the different visualisations the user has selected – analogous to a file path in hierarchically based systems. The central pane shows all the files which meet the current selection criteria which, in the initial phase, is every file on the disk. Finally the rightmost pane contains the properties of the currently selected file. (Please note that the current interface is for development and research purposes and is not intended as final, end user system). To

give you some idea of how the system works, imagine searching for a Microsoft Word® document that you edited last week.

Figure 4 *– The initial view of the attribute browser*

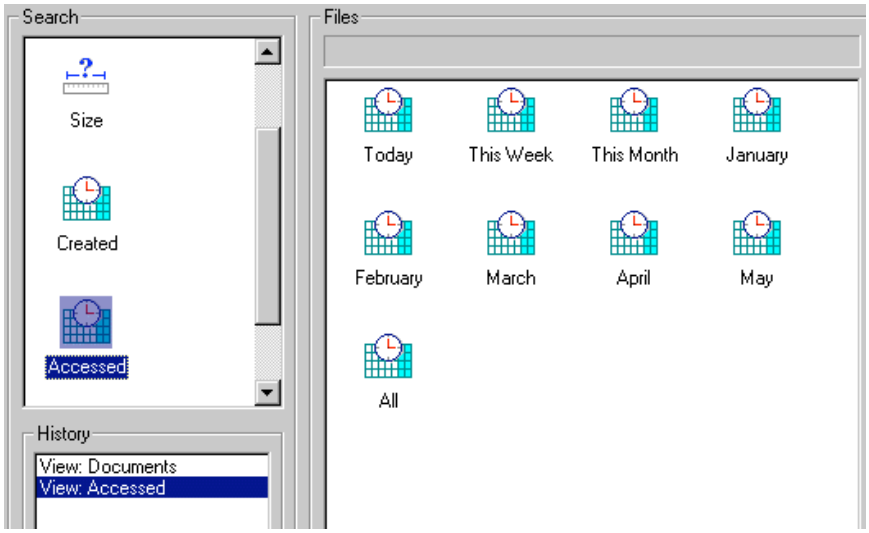

Figure 4a *– selecting files by access date*

Selecting the 'Accessed' attribute shows the time groupings (as in Figure 4a) when the files were last accessed. Then, selecting 'This Week' displays all the files accessed in the last week (Figure 4b). The search is further refined by selecting 'Type' (Figure 4c) to show all the different document types and then selecting 'doc' to reveal that two documents of type 'doc' were accessed in the last week (Figure 4d). In Figure 4d, the properties of the target file which our system can extract are visible in the 'Properties' pane. This example neatly shows the 'overview, zoom and details on demand' mantra in action.

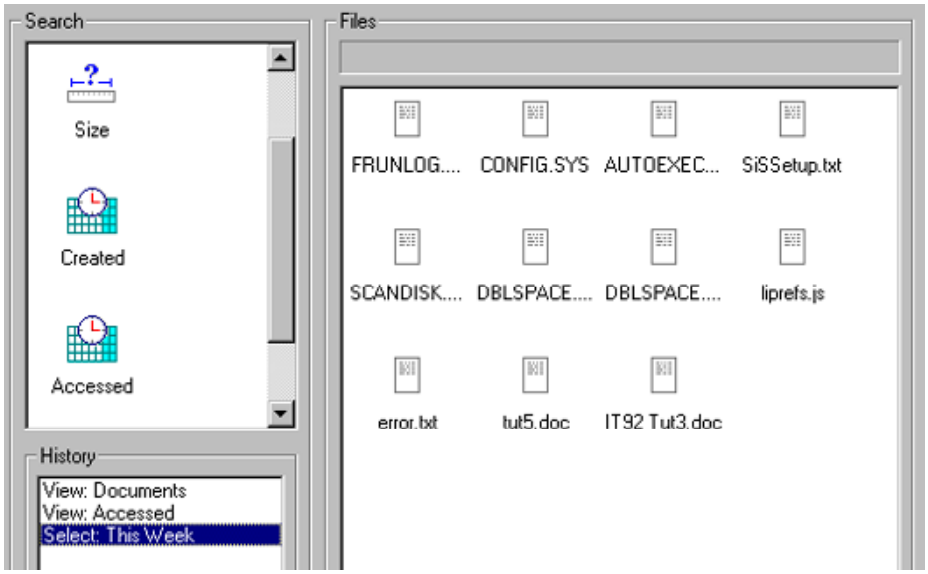

Figure 4b *– showing files accessed in the last week*

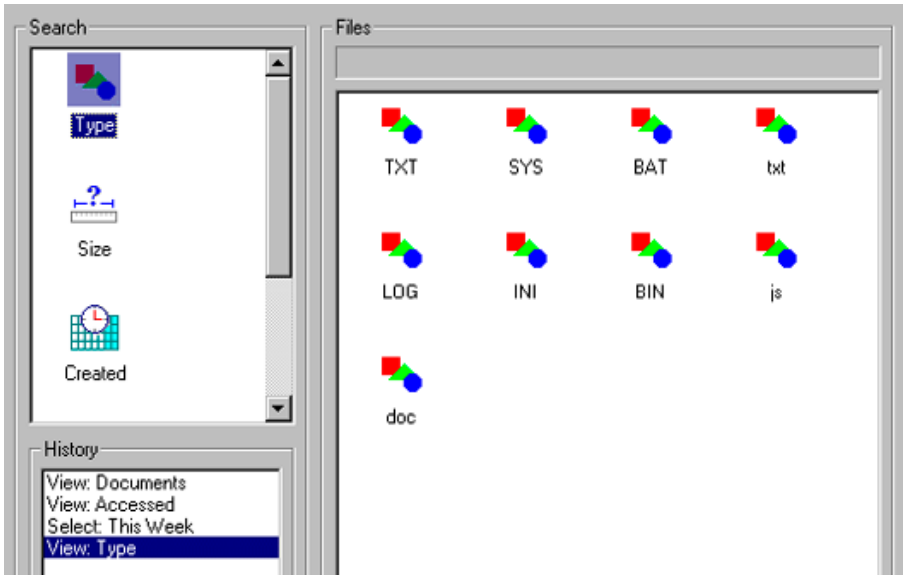

Figure 4c *– showing file types accessed in the last week*

Figure 4*d – Properties of the desired file*

In the above example (using date) there is an implicit 'group-by' hierarchy (year-month-day) which we can use to provide the level of detail. The net result is that users can find a file should they remember when they last worked on it. Alternatively, they can select the 'Created' attributed and see an identical visualisation, but this time based around when they first worked on a file

Besides access date, similar visualisations may be created for:

- Application type – here, the registry information can be used to show items organised by the application that created them and then by the file type
- Creator – in this instance, we can use the group/user hierarchy to create a visualisation around who created the file

At present, the system is implemented on top of Windows 98 and is able to access all attributes available to the host operating system, making possible a wide variety of possible visualisations. It is also worth noting that by considering a file's path as an attribute, it is possible to re-construct the original view of the file store.

In principle this approach could be extended to view all files as objects and then to dynamically scan and index the file/object store not only by object attributes but also by available methods. In addition to filtering by attributes as discussed above, this would enable a user to locate all files that supported a common method such as 'preview', 'embed' or 'print'. This would allow a user to search for a file with a query such as 'are there any image files that I can embed in my current document?' It is envisaged that such a system would dynamically scan and index file store objects with the current default attributes acting as a base line and any new attributes or methods automatically added to the list of searchable fields in the database. The presentation of newly discovered fields (attributes / methods) could then be restricted by how prevalent they were in the file store – if only one or two files supported a particularly obscure method, the interface would quickly become overloaded unless some form of pruning was available.

5.4 Combining Queries

In initial trials, the history pane has proven very flexible. Should a query fail to produce the expected results, a user can easily return to the step where they suspect the search went astray and investigate alternatives from that point. The provision of the history pane, however, requires us to reconsider the role of the 'union', 'intersect', and 'difference' operators. Again, there do exist visualisations of these operators in visual SQL systems, but these are built for users familiar with relational constructs. The closest system to ours is the PDQ (Pruning with Dynamic Queries) system [Card 1999]. However, this system is built on the idea that users are only interested in pruning (i.e. intersection) and not in union or difference. Whilst pruning is in line with the visualisation mantra, we wish to support the full range of relational algebra operations. We have several design alternatives (see Figure 5 for one such suggestion) but need to conduct some observational studies to determine typical user queries and determine if it is worth providing this functionality and run the risk of obfuscating the interface.

Figure 5 *– The upper query shows a union of all the files that were created on the 13th of May and all the files created by Gary. The second query shows a difference operation, resulting in the files created on the 13th of May that were not created by Gary.*

5.5 Presentation properties

The relational model is based on an abstract mathematical theory. By choosing it as a basis for our design, we run the risk of ignoring how presentation is used to aid searching. However, our work is concerned with providing a structure within which files may be organised – how the end result is visualised is not within the remit of our work.

One possible criticism might be that screen space is an effective way to organise files and folders and people search for files based on location. Our reading of the literature shows that there are contradictory findings on this assertion (e.g. [Jones 1986] and [Mander 1992]) so have not used spatial location as an aid in the visualisations (icon locations are dynamically calculated).

6. DISCUSSION

In this paper we set out to provide a more flexible file browsing system for novice users. To do this we used the ideas of relational databases to help understand what features such a system should support. The system was then implemented on a PC to test those features. We are now at a point in our research where we shall have to conduct user testing to determine which of the following features should be added to the system.

6.1 User Defined Meta-data

We have already made that point that any system which relies on user defined meta-data is unlikely to succeed for most users. However, this should not

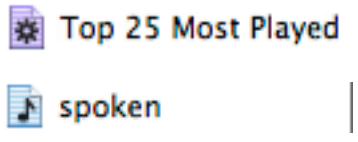

Figure 6 *– iTunes: The upper icon shows a "computed" playlist whilst the lower one represents a list assembled by*

preclude the possibility of allowing users to attach data, should they wish to. In Quan's system [2003], users are limited to attaching Boolean metadata to files (e.g. a document does/does not relate to the attribute 'Work in progress'). We believe that such a scheme is limiting and users should be able to attach metadata of any type to a file.

6.2 Temporary tables

We believe that users will want to save query results to save them reconstructing common queries every time the results are required. To return to the language of relational databases, this would, in effect, be a database cursor. Fortunately much work has been done on this already ([Dourish 2000], [Mander 1992]) and there exist schemes for representing computed and user defined queries (see Figure 6).

7. THE VALUE OF A COHERENT DESIGN IDEA

Part of the reason for conducting this research was to see the effectiveness of creating an interface by following a well formed idea from computer science – in this case, relational databases. Since this work was first published (early 2003) Microsoft released the first version of its new operating system, codename Longhorn [Microsoft 2003]. One of the stated goals of the Longhorn design is to help people manage terabytes of disk space. From an architecture perspective, Longhorn will achieve this by storing files in a relational database. Like our system, the user does not interact directly with the file store via SQL, but rather sees dynamic folders, which can be selected an rearranged according to the task in hand. Below is a screenshot (Figure 7) whereby the files on a disk are organised by author. Note that different icons are used to show the relative size of the folders. This view was created by selecting the "All Authors" tag at the left of the screen.

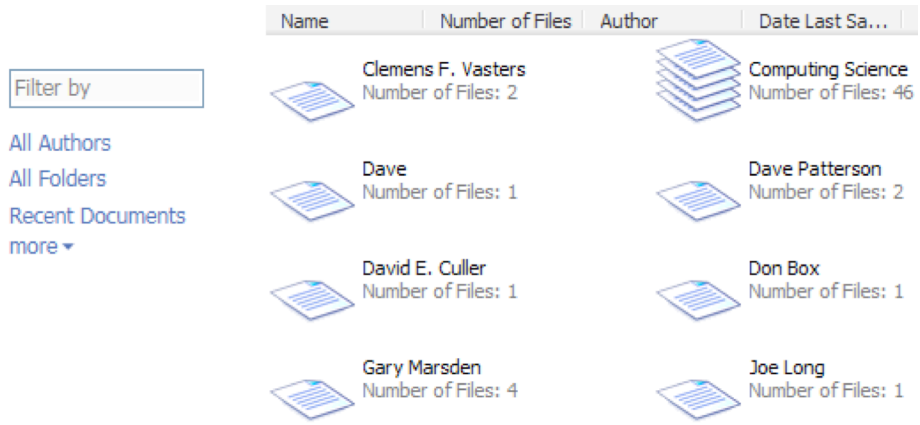

Figure 7 – Organizing the file store in Longhorn according to file authors

Now suppose that we wish to see all the documents written by "Gary Marsden" in October. Sadly, the menu on the left does not provide an option to rearrange the file store around date. Instead, this functionality is presented in the columns at the top of the page – "Date Last Sa…" from figure 7. The results of clicking the tab are shown in figure 8. Using attribute columns is strange in two counts: firstly, these attribute column headings are at the top of columns that do not exist (an icon view is selected) and secondly, it is not clear why some attributes are shown on the left menu, some are shown at the top of the screen and some are shown in both.

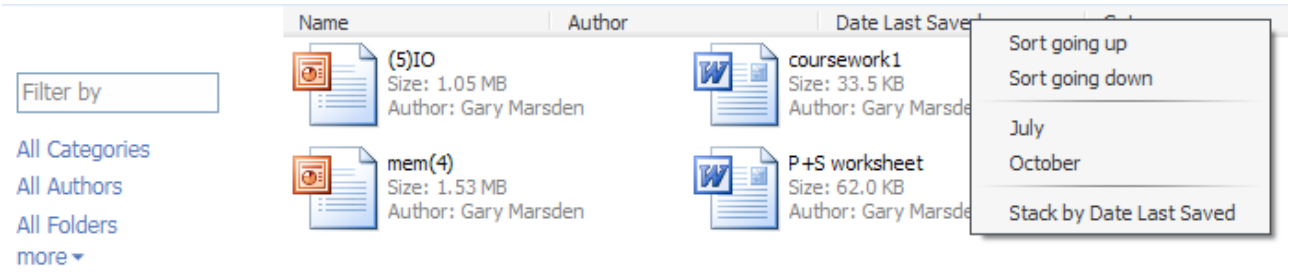

Figure 8 – Organising files by date saved

One reason for these design choices may be that some options (like "All Authors") are seen as queries on the relational file store whilst others ("Date Last Saved") are seen as display attributes. This leads the interface designers to create two different interaction techniques. However, to the user, the task is identical and it is not clear why two different interaction techniques are required.

Had the Longhorn designers used a coherent interface metaphor, this problem would not have arisen. Also, they have missed the opportunity to include key functionality such as union and difference.

It should be stated that the version of Longhorn evaluated was a pre-alpha created just for a developers' conference late in 2003. It is likely, therefore, that the interface will change before the final release in 2006. We can only hope that the developers take note of our comments and are able to produce a more integrated system in time for the final release.

8. EVALUATION

We were interested in evaluating two key aspects of the system; namely performance and usability.
To study performance, we set the task of implementing the browser to a second year data structures class. Working in groups, these students implemented seven different prototypes. These prototypes were then tested on a hard disk of 42,000 files. The fastest of these systems was able to process a query in well under a second (Pentium 4, 1.7 Ghz, running Windows XP with 512 Mb of RAM). This falls within Nielsen's one second limit [Nielsen 1994]. Admittedly, these systems needed to index the hard disk before performing the query, but the indexing process took less than five minutes – a task that could be performed whilst, say, a screen-saver is executing.
The usability evaluation of the system, however, is less straightforward. We conducted initial heuristic evaluations to refine the initial design. To fully evaluate the system, however, would require a long term study wherein the system is installed on various PCs and interaction logs kept over a period of time. Whilst this evaluation is in the planning stage, we can report that one of the authors has the software installed on their computer and uses it on a regular basis.

9. CONCLUSION

Since the 1970s, a lot of work has been conducted into database design, which has resulted in systems which allow computers to retrieve information more quickly. Over that same period relatively little has been done to improve the situation for ordinary computer users. We have attempted to leverage database, and HCI research into a system which organises documents according to the users', not the computer's, needs.

By doing so, we believe that we have created a flexible system with a coherent design. We have shown that the implementation of a system does not create large performance overheads but a long term usability study is required to see how and where this system could be adopted into a user's document management and retrieval strategy.

10. ACKNOWLEDGEMENTS

This work was supported by a travel grant from the Management Faculty at the University of Stirling and Microsoft EMEA. Implementation of the evaluation prototypes was undertaken by second year Computer Science students at the University of Cape Town.

11. REFERENCES

BARREAU, D. and NARDI, B. (1995) 'Finding and Remembering. File Organization from the Desktop.' SIGCHI Bulletin 27(3), pp. 39-43.
CARD, S.K., MACKINLAY, J.D. and SHNEIDERMAN, B. (1999) 'Readings in Information Visualization' Morgan Kaufmann.
CORMEN, T.H., LEISERSON, C., RIVEST, E. and STEIN, C. (2001) 'Introduction to Algorithms', Second Edition, MIT press.
DATE, C.J. (1993) 'An Introduction to Database Systems' Addison-Wesley, fifth edition.
DOURISH, P., EDWARDS, W., et al. (2000), 'Extending Document Management Systems with User-Specific Active Properties,' ACM Transaction on Information Systems 18(2) , pp. 140–170.
FREEMAN, E.T. (1997) 'The Lifestreams Software Architecture' PhD Thesis, Yale University.
GIAMPOLO, D. (1998) 'Practical File System Design with the Be File System.' Morgan Kaufmann.
GIFFORD, D., JOUVELOT, P., SHELDON, M. and O'TOOLE, J. (1991) 'Semantic file systems'. Proceedings of the Thirteenth ACM Symposium on Operating System Principles (Pacific Grove, CA), ACM Press, New York, NY.
JAVA (SE) 1.4.1 (2003) Package Tree, http://java.sun.com/j2se/1.4.1/docs/api/, last visited 9th May 2003.
Jones, W.P. 'The Memory Extender Personal Filing System' Proceedings of CHI'86. April 1986, pp.298-305.
JONES, W.P. & DUMAIS, T. 'The Spatial Metaphor for User Interfaces: Experimental Tests of Reference by Location versus Name' ACM Transactions on Office Information Systems, Vol.4, No. 1, January 1986, pp 42-63.
MCCARTHY, S., LEIS, M., BYAN, S. (2002)'Larger Disk Blocks or Not,' Proceedings USENIX FAST Conference, Monteray, CA, January 2002.
MANDER, R., SALOMON, G. & WONG, Y.Y (1992) 'A 'pile' metaphor for supporting casual organization of information.' Proceedings CHI 1992 pp.627-634.
MARSDEN, G., GILLARY, P., THIMBLEBY, H. & JONES, M. (2002) 'The Use of Algorithms in Interface Design.' International Journal of Personal and Ubiquitous Technologies, Volume 6(2) pp.132-140.
MICROSOFT (2003) "Longhorn Developers Center." http://msdn.microsoft.com/Longhorn/ Last visited 2nd March 2004
NIELSEN, J. (1994) 'Usability Engineering' Morgan-Kaufmann, Section 5.5
QUAN, D., BAKSHI, K., HUYNH, D. & KARGER, D. (2003) 'User Interfaces for Supporting Multiple Categorization' Proceedings Interact 2003.
RODDEN, K. & WOOD, K. (2003) 'How Do People Manage Their Digital Photographs?" Proceedings CHI 2003, pp. 409-416.
SCHAFFER, D. and GREENBERG, S. (1993) 'Sifting through Hierarchical Information' Proceedings CHI 1993, pp. 173-174.
SCOPEWARE (2004) http://www.scopeware.com/ Product Web page, last visited 10th March 2004
SHNEIDERMAN, B. http://www.cs.umd.edu/~ben/ Personal web page, last visited 13th May 2003.
SULLIVAN, K. 'The Windows 95 user interface: a case study in usability engineering' Proceedings SIGCHI 1996, pp. 473-480.
WHITTAKER, S. and SINDER, C. (1996) 'Email overload: exploring personal information management of email.' Proceedings of CHI 1996, pp.276-283