The global intelligent file system framework.

Gooch, Joanna

The Global Intelligent File System Framework

Joanna Gooch BSc. (Wales)

A thesis submitted to the University of Wales in candidature for the degree of Philosophiae Doctor

Department of Computer Science
University of Wales, Swansea

September 2006

Abstract

Since its inception the Internet has grown rapidly in both size and importance in our everyday lives. The Internet today is the preliminary model of what is commonly called the global information infrastructure. However, at the moment this "infrastructure" is considered to be an addition to our computer, and is not an integrated part of a file system which is essentially a "local information infrastructure" of a computer. Advancements in the sizes of disks in computers, network bandwidth and the types of media available mean users now keep large amounts of files in their personal data storage spaces, with little or no additional support for the organisation, searching or sharing of this data. The hierarchical model of file system storage is no longer the most effective way of organising and categorising files and information. Relying largely on the user, rather than the computer, being efficient and organised its inflexible nature renders it unsuitable for the meaningful coordination of an increasing bulk of divergent file types that users deal with on a daily basis.

The work presented in this thesis describes a new paradigm for file storage, management and retrieval. Providing globally integrated document emplacement and administration, the GIFS (Global Intelligent File System) framework offers the necessary architecture for transparently directing the storage, access, sharing, manipulation, and security of files across interconnected computers. To address the discrepancy between user actions and computer actions, GIFS provides each user with a "Virtual Secretary" to reduce the cognitive workload and remove the time-consuming task of information organisation from the user. The Secretary is supported by a knowledge base and a collection of intelligent agents, which are programs that manage and process the data collected, and work behind the scenes aiding gradual proliferation of knowledge. The Virtual Secretary is responsible for providing fast and accurate assistance to aid users who wish to create, store, retrieve, share, secure and collaborate on their files.

Through both system prototyping and performance simulation it is demonstrated that it is desirable as well as feasible to deploy a knowledge base in supporting an intelligent user interface that acts like a human assistant who handles paperwork, looks after filing, security and so on. This work provides the contribution of a new framework and architecture to the field of files systems and document management as well as focusing on reducing the burden placed upon users through everyday usage of computer systems. Such a framework has the potential to be evolved into a highly intelligent assistant to a user over a period of service and the introduction of additional agents, and provides the basis for advancements in file system and organisational technologies.

Summary

Since its inception the Internet has grown rapidly in both size and importance in our everyday lives. The Internet today is the preliminary model of what is commonly called the global information infrastructure. However, at the moment this "infrastructure" is considered to be an addition to our computer, and is not an integrated part of a file system which is essentially a "local information infrastructure" of a computer. Advancements in the sizes of disks in computers, network bandwidth and the types of media available mean users now keep large amounts of files in their personal data storage spaces, with little or no additional support for the organisation, searching or sharing of this data. The hierarchical model of file system storage is no longer the most effective way of organising and categorising files and information. Relying largely on the user, rather than the computer, being efficient and organised its inflexible nature renders it unsuitable for the meaningful coordination of an increasing bulk of divergent file types that users deal with on a daily basis.

The work presented in this thesis describes a new paradigm for file storage, management and retrieval. Providing globally integrated document emplacement and administration, the GIFS (Global Intelligent File System) framework offers the necessary architecture for transparently directing the storage, access, sharing, manipulation, and security of files across interconnected computers. To address the discrepancy between user actions and computer actions, GIFS provides each user with a "Virtual Secretary" to reduce the cognitive workload and remove the time-consuming task of information organisation from the user. The Secretary is supported by a knowledge base and a collection of intelligent agents, which are programs that manage and process the data collected, and work behind the scenes aiding gradual proliferation of knowledge. The Virtual Secretary is responsible for providing fast and accurate assistance to aid users who wish to create, store, retrieve, share, secure and collaborate on their files.

Through both system prototyping and performance simulation it is demonstrated that it is desirable as well as feasible to deploy a knowledge base in supporting an intelligent user interface that acts like a human assistant who handles paperwork, looks after filing, security and so on. This work provides the contribution of a new framework and architecture to the field of files systems and document management as well as focusing on reducing the burden placed upon users through everyday usage of computer systems. Such a framework has the potential to be evolved into a highly intelligent assistant to a user over a period of service and the introduction of additional agents, and provides the basis for advancements in file system and organisational technologies.

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed .... (candidate)

Date ....30th September 2006........

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed .... (candidate)

Date ....30th September 2006........

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed .... (candidate)

Date ....30th September 2006........

Acknowledgments

I would like to acknowledge the help of many people during my academic career at Swansea. To my supervisor, Professor Min Chen for his guidance and also Dr Phil Grant for his role as my second supervisor. Although not directly in a supervisory role, my gratitude goes to Professor Faron Moller for his support and witty reparté and letting me sit in the big comfy chair.

Any thesis acknowledgment would not be complete without a mention of those people who were initially responsible for my progress, development and bank balance. My parents have long tolerated having to tell people I'm a student, so I hope that this thesis not only makes them proud, but also gives them the relief of being able to tell people I'm a doctor and the knowledge that I love them both very much. I would also like to acknowledge my sister, for both being my big sister and for moaning until I was forced to mention her here.

Two other people were also completely indispensable to me during my academic career. An honorable mention goes to Chris Moutray for his love and support and also to Roz Swayne, who went above and beyond the call of duty. Without them this thesis would have never existed and I might not either. I owe you both so much.

I also have to acknowledge the motley crew of friends otherwise referred to as "The Pouch". To Andy Gimblett, Basheera Khan, Jason Bees and Dave Brooks for cheering me on, providing endless hours of entertainment and inventing new words; David Chisnall for his technical help, constant reassurance, amusing camera work and enduring my comments on his personal life; and Will Harwood for being my bicycling companion and a fantastic kinky butler (you make a mean cup of coffee).

For those people who had to put up with me jumping around the lab, I offer both apologies and thanks. My first year would not have been the same without the acrobatics enabled by Gareth Daniel, my last year would have suffered were it not for my surrogate big brother Will Thimbleby, and all the years that went in between would have been plain wrong without Alfie Abdul-Rahman. A special mention goes to my test subjects for my second case study: Min Bhogaita, David Dalton, Will Harwood, Kay Patel and Peter Robinson. I'm sorry for taking such delight in your memory failure!

There are several other people whom I would like to thank in the little space I have remaining: Justin Mitchell for his friendship over the last seven years; Lidia Oshlyansky for our Saturday morning brunches and being the voice of sanity; Nicola Morgan for chatting and singing; Dr Andy Jones and Dr Andrew Blyth for giving me something I wanted to work toward and Andy Price for everything I could possibly need from a friend.

Contents

Chapter 1

Introduction

1.1 Motivation

Anyone who has used a computer regularly over the past 5 or so years will be able to testify at the increasing amounts of data which we are bombarded with on a daily basis. Most users have no idea how to best store, organise and search their own data, and even the best user-created hierarchy structures inevitably contain a folder named "stuff", "random", "misc" or "to sort" containing all the files that did not seem to fit anywhere else or required extra thought to be categorised. Computers have always been hailed as the answer to our problems of time-consuming and monotonous tasks, but the up-keep and care taking of using these systems is becoming increasingly time consuming in itself.

The main problems that users face is deciding what should be done with a file. First of all deciding how important it is, then considering whether or not it should be kept in a "working" directory, or a more permanent storage/archive directory. Then if it requires the extra security of setting a password or encryption, who should be allowed to see it, who should be allowed to edit it, and whom it should be hidden from. This is without considering the categorisation of the file that most users have to undertake in order to place a file on the physical disk in a nested set (or tree) of directories.

Consider that in a paper-based office there is usually at least one secretary or clerical assistant who performs a variety of organisational tasks. This person is responsible for the paper filing needs of one person or a group of people, and can store files in and retrieve them from an assortment of places such as a filing cabinet, a safe or someone's desk with appropriate confidentiality. It is not unusual for this person to also open, filter and forward incoming mail to the required individuals, bring time sensitive documents to the owner's attention, copy documents and deliver them by hand to the intended recipients.

Whilst the advent of emails and the proliferation of networked offices has changed the role of a secretary somewhat, the analogy remains useful. The idea that within an electronic file system users should be without such assistance is similar to suggesting that a manager of an office should be responsible for dealing with all their own paperwork, resulting in a desk piled high with stacks of documents which there is little time or inclination to deal with. Running an office in such a manner would no doubt cause a rapid degradation in terms of productivity and service and is obviously not the ideal situation.

In the modern workplace (and to a slightly lesser extent, at home) people are now acquiring incredible amounts of data through everyday activities. Not only are text-based documents created electronically for use in the workplace, jokes and 'virals' are circulated amongst friends via email, users are constantly searching and saving information from the World Wide Web and taking photos or videos with digital cameras. The sheer amount of data now available to users means they no longer have the time nor the organisational skills to process and store it all.

Hierarchies have been the structure of file systems since the 1970's, but their inflexible nature means that they are no longer suited to the everyday needs of computer users [207]. Consider a user looking at a set of photos which were taken on holiday. An example of directory path they might have stored their photos in could be 'My Documents/My Pictures/Photos/Holiday05'. This seems almost sensible until you realise there will be a multitude of other file types associated with their holiday, perhaps video files, digital maps, diaries or itineraries or saved web pages of travel arrangements. None of these files could be categorised as being pictures, or photos and so do not belong in those directories, yet they are all related to the holiday and the photographs themselves. If all these files were to be put in the directory 'My Documents/Holiday05' it would create difficulties when a user wanted to browse their entire collection of photos. Users need files to be categorised by several different attributes, but the structure of hierarchical file systems does not allow for this.

While a hierarchy seems a logical way for a computer to organise and store files, it is certainly not a logical way for humans to do the same [180]. There is no guarantee that a user will be able to create a suitable storage hierarchy, nor that they will use it consistently and without error. Users are prone to deleting the wrong versions of files without a back up from which the original can be restored. Applications (such as Adobe Acrobat) often have pre-defined directories that files are saved to by default. If, without thinking, a user saves a document without properly reading the file dialogue box (perhaps if they were in a rush) then a file could be stored somewhere seemingly random. There is also the case where an application saves multiple files in one action without letting the user know explicitly. For example, saving a web page from within an Internet browser will save not only the HTML page but will also generate a folder containing the images and other files included in that page.

File organisation is a perpetual activity for every computer user. However, the level of difficulties in this activity is becoming increasingly noticeable, largely due to information explosion and deficiencies in current file systems. For example, many managers and secretaries are constantly looking for extra disk space for storing documents, or looking for files previously created on their computers. The hierarchical tree structure available in most file systems (e.g., directories and folders) is a satisfactory mechanism for short-term and small-scale document organisation. However, it becomes less user-friendly, often clumsy and problematic when dealing with a large volume of files that are to be maintained over a long period. The sheer volume of information users are confronted with reduces their likelihood of organising their documents. Categorising files and creating complex hierarchies for suitable file storage can be a time-consuming task which places extra cognitive load on the user [238].

Current ***search*** functions also lack in the required flexibility for users to deem them useful. A deficit of searchable criteria, a slow result time and poor quality of results are all reasons that discourage users from using search functions that are built into their file or operating systems. This is another example of how the file system operates on the agenda of making the task easier for the computer, not the user. There are no options to "find more files like this one", even though comparisons are a perfectly normal and logical path to follow in human thought processes.

Where there are document management systems with better search capabilities in place, it is unlikely that a user will spend the required time and effort in "training" a digital personal assistant, even if it would save them time in the long run [114]. The same can be said for archiving older files. Archiving is a process rarely used by users, partly due to the growing size of storage media, but also as it is an extra housekeeping task for a user to perform.

File sharing is perhaps the most common activity in a collaborative environment [88]. A user may typically create a document, and wish to distribute it to a group of other users. There are usually some additional requirements associated with this task, such as read/write access permission, and transmission security. Despite the fact that a variety of mechanisms may be used for supporting this activity, they lack in either user-friendliness or security, and in comparison with a human secretary, most of the mechanisms leave a lot to be desired. For example, email attachments, perhaps the most commonly used mechanism, may incur unnecessary duplication and excessive space wastage. Many organisations and computer users are being inundated with email attachments sent to them endlessly, and often pointlessly. Uploading a document onto a web site is another mechanism typically for read-only file sharing. It however requires some technical skills for setting up the service at the server-end. Networked operating systems such as Windows also offer mechanisms for shared user space. Nevertheless, most of these mechanisms are based on file owners or groups, rather than on individual files. To most users, setting up a mechanism in order for others to share a specific file is not really a trivial task.

In an age when almost every facet of our lives involves interaction with some kind of computer system, ***file security*** takes on an increasingly important role. Most file systems rely on the security mechanisms provided by the operating system to keep its contents secure. However, if these security measures usually consist of a password login and basic firewall which would provide no major problems to a persistent hacker, especially as many scripts or executables are freely distributed across the Internet for such purposes. If these mechanisms were to be by-passed then an intruder could gain unlimited access to the contents of the files. A way to combat this problem is by using an encrypted file system, but such systems require considerable set-up and administration, a task that non-technical users would doubtful be able to manage. File systems offering per-file encryption rely on the user remembering passwords or keys in order to decrypt their files [40]. This could lead to one user having to remember a variety of different passwords that should (if the password is to be considered secure) be comprised of a seemingly random collection of characters including digits and symbols. Encrypting the contents of a file system also creates problems for people working in a collaborative environment. If an encrypted file is to be shared amongst users or have multiple authors then there needs to be a mechanism for key sharing, and users should not be expected to remember the keys for each file.

Although both previous and current technologies have individually addressed the problems related to file management discussed above, there has yet to be a combination of these technologies to provide a complete solution. Digital personal assistants [176] have been context-specific applications that do not solve the underlying problems of file storage but provide additional organisational capabilities to users. In particular, previous systems have all required the user to specify the location of a file within the file system. The 'secure' and encrypted file systems have not been designed to scale over a large network or to be deployed in a collaborative environment with multiple users accessing and editing the same files. Knowledge-based systems and tagging [38] are growing in popularity as this approach to knowledge management is capable of dealing with the increasing amounts of data that users are producing. Adaptive and search-based interfaces are allowing users to locate information more easily [1], but these systems do not remove the burden of file location, security management and the other activities commonly associated with file system management.

1.2 Aims and Objectives

The aim of this thesis is to offer an alternative paradigm to the design, implementation and deployment of current file systems. Recent changes in the amount of computer data stored and accessed by users has meant that the older, monolithic file systems both no longer service the needs of users or are being utilised for the purpose originally intended.

However, these systems have shaped the way people think about and use computers, so the *first objective* of this thesis is to provide a thorough review and critical analysis of previous works on operating systems, file systems and security, knowledge bases, intelligent agents and the Internet. In particular, the design ideas behind previous and current file systems are re-examined and critiqued.

The *second objective* of this thesis is to propose the concept of a *global file system*. All the required technologies are already in place, including the global infrastructure of the Internet, but they have yet to be combined in such a way to provide scalable, secure and consistent file access. The global file system should offer increased functionality over existing file systems, catering for a wide-range of user needs including document placement, version control, support for collaborative environments, secure storage, and powerful and intelligent search functions through an adaptive interface. The addition of these extra features in a file system will not cause disadvantages for the user that are usually associated with these utilities. The use of such a file system will reduce the time, effort and cognitive ability needed to manage files as well as providing a simple but powerful service to users even with little technical expertise.

The *third objective* is to design the knowledge-based framework of the entire system, particularly those parts involved with the production, maintenance and analysis of the knowledge bases as well as the optimal storage of files. With such a generic framework in place, we aim to develop and implement the separate parts required to fulfill the concepts presented. This will include the structure of the knowledge bases, the user interface, integration with the operating system, communication protocols and the file storage system.

The *final objective* is to demonstrate the practical feasibility and scalability of the system through simulation of extended use. Through these studies it will be shown that a global file system (and in particular, the knowledge based approach) could scale to long term service, and also that over time the system could improve the services offered to users by means of adapting and predicting the user's needs. The analysis of the results of these tests will be used to facilitate further improvements.

If the above objectives of this thesis were to be met, the main contribution of this work to the scientific community would be (i) the introduction of a new conceptual framework for a file system that is centered around a knowledge base in conjunction with transparent internal organization of files for global access (in place of the traditional user-defined hierarchical file organization) and intelligent search-based file archiving and retrieving facilities (in place of the traditional listing and browsing facilities); (ii) a good understanding, supported by experimental results, as to whether or not such a framework is scalable in relation to the knowledge accumulation and the future computational resources; (iii) a collection of new methods proposed in the case that a naïve approach cannot achieve the required scalability;(iv) a proposal of the technical architecture for implementing such an architecture, supported by an investigation into the technical feasibility of a collection of selected system components which are considered to be non-trivial in their integration into a file system. Although this may not be a definite guideline for any commercial implementation of a new file system, it would be a first design exercise and feasibility study for such an ambitious concept ever reported in the public domain.

This thesis is, however, not intended to deliver a complete file system based on such a conceptual framework, which is most likely to be an unrealistic objective of a PhD programme. It is not intended to prove through user studies that such a file system would be superior to the conventional file system, which would be a valid and useful study if there were existing prototypes of a comparable quality and completeness to the commercial file systems.

The work conducted for achieving the above-mentioned objectives are detailed in Chapters 2 - 7, and the assessment of the scientific contributions are discussed in Chapter 8.

1.3 Outline

Chapter 2 will provide the reader with a background knowledge of file storage and management systems, including a detailed look at the architectures and histories of the most important advances in storage technologies, document management systems and the Internet. By studying the historical development of file systems on numerous platforms, a further understanding of how situational and technological developments influenced file system advancements can be obtained. Following the historical overview, the more significant file system architectures are studied in more technical detail. The issues faced by current and future file systems are presented, including discussions on interface design and security. Finally, this chapter looks at the development and growth of the Internet, not just as a communications medium but with special emphasis on its similarities to the more 'traditional' file systems, including the areas of addressing, naming, searching, security, interface development and growth.

Chapter 3 introduces the reader to the problems of "Information Overload", a concept which is now all too familiar but by no means a new development. This is followed by a background discussion of current issues facing those in the fields of information retrieval, file system design and artificial intelligence. The designs, uses and drawbacks of current digital personal assistants, office assistants, personal search agents and information retrieval systems are presented, along with an overview of document management and classification systems. Chapter 3 aims to equip the reader with a basic knowledge of file retrieval, artificial intelligence systems, digital personal assistants, agent technology and knowledge-based systems.

Both chapters 2 and 3 indicate material for further reading.

An in-depth overview of the uses of GIFS is presented in chapter 4. From the perspective of a computer user, it is demonstrates how the deployment of such a system could save both time and effort. A background is provided to give the reader a strong understanding of the ideas and concepts that lead to the development of a "Virtual Secretary" and is further reinforced by a detailed example of a file life-cycle. Each feature within GIFS and the Virtual Secretary is explained, without the need for in-depth knowledge of the system architecture.

Chapter 5 focuses on the aspects of knowledge acquisition, processing and dissemination within GIFS and takes a closer look at the under-lying data-mining and agent technologies. Firstly the technique for gathering data from human interactions with the system is examined, leading onto the design principles of the knowledge base, the terminology and definitions used and the internal structure of the data. With the basic concepts in place, the life-cycle of data and knowledge within the system is explored, including the algorithms and formulae which are used to specify search results and suggestions, both pre-calculated and real-time. The technology behind the intelligent agents required to build such a system is presented, along with classifications of each agent within GIFS. The tasks of each agent are described with in-depth examples of the data created and analysed in order to support the Virtual Secretary.

Chapter 6 encompasses two experimental studies in order to measure the scalability and feasibility of the system. For both, a detailed example is provided of a scenario of file system usage where a global file system would be advantageous. Firstly, this chapter looks at the problems of knowledge base and data growth, using data which was collected from previous file system studies and created institutionally. In order to predict the behaviour of file and data growth, a simulation program is presented to provide data to analyse in the case study. The study in itself comprises of the analysis and measurement of several different algorithms and techniques to show how a Virtual Secretary system could save the user time. The second case study looks at the issue of feasibility and more specifically how much time and effort the system could be expected to save a user. By using an example of the ever-changing access lists of a file throughout its life-cycle, it is shown how the different settings will cause the Secretary to behave in different ways in order to provide a more useful service to the user.

The technical details of GIFS can be found in Chapter 7, including details of the underlying system design and a discussion on the merits and drawbacks of alternative architectures. The design and implementation of the communications protocols and authentication methods deployed are examined in conjunction with the encryption and security features provided by the system. Chapter 7 will provide a technical illustration of the system implementation and how it interfaces with current operating systems and networks. Perhaps most importantly, this chapter contains a detailed walk-through of a typical file added by a user to the system, demonstrating the design concepts discussed previously and the practical operations which are transparent to the user.

Chapter 8 contains a critical assessment and evaluation of the research with respect to the aims and objectives. This chapter will provide an overview of the contributions that this thesis has provided to the scientific community. It will provide an overview of the main highlights that have resulted from the research presented hereof. Furthermore, this chapter will identify the future work which would advance the concepts presented herein.

Chapter 2

File Storage and Management

Since computers were first used as a storage medium, the design and implementation of file systems has been a major research area for both academic institutions and software manufacturers. With modern disks and operating systems now capable of storing huge amounts of data, file systems have had to constantly evolve in order to meet the user's needs and take advantage of new technologies.

This thesis proposes a framework for a global and intelligent file storage and management system, and examines its use and implementation mainly from the perspective of file systems. This review chapter provides us with the background of the evolution of file systems developed in conjunction with various operating systems. The chapter also briefly examines the Internet as a quasi-file-system, focusing on the features relevant to the notions of a traditional file system. This chapter helps underline the novelty of the proposal of GIFS (Global Intelligent File System) detailed in Chapters 4-7, and in particular the feasibility of the architectural design detailed in Chapter 7.

2.1 Historical File System Development

Over the years, file systems have been developed for a number of different platforms with different storage needs. Table 2.1 shows a basic time line of file system development and the significant contributions generated by each.

File System	Made By	Year	Key Features
CP/M	Gary Kildall	1977	No subdirectories, but 16 “user areas”, no security
FAT12	Microsoft	1977	Designed for floppy disks, maximum partition size 32MB
V7FS	Bell Labs	1979	i-nodes, single, double and triple indirect block addressing
FFS	Kirk McKusick	1983	Cylinder blocks, long file names, 5-attribute directory entry, 2 block sizes
AFS	CMU	1984	Large-scale, transparent, low-bandwidth, distributed file access
NFS	Sun	1984	Protocol for distributed FS. Built on ONCRPC4
HFS	Apple Computers	1985	Successor to MFS, B*tree structure, GUI
NWFS	Novell	1985	Heavily modified version of FAT
FAT16	Microsoft	1987	Allowed multiple disk partitions
HPFS	IBM & Microsoft	1988	B-tree structure, Allowed long filenames and 3 timestamps per file
JFS	IBM	1990	First commercially successful journaling FS
VxFS	Veritas	1991	First commercial journaling file system
NTFS	Microsoft	1993	Quotas, alternative data streams, sparse files, reparse points, volume mount points, hard links, file compression and encrypted storage
LFS	Margo Seltzer	1993	Log-structured with similar on-disk format to UFS
XFS	SGI	1994	Journaling FS
FAT32	Microsoft	1996	Long file names, 32-bit cluster field
BFS	Be Inc.	1996	Journaling FS with extended file attributes, indexing and querying
HFS+	Apple	1998	Unicode names, long filenames, 32-bit block addresses
GFS	Sistina	2000	Non-client server model for shared storage across Linux clusters
ReiserFS	Namesys	2001	First journaling FS to be included in standard Linux kernel, metadata only journaling
UFS2	Kirk McKusick	2002	64-bit block pointers
Fossil	Bell Labs	2003	Snapshots available to all users
GoogleFS	Google	2003	Optimised for large files and hardware failures
ZFS	Sun Microsystems	2004	Extremely large capacity
WinFS	Microsoft	–	Relational databased FS

Table 2.1: Important file system developments

2.1.1 MSDOS and Windows

2.1.1.1 FAT12

The initial version of FAT is now commonly referred to as FAT12 and was written by Bill Gates and Marc Donald in 1977 for managing disks in Microsoft Disk BASIC. It was originally intended for use on floppy disks only, there were no hierarchies and the maximum partition size was 32MB. The maximum number of files allowed was a couple of dozen as the sole root directory had to fit on the first track of the disk. In 1983 with the release of MSDOS 2.0, hierarchical directories were introduced which added capacity as the overall size was no longer restricted to the fixed size of the root directory. There could now be as many files as there were numbers of clusters.

2.1.1.2 FAT16

The original FAT12 file system was only intended to work on floppy disks, so when IBM brought out a new PC featuring a 20MB hard disk in 1984, Microsoft released MS-DOS 3.0. FAT16 was named as the cluster addresses were increased to 16-bit which enabled much bigger file systems as the number of clusters could increase to 65,517. However, in the initial version of FAT16 the maximum number of sectors and maximum partition size of 32MB was not altered.

Hard disks can have up to 4 primary partitions, however MS-DOS would only use the partition marked as active which was also the partition it would boot. In January 1986 MS-DOS 3.2 was released which introduced a new kind of partition, an extended partition. The extended partition acted as a container for additional partitions called logical drives and to begin with only one logical drive was allowed. By the release of MS-DOS 3.3 in August 1987, up to 24 logical drives were allowed (a limit most likely imposed from the C: - Z: disk naming system), but still only inside one extended partition.

The second version of FAT16 was released in November 1987 in Compaq DOS 3.31 with the 16-bit disk sector being increased to 32 bits. Although there were only minor changes to the way things were stored on the disk, the entire DOS code had to be rewritten to use 32-bit numbers, which was made more difficult by the fact it had originally been written in 16-bit assembly language. In 1988 these changes were released to a wider audience in MS DOS 4.0. Partition size was now restricted by the 8-bit signed count of sectors-per-cluster (maximum 64^2). Sectors were 512 bytes which allowed 32KB clusters and thus the limit for a FAT16 partition size was set at 2GB. FAT16 is still used on most removable media (e.g. USB flash drives and camera memory cards) as it has better performance than NTFS over small volumes.

2.1.1.3 NTFS

The New Technology File System was first seen with the release of WinNT 3.1 in July 1993. It is the standard file system of Windows NT and its descendants: Windows 2000, Windows XP and Windows Server 2003 [76]. Other versions of Windows are unable to read NTFS volumes without the assistance of third-party utilities. There are currently 5 different versions of NTFS, with each providing additional features [244]. Overall it has introduced the concepts of quotas, alternative data streams, sparse files, reparse points, volume mount points, directory junctions, hard links, hierarchical storage management, native structured storage, volume shadow copy, file compression, single instance storage and encrypting FS [192]. The main drawback of this file system is the lack of support from non-Microsoft operating systems which has been caused by Microsoft keeping the exact specifications secret. There now exists a reverse engineered implementation of NTFS which runs on FreeBSD and Linux that offers both read and write support, but for many years non-Microsoft operating systems only had the capability to read from NTFS file systems (if at all).

2.1.1.4 FAT32

The volume size limit of FAT16 caused Microsoft to implement a new version of FAT to release with Windows 95, that is, FAT32. Cluster counts are held in a 32-bit field, with 28 of these currently in use. Mathematically speaking, FAT32 should support $> 2^{28}$ clusters, allowing drive sizes to be up to 2TB, however an error in Scandisk won't let FAT grow to over 2^{22} clusters, so the volume limit is 124.55GB. The maximum possible size of a file on a FAT32 volume is around 4GB which was more than adequate at the time of release. In more recent years this has become an annoying restriction for some users as modern applications regularly create files that exceed this limit.

Windows ME was the last Microsoft operating system based on MS-DOS, and hence there have been no more FAT file systems. Later versions of Windows can read FAT32 but are restricted to creating FAT32 partitions of 32GB or smaller [193], and although this restriction is documented [282] it has not been explained, leaving some experts to speculate that "Microsoft intentionally crippled the FAT32 file system" [209].

2.1.1.5 WINFS

Windows Future Storage is not a completely new file system, as it was intended to be built on top of NTFS [234]. It was first expected to be released with Vista, the new Microsoft operating system in 2006, but is now not expected until late 2007 at the earliest [307], if at all. It flattens the hierarchy to a single directory and uses a relational database to enable searching on file attributes [120]. It is implemented in SQLServer, C# and C++ [235], and can allow the synchronisation of external storage, using agents for the tasks of data gathering and sorting [308]. It is similar to concept to the Be File System, released many years earlier [213] which can also been seen in §2.1.4.6.

2.1.2 MacOS

2.1.2.1 HFS

Macintosh computers originally used the Macintosh File System (MFS) introduced in 1984. As Macintosh systems stored more data than the other file systems available at the time, Apple developed an entirely new file system. It was only designed to work on small and slow media, so several performance enhancements were made, (such as referring to files by handle instead of name and storing all file and directory information in a single file) which worked well until large volumes caused serious performance problems. The time needed for MacFS to display the contents of a large directory was very slow due to the way it had to search through all the files with a matching directory handle. This prompted the development of HFS in 1984. The directory structure of MacFS was replaced with a B* tree that could be searched very quickly regardless of the number of files [17]. As with advancements in other file systems, HFS also increased the size of various structures to hold larger numbers, with 32-bit numbers replacing 16-bit nearly everywhere. One of the instances that this increase did not take place in was the restriction on the file directory which would still only hold a total of 64K files.

In the same way as MacFS, HFS stored all the locations of files in one single directory structure called the Catalog File. This made it radically different to the other file systems available at the time, where directory information was stored and organised by each particular directory. Performance could be affected if using the system when multitasking, as active programs would all be stuck waiting to write to the catalog file if one program was already using it. When HFS was first written, the Macintosh did not have multi-tasking capabilities so the possibility of this bottleneck was not a reality. Compared to other file systems available at the time, HFS was perceived to run a lot faster due to the caching and searching of the B*-tree structure.

2.1.2.2 HFS+

HFS+ (also known as MacOS Extended) was released with MacOS 8.1 in January 1998 to replace HFS as the primary file system used on Macintosh computers. HFS became limited when disks began to approach 1GB in size due to its 16-bit allocation mapping table. Even small files would take up the space of one allocation block, meaning that a large amount of space was being wasted. HFS+ uses a 32-bit allocation mapping table to reduce this problem, as well as supporting much larger files (block addresses are 32-bit instead of 16) and permitting long filenames up to 255 characters in length. Another difference between this and the old version of HFS is the use of Unicode instead of MacOS roman for the naming of files and folders. Since its first release Apple has included support for optional journaling features and case sensitive file and directory names. The access control list, which was based on the file security added in MacOS Server 10.4, was designed to be fully compatible with the system used in Microsoft Windows XP and Windows Server 2003. Some MacOS versions support only a subset of the HFS+ format, which has the effect of limiting the capacity and maximum number of files and folders allowed within a folder.

2.1.3 Unix/Linux

2.1.3.1 V7FS

Originally Unix file systems were derived from MULTICS, as two of the main contributors, Kenneth Thompson and Dennis Ritchie were heavily involved with both projects. As such, Unix file systems tend to include sophisticated multi-user support. One of the most well-known versions is the Unix version 7 file system [277]. The file system is a directed acyclic graph from the root directory. File names can be between 1 and 14 characters long, and composed of any character except '/' (the path separator) or NUL. There is a directory entry for each file in a directory and its i-nodes. The directory entry only has 2 fields, the file name (14 bytes) and the i-node number (2 bytes), which gives a maximum number of files limit as 64K. The i-node contains the attributes of file size, creation, last accessed and last modified times, owner, group, protection and the number of directory entries pointing to this i-node. In order to handle larger files V7FS uses single, double and triple indirect block addressing.

2.1.3.2 FFS

The Berkeley Fast File System (FFS, also known as UFS1) was originally derived from V7FS and is used mostly by BSD-derivative distributions of Unix [255]. It introduced longer file names, with the limit being increased from 14 characters to 255, and the largest improvement in performance was caused by using a larger block size than previous systems [278]. It also introduced an expansion of attributes in the directory entries for files, which now contained 5 instead of 2 fields. These were i-node number, entry size, type (e.g. file or directory), size of filename in bytes and the filename (padded to the 32-bit boundary) [191]. Another new feature was that of dividing a disk into cylinder groups, each with their own superblock, i-nodes and data blocks. This approach tried to keep all related data of one file close together to decrease disk search times. For the first time in a file system FFS included two different block sizes, not just one. It is more efficient for large files to be stored across a small number of large blocks than many small blocks. As most Unix files are small, having only large blocks would be a waste of space. The extra efficiency that this provides is balanced out by the extra complexity in the code that was required to implement it [189]. The reliability of the file system was improved by the careful ordering of disk writes [190]. After a crash, fsck (the the disk checker program, [188]) can recover the file system to a usable state more quickly by deducing from inconsistent data what happened directly before the crash. All the metadata writes are forced synchronous and not buffered in memory, so once the calls that caused the change has completed, the data has changed on the disk.

FreeBSD's UFS2 was developed mainly to offer better support for extended attributes, but also included the addition of 64-bit pointers, lazy i-node initialisation and support for variable-sized blocks. The rest of UFS2 remains similar to UFS1. with additional contributions from Kirk McKusick and Paul-Henning Kamp.

2.1.3.3 AFS

The Andrew File System originated at Carnegie-Mellon University in 1984 as a distributed file system [276] for accessing files across a campus wide network. It was part of a project started in the early 80's which was named after the university's main benefactors, Andrew Carnegie and Andrew Mellon. It was first required to support up to 8000 users, providing storage for the files of both students and staff with minimum bother. Compared to other distributed systems this was a very large number of users and so had a major effect on the design, with no centralised algorithms being used [135]. The main concept of the file system was to reduce network traffic across the back bone as much as possible and do most of the work on the workstations. This was achieved by arranging workstations into clusters, each with a file server and each cluster then being connected to the backbone. There was still the possibility that a user could be on a workstation far away from the file server holding their files so some network traffic was to be expected. Workstations and servers both ran versions of Berkeley Unix, although they both ran slightly different software. The workstations ran client code called Venus which was in the kernel, and the servers ran a program called Vice which was not originally put in the kernel but was later moved there for efficiency [279]. Each server and workstation had a hard disk drive, on the clients they were used to store temporary files and cache requested files only. As the workstations were not used for non-temporary storage, only the servers needed to be backed-up. All the traffic across the network is encrypted, and clients are never trusted by the server [199]. The directories are protected by control lists, but each file still has the 9 Unix rwx bits for compatibility [58].

2.1.3.4 LFS

The Log-Structured File System was originally developed in 1993 by Margo Seltzer of Berkeley. The majority of the system is a reimplemented version of the Unix file system, but optimised in order to take advantage of the increases in processor and memory speed and size [182]. The slow access times of a hard disk create bottlenecks in file systems so in order to use the full bandwidth of the disk the Log-structured File System was created [241]. LFS treats a disk as one giant log, with all pending writes buffered to memory and the written to the end of the disk in one contiguous segment [242]. The similarity between LFS and UFS means that the lower level Berkeley FFS code in UFS can be replaced by LFS code, whilst sharing higher-level UFS code with FFS.

2.1.3.5 XFS

Silicon Graphics started the development of XFS in 1993 for their IRIX operating system (their version of Unix). It was released in 1994 on IRIX 5.3 as a high-performance journaling file system. It supports journaling, 64-bit files and highly parallel operations. Unlike most other file systems, it uses B+ trees for most internal structures to give a high degree of efficiency [236]. Most file systems use a bitmap to manage free disk space, but XFS uses two B+ trees. One tree records the free space ordered by starting block number, the other B+ tree sorts the free space by length. This gives a very fast and space efficient way for finding the appropriate amount of free space for a file. The space needed on disk for the allocations of i-nodes is allocated as needed, with their locations stored in yet another B+ tree. Instead of using direct, indirect, double-indirect and triple-indirect blocks with fixed sizes for files, a B+ tree is used, with another being used to store the contents of directories, giving much faster search times for large directories than in traditional list-based file systems. As it was designed to be massively parallel, it uses a fine grain locking system which allows multiple reads and single write to a file at the same time [261]. In May 2000 it was released under the GNU public license, and as a result is available in almost all Linux distributions as a choice of file system.

2.1.3.6 GFS

The Global File System began life as a PhD thesis by Ken Preslan at the University of Minnesota. A research group looking at ocean currents needed a way to sorts lots of simulation data that could be processed by their cluster. It was originally written under IRIX, but soon ported to Linux for ease of writing. The basic idea was to have symmetric access to a storage device that was shared among many computers, with a single system image so all the machines saw the same file system [220]. With most cluster-based file systems, management is a big issue so lots of work was put into creating a fine-grained locking system. The requirement of locking individual i-nodes lead to a slightly different disk layout compared to other file systems. First of all there is a gap (where other systems would normally have a boot block) to prevent someone accidentally over-writing the data on the disk. Next on the disk comes the superblock, then the resource groups and finally a number of journals. A resource group is similar to an ext2 or 3 block group that can be locked independently.

In GFS2 the journals are scattered around the disk like "normal" i-nodes. If a node fails, another node can recover that node's journal but must first fence it off from the rest of the system to make sure it was not just taking a long time to complete an operation [226].

2.1.3.7 Resier3&4

When it was introduced in 2001, ResierFS [228] (referred to as Reiser3 after the announcement of the next version, Reiser4) offered features that were not available in the most-used Linux file system of the time, ext2. The most publicised addition was the facility for metadata-only journaling and it also had the ability for online resizing, (growth only) and tail packing (a scheme to reduce internal fragmentation). ReiserFS stores all the data (metadata, directory entries, i-node block lists and tails of files) in a single B+ tree, assigning each item with a universal object ID. The single tree design was intended to avoid the problems caused by linear time directory lookups and limitations on the total number of files due to i-node location calculations using a fixed formula [53].

Reiser4 is a new version of the ReiserFS, but written completely from scratch. Using a type of B* tree called a dancing tree [7], the underpopulated nodes are only merged when flushed to disk unless under memory pressure or when a transaction is completed. The lack of fixed blocks means files and directories can be created both space and time efficiently. It has yet to be included in the Linux mainline kernel due to coding issues arising from its use of 64-bit numbers in certain places.

2.1.3.8 GoogleFS

The Google File system was created to meet the data processing needs of the Google search engine. It was created by Sanjay Ghemawat, Shun-Tak Leung and Urs Holzle in 2000 for use on Linux. The design differs from traditional distributed file systems in a number of ways borne out of the specific demands of the Google file servers. Firstly, as the machines Google use for processing data are both numerous and relatively inexpensive, hardware failures are to be expected, some of which will be unrecoverable. This created the need for the file system to be capable of monitoring and detecting possible errors, a high level of fault tolerance and a system for automatic recovery [162]. Secondly, the files that Google uses are larger than those traditionally created by most file system users, so parameters such as block sizes have been optimised. Thirdly, files are appended to rather than overwritten and rarely deleted, so most of the optimisations for writing data concentrates on appending rather than random writes to the middle of files. The atomic append operation allows multiple clients to append to the same file at the same time without the need for extra synchronisation between them.

2.1.4 Others

2.1.4.1 CP/M

The first file system of note was developed for the CP/M (first called "Control Program/Monitor", later renamed on commercial release to "Control Program for Microcomputers") operating system in 1974. It was originally written as a pet project of Gary Kildall, which grew to become a commercial product released in 1977. File names were made of up to 8 characters followed by a full stop, and then an extension of up to 3 characters identifying the file type [292]. There were no sub-directories in the file structure, but in order to make it more compatible with multi-user systems the concept of user 'areas' was introduced. Placing a file in user area 1 would make it inaccessible to all other users, although it was possible to place files in areas independent of the currently set user area. As CP/M was designed as a single-user operating system there were no security checks in place to prevent a user from accessing each of the 16 user areas successively.

CP/M and early versions of MS-DOS were very similar internally, and identical in terms of the file-handling data structures and disk drive letter naming. The user area concept found in CP/M was never ported to DOS, the main innovation of DOS being the inclusion of the FAT12 file system [280].

2.1.4.2 NFS

NFS is a protocol which was developed by Sun Microsystems in 1984, defined in RFCs 1094 [269], 1813 [57] and 3530 [260]. It defines a remote FS allowing users to access files on a server as if they were working on a local disk. This system allowed many lower-cost computer systems to share one high capacity disk at the same time. Built on Open Network Computing Remote Procedure Call (ONCRPC), it is used mainly on Unix but can also be ported to MacOS and Windows which has let Unix system users connect to servers on a variety of platforms. Offering an almost transparent service, a user logs in to a workstation which automatically mounts the server disks and works on the files as if they were stored locally.

Since the beginning, there have been four major revisions of NFS, version 1 was never released to the outside world as it was the prototype system. Version 2 was distributed in 1985 with the SunOS 2 operating system and licensed to many Unix workstation vendors. The NFS V2 specifications changed many times over the 10 year lifespan which created occasional incompatibilities between different NFS implementations. Version 3 kept the same security model as version 2 but incorporated many performance improvements. It was developed during a series of meetings in Boston in July 1992 and was later released as RFC 1813 [57].

2.1.4.3 HPFS

The High Performance File System was created by Microsoft and IBM in 1989 for OS/2 as an better alternative to FAT. It allowed long filenames in mixed case characters (length of 256 characters compared to 11 in FAT) and used the disk space far more efficiently, causing less fragmentation of files [92]. Files were stored on a per-sector basis instead of across multiple-sector clusters and the internal architecture meant that related items were kept close to each other on the disk volume. It also included different timestamps for each file stamp (e.g. created, accessed and modified) in the 64KB of metadata it kept for each file. It had a B-Tree directory structure with the root directory located in the middle of the disk rather than at the beginning yielding faster average access times [48]. IBM offered two types of drivers for the file system but the fastest set meant paying Microsoft for each copy sold. This combined with the long disk check times on crash recovery encouraged IBM to port JFS to OS/2 as a substitute.

2.1.4.4 JFS

In 1990, IBM introduced its Journaled File System to the AIX operating system. It went on to become the first commercially successful journaling file system, remaining the file system for AIX computers for over 10 years. The development of the first implementation of JFS is very closely linked to the memory manager of AIX, a common design for a file system supporting a single, closed-source operating system. In order to support machines with more than one processor and enhance scalability and portability (with a view to running on other operating systems) work began on a new version of the Journaled File System in 1995. It was first shipped on OS/2 Warp Server for e-business in April 1999, 4 years of designing, coding and testing later. Over the same time period others from the JFS development team worked to port this new JFS source base to AIX. The Enhanced Journaled File System (also known as JFS2) shipped in May 2001 on AIX 5L. The open source community was granted access to a snapshot of the original OS/2 JFS source in December 1999, when work began to port JFS to Linux.

2.1.4.5 VxFS

VxFS is a journaled file system created as a commercial product by Veritas [273] in 1991. With filenames of up to 255 characters in length, VxFS supports a maximum file system size of 1 terrabyte and offers 4 different block sizes. It uses extent-based allocation and attributes (where a fixed contiguous chunk of disk is partitioned into fix-sized blocks), fast file system recovery and access control lists [291].

2.1.4.6 BFS

BFS is a 64-bit journaled file system and the native file system of the Be Operating System, written by Dominic Giampaolo and Cyril Meurillon in 1996. In addition to the usual name-based hierarchical interface for file location, it also indexes extended attributes to enable a query-based interface [111]. When writing a file to disk, it uses "preallocation"', trying to give a file enough room to grow so it can be written in one contiguous chunk to avoid fragmentation. To avoid wasting space with large block sizes, it offers blocks down to 1KB. The main ideas and features behind BFS are very similar to those purported to have been included in WinFS (see §2.1.1.5), however BFS runs only on Unix based operating systems.

2.1.4.7 Fossil

Released at the end of 2002, Fossil was developed by Sean Quinlan, Jim McKie and Russ Cox at Bell Labs to run on their Plan 9 operating system. One of the main features of Fossil is the ability to take snapshots of the whole file system. At set intervals or on command it can take whole snapshots, continuing until the partition has been filled [223]. Snapshots are available to all users (not just administrators), but only allows access to files that the users could originally see in order to stop users looking at another's old files or passwords [173]. Only the non-permanent snapshots can be deleted, as the permanent ones are stored using Venti [222] (a network user space daemon also by Bell Labs).

2.1.4.8 ZFS

Designed and implemented by a team at Sun Microsystems lead by Jeff Bonwick [46], ZFS originally stood for "Zettabyte File System" in reference to the storage capacity. ZFS is designed for use on the Solaris operating system and heralded as a limitless file system, or rather one whose capacity limits could never be reached. It can accommodate 16 billion billion times the capacity of current 64-bit file systems, and Bonwick is quoted as saying "Populating 128-bit file systems would exceed the quantum limits of earth-based storage. You couldn't fill a 128-bit storage pool without boiling the oceans." [130]. When the contents of a file is changed, ZFS writes the data blocks that have been changed to a different location than the original. The metadata referring to the file is then updated to show where the new versions of the blocks can be found. If the metadata is never written (perhaps due to an error) then the old version of the file will still be recoverable as the original blocks will have not been overwritten. This enables ZFS to take snapshots in O(1) time instead of O(n) time.

2.2 Architectures

As seen above, the history and development of file systems has lead to many different approaches in oder to solve mostly similar problems. Whilst it might not be possible to say which approach is "best" or the most correct, there certain features in several file systems which have presented interesting ways of thinking with respect to file system design. In order to understand more fully the issues faced by current file systems, it is useful to take a closer look at the more important architectures that have been presented throughout history.

2.2.1 FAT

The three versions of FAT (12, 16 and 32) all differ slightly in implementation as would be expected over such a long development period, but are all based on the same model, the use of the File Allocation Table. In a FAT file system, the first thing on the disk is the boot sector which contains important information on the file system as well as the boot code. The FAT itself comes after this boot sector, and keeps track of the disk space (it performs the same job as the i-node table and free list in Unix). In some systems the FAT is replicated so the file system is still accessible should the original become damaged. Following the FAT is the root directory, and then the file data.

The FAT contains one entry per block on the disk. (The size of the blocks is defined in the boot sector, between 1 and 8 sectors, depending on the size of the disk) Originally each FAT entry was 12 bits in the first version, which means a maximum file system size of 4096 blocks. In FAT16 16 bits were used per entry allowing 64K blocks (per partition). FAT32 only currently uses 28 of the 32 bits so technically it could be called FAT28, but powers of two sound better. There is a one-to-one

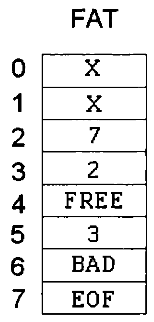


Figure 2.1: An example of FAT

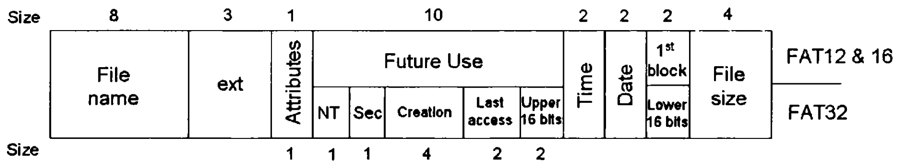


Figure 2.2: Old and new FAT directory entry structure

relationship between entries in the FAT and then number of disk blocks, except for the first two entries which contain the disk class information.

In Figure 2.1 the FAT entry for 5 is 3, meaning that the next block of this file is found in block 3. Block 3 has a value of 2, meaning block 2 comes next, followed by 7. Block 7 has a special marker to denote the end of the file. The directory entry for each file gives the starting block of the FAT and by chaining through the rest of the FAT entries all the blocks are found. Another special code is used to mark free blocks, so when a file grows the system will look for a free space in the FAT and assign that block to a file. This causes fragmentation of files, with blocks belonging to the same file being very far apart on the disk.

The directory entry size for MS DOS has always been 32 bits in length. This was perfectly adequate for FAT12 and 16, but extra additions to the functionality of the file system in FAT32 meant that some internal restructuring was necessary. The two different versions of the directory entry can be seen in Figure 2.2.

In FAT12 and 16, only short filenames were permitted (up to 8 characters and a 3-letter extension). This comprised the first 11 positions of the directory entry. Next came an attribute byte which contained the following bits: A - archive, D - directory entry, V - volume label entry, S - system file, H - hidden file and R - read only. The following 10 bytes were reserved for future use (and came in very useful in FAT32), then 4 bytes for the time and date of last modification, 2 bytes for the starting block of the file and 4 bytes for the total file size.

In FAT32 long file names were introduced, but in order to keep the directory entries compatible with FAT12 and 16, instead of a complete restructure, the reserved 10 bytes were given a use. The rest of the directory entry structure remained identical. The NT field exists solely for compatibility with NT to display file names in the correct case (DOS names were all upper-case). The Sec field adds additional accuracy (to 10 msec) to the creation time field. The last access field contains the date (but not the time) of the last access, then the final reserved 2 bytes are used to store the first 16-bits of the file starting block (as in FAT32 block numbers are 32 bits, the extra 16 are needed in addition to the original starting block field to fit the whole number in).

2.2.2 HFS

HFS is implemented in a different way to most file systems of a similar time. The fact that Macintosh computers maintained more metadata is reflected in the file system design. Each volume on the disk is divided into logical blocks of 512 bytes. The logical blocks are grouped together into allocation blocks (which each contain $\geq$ 1 logical blocks). There are five structures within an HFS volume: the boot blocks, master directory block, volume bitmap, extent overflow file and catalog file. The boot blocks contain all the system startup information, and the master directory block holds information about the volume, such as the date and time of creation, block size and total number of files. The volume bitmap keeps track of which blocks have been used for files, there is one bit for each block on the disk set to 1 if a block is in use, 0 if not. The two main data structures in an HFS volume are the catalog file and the extent overflow file.

The catalog file is a B* tree which contains the records for the files and directories in the volume. There are four types of records: directory records, directory threads, file records and file threads, with each item being uniquely referenced by a Catalog Node ID (or CNID). The file thread contains the name of the file and the catalog node ID of the parent directory and the file record holds the metadata of the file. Files on Macintosh computers are comprised of two parts (or forks), the data fork is the stream of bytes containing the file data, whilst the resource fork contains file metadata. Unlike other file systems with a graphic user interface, the record holds data on how to display the file's icon properly in the interface, as well as the directory path and the resource fork, which contains the first 3 extent records (contiguous blocks of data) for the file. Similarly, the directory thread holds the name of the directory and the CNID of the parent directory and the directory record holds the directory metadata [47]. The catalog file holds all the records for every file and directory in one giant structure, not on a per-directory basis seen in file systems like FAT and FFS. Thus the hierarchy of the file system is stored implicitly in the catalog file, as each file thread contains the parent directory the file.

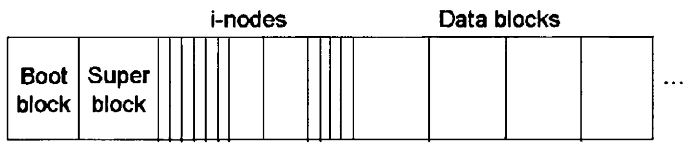


Figure 2.3: UFS disk structure

The extent overflow file keeps track of all the sections of files that are not stored in contiguous sections on the disk. The first 3 extents of a file can be stored in the catalog file record, so the extent overflow file only needs to be used when the file uses more than 3 extents. It is also a B*-tree structure like the catalog file but the contents are much simpler. Each extent record is comprised of three extent descriptors which each encode an extent's starting block and length into a 32-bit number. Later versions gave the extent overflow file the capability to record locations of bad blocks so they were not written to.

2.2.3 FFS

The Berkeley Fast File System was created out of a need to have a higher bandwidth file system than the one originally used on Unix. The old file system divided a disk into ≥ 1 partitions. Each partition could contain a file system the structure of which can be seen in Figure 2.3, but a file system could not span > 1 partition. The file system was described by the superblock, which contained the number of data blocks, the count of the maximum number of files and a pointer to the 'free list' of free blocks. In FFS the superblock is duplicated for safety. The superblock succeeds the boot block on this disk, but some Unix implementations do not use a boot block so it is left as empty space. Next on the disk comes the i-nodes (standing for index-nodes), numbered from 1 to the maximum, with 1 i-node per file. Each i-node is 64 bytes long and contains accounting information for each file and enough information to locate all the blocks. The file system treats directories and files in a similar manner, with directories being files that point to more files. After the i-nodes come the data blocks where all the files and directories are stored. The i-nodes are kept in a separate location to the file data, causing a long seeking time. Files that were stored in the same directory did not share consecutive i-node records, meaning that lots of disk accesses were required to view all the files in one directory.

The directory structure consists of 4 fixed-length fields and one variable length field, seen in Figure 2.4. First is the i-node number (2 bytes), followed by the entry size (in bytes) so the next entry can be located after the variable-length field. The type field denotes whether this directory entry refers to a file or directory. Finally comes the length of file filename and then the filename itself, terminated by a 0 and padded

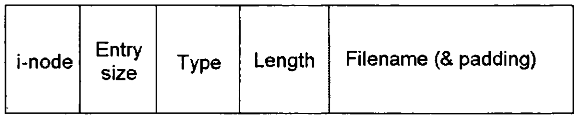


Figure 2.4: Directory entry structure for UFS

to the next 32-bit boundary. As these directory records are searched through in a linear fashion, it can take a long time to find an entry near the end of a large directory. FFS uses caching to improve file entry search performance, if a file name is found in the cache then it does not need to search through all the directory entries.

In FFS, the basic block size increased from 512 to 1024 bytes, and the superblock is replicated for safety. Each partition is divided into areas called cylinder groups that are made of ≥ 1 consecutive cylinders on the disk.

The format of i-nodes differs between systems, but they usually contain: the file type and the 9 rwx permission bits, the number of links to the file (e.g. how many directory entries point to it), the owner's identity and group, the file length in bytes, timestamps of the file's last read, write and i-node modification time plus thirteen disk addresses. The first 10 of these addresses point to data blocks for the file. As block size is 1024 bytes, files of up to 10,240 bytes can be handled in this way. For larger files, address 11 points to a disk block that contains 256 disk addresses. This means files of up to 10,240-(256*1024) = 272,384 bytes are stored in this way. The same technique is used for handling larger files, by utilising the 12th and 13th addresses as pointers to 256 disk addresses each. This means that theoretically, the maximum number of blocks a single file could occupy is 16843018 and the size would be 17,247,250,432 bytes. However, file pointers are only 32 bits in length so the maximum file size is set as 4,294,967,295 bytes.

The bookkeeping information is put on the disk at a varying offset for each cylinder group so the destruction of 1 platter does not destroy everything. The data is laid out so that larger blocks are transferred in a single disk transaction, increasing the throughput. To store smaller files efficiently with these large block sizes the blocks can be broken down into fragments (2, 4 or 8). It uses two different file system layout policies. One is global and uses the system-wide summary information to decide the placement of new i-nodes and data blocks. Local allocation routines use a local scheme to lay out data blocks. Free blocks are kept in a linked list and then used when required which originally caused blocks for each file to be spread randomly around the disk. By using cylinder groups, FFS attempts to keep file data and i-nodes for each file on the same cylinder, meaning faster access times. It also added support for long file names, file locking, symbolic links, file renaming and quotas.

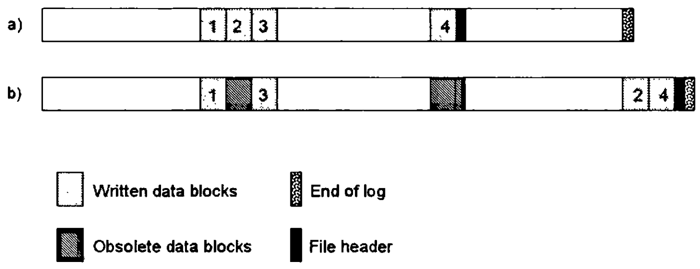


Figure 2.5: Updating blocks in LFS

2.2.4 LFS

Whilst the speed of computer chips and the size of memories have rapidly expanded, the time for disk accesses is still slow in comparison and creates a bottleneck in some file systems. Researchers at Berkeley reimplemented the Unix file system in order to use the full bandwidth of the disk and thus created the Log-structured File System. The entire disk is structured as a log, and all pending write operations to the disk are buffered in memory and periodically written to the end of the disk in a contiguous segment. A single segment could therefore contain file data, i-nodes and directory blocks all mixed together. Each segment begins with a summary denoting what an be found where. Obviously, finding i-node is more difficult than under the usual Unix FS as they are scattered about and not kept in a fixed position on the disk. To combat this problem, an i-node table is maintained and entries indexed by i-number. This map is stored on the disk, but also cached so that the most frequently accessed entries will already be loaded into memory. Most of the on-disk format of LFS is the same as in FFS, with the indirect block, i-node and directory formats being identical.

As disks are not infinite in size, at some point it will not be possible to add any more writes to the end of the log. Some parts of the log will contains parts of files or i-nodes that are no longer in use (for example if the file has been updated or deleted), so LFS has a cleaner process that regularly scans and compacts the log. The cleaner examines the log one segment at a time, and when it finds parts of the disk that are no longer in use, it discards them and moves the remaining entries in the segment to the buffer to be written back to a new segment, seen in Figure 2.5. It then marks the original segment as free so it can be used for new data. In this way, the disk acts like a large circular buffer. Bookkeeping tasks are more difficult than with the other FS, for example when a file block is written back to disk its i-node must be found, updated and put in the write buffer to be written in the next segment, plus the corresponding entry in the i-node map must be updated.

2.3 Issues with File Systems

Despite great advances in computer hardware and software technologies, the methods of storing and retrieving files have remained essentially the same since the introduction of the file concept. As is to be expected when the same system is used for an extended period of time without any major technological leaps or complete redesigns, old problems which plagued previous file system designs are now resurfacing for current file system technologies. The ways in which people use computers are also constantly evolving, so new challenges and issues are also now being presented.

2.3.1 Hierarchical File Systems

Hierarchical file systems have been with us since the 1970's, but since then the average amount of disk space per user has greatly increased, as well as the number of relatively naïve users. Having to file documents into a hierarchy is too constraining, as documented by many sources, [88, 104, 229], and meanwhile users do not want the extra hassle of categorising their files.

There are two major problems associated with hierarchical file systems: [88] one is that documents often need to be stored or classified in a variety of different ways, the other being that folders are used for storing files but are also utilised by the system for management tasks such as sharing. Most users do not well understand the concept of a file hierarchy [207], are bad at choosing meaningful file names which they then have difficulty remembering and searching for [155]. Users have no need to know how or where their files are stored, yet this task is currently left as their responsibility.

Folders do not only organise documents, they can also obscure them, there are too many different hierarchies for different applications and they have a basic limitation of inflexible and strict structure [145]. Users were found to be "squeezing" additional information into the hierarchy which would have been better represented in a different form.

A study [85] into the file system requirements of large-scale applications (such as video servers, electronic libraries, database mining and input/output intensive parallel programs) showed that hierarchical-based file systems are not flexible enough (either in architecture or functionality) to support large volumes of data access.

2.3.2 Relational File Systems

By using a relational database, Marsden and Cairns [180] proposed a flexible file browsing system for novice users. They also highlighted the need for users to be able to attach their own metadata to files and to save the results of search queries for reuse.

In the Semantic File System [112] the file store was replaced by a relational database, which the user can manipulate to create virtual directories and then explore. However, the users still needed to understand the concept of a hierarchy. Gopal and Manber extended the ideas of the Semantic File System to include group-related material [116]. Directories are populated with links to information gathered from previous queries that may be of interest. The user can narrow down the search space by adding or removing links to the semantic folders.

Quan et al. [221] built a file system which allowed users to attach metadata to files (as well as their own attributes), so the files can be classified in multiple ways. The system then visualises the created hierarchy (and not the logical organisation the files on disk) according to the user-assigned attributes. Similar approaches to this solution have been proposed [146], however these systems rely on users assigning the metadata manually. In user studies of personal filing [239], it was found that not unexpectedly, users are not likely to invest the necessary time to assign metadata to files.

Storing files with respect to one attribute can be seen to be a good idea. Studies of email storage [305] show that email messages are often stored in a single list, and then sorted by a particular attribute for a required task (e.g. sort by date in order to retrieve last week's email).

It is not important to the user how files are stored, but how they are retrieved and viewed. BFS [111] is an example of a file system that stores files in one way (using a relational database) but presents them to the user in a hierarchy.

MyLifeBits [109] is a project to fulfill the vision of Memex, presented by Bush in 1945 [56], "a device in which an individual stores all his books, records and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility". It is a system for organising multimedia files and eliminating the need for paper based memories such as photographs and letters etc., and indexes each file in a database in order to arrange them into collections without the usual hierarchy of file systems.

Van Zwol and Apers [290] presented a Webspace method for modeling large collections of related documents by using database technology and the integration of the Webspace method with context-based file retrieval. The Webspace method was implemented in Mirror [296], an information retrieval model that allows users to specify information in terms of keywords, then uses relevance feedback to calculate the relevance of various multimedia types. The Webspace model distinguishes two different levels: document (where the file is specified in XML) and semantic (the concepts derived from documents are modeled in terms of an object-oriented schema).

2.3.3 Distributed File Systems

Distributed File Systems allow users to share data and storage resources by using a common file system, which is usually implemented as a part of the operating system [168]. Ideally, a DFS should look like a conventional file system, with the dispersion of servers and file storage devices being transparent to the users [270] (for example the file abstraction in the Roe File System [97]). One advantage of a distributed file system is that the responsibility for maintenance and upkeep of the system is transfered to those in charge of the network, meaning that important tasks such as back ups and archiving [225] are (hopefully) more likely to occur.

The Sprite File System [204] used large caches on diskless workstations for access to shared data. The files were kept consistent by a simple algorithm, as at the time collaborative write privileges were rare (which is obviously not the case anymore).

One of the major problems faced by distributed file systems is that of file naming. There are three main approaches to solving this problem [21]. The simplest approach guarantees a unique system-wide name by combining the hostname with the local file name, as used in Ibis [283]. The second approach is to mount the remote directories to their user's local namespaces, as popularised by Sun's NFS [246], and the third uses a single global naming structure, variations of which can be seen in Andrew [249], Sprite [300], and DOMAIN [164].

Sprite, AFS, NFS and Spring all make extensive use of caching in order to reduce network traffic and improve perceived reaction speed. Of these, only Sprite [203] has a distributed shared memory and isn't built on top of a monolithic operating system.

2.3.4 Other File Systems

Vesta is a parallel file system designed to run on Vulcan, a massively parallel prototype machine built at IBM [72]. It partitions files into multiple disjoint sequences that can be accessed in parallel [73].

Hess and Campbell proposed a context-aware file system [131], in which data and applications are not tied to a machine, but can be mapped to a physical space based on resource availability or the role of that space (such as a conference room or classroom). In this system, file storage is implicitly linked to the user and "follows" him around as a personal data cloud, becoming available to applications when he enters a new space.

2.3.5 User Interfaces

The interface designers of Windows '95 realised that most users did not understand the concept or the architecture of a file system hierarchy after carrying out a user study [268]. Instead of redesigning the file explorer interface, they added in features such as "My Documents" and assigning default folders for applications to use. However, this approach, although improving the situation still continues to confuse users [43].

FlexiView [251] is a text-based system for viewing file storage hierarchies, whilst Card et al. [59] explored different graphical visualisation methods such as treemaps, cone trees and hyperbolic trees. Whilst both methods can produce good visualisations, the both rely on the incorrect assumption that the underlying structure (e.g. the hierarchy) is meaningful to the user.

2.3.6 Security

In the Unix file systems, each file has a bit map associated with it to describe who can access a file [289]. This map has three rwx field for controlling the Reading, Writing and eXecution of a file by the owner, the owner's group and everybody else respectively. So rwxr-x--x means that the owner has complete access for this file, the owner's group can read and execute the file, whilst strangers can execute only (they would be unable to 'steal' a copy of the file as they do not have read access). Users are assigned to groups by a system administrator, called a superuser. The superuser can also override security and access any file.

The file security implemented in NTFS is considerably more complex than that found in Unix-based systems [265]. When the user logs in, their initial process is assigned a token by the operating system. This token contains the user's SID (security ID), a list of security groups that the user belongs to, and any special privileges. The access token puts all the security information in one easy-to-find location. When an object (e.g. a file) is created, it can have a parameter called a security descriptor which contains a list of entries, referred to as an access control list. Each entry permits or prohibits a set of actions on the object by a SID or group. When a process tries to perform an action on the given object using the handle which was returned from the opening, the security manager gets the process' access token and then looks down the access control list. When it finds an entry matching the caller's SID or the caller's group, it takes the specified access as definitive and looks no further down the list. Entries denying access are placed higher up the list so a user specifically denied access to a file could not gain access to it by being a member of a permitted group appearing further down the list.

Kuo and Lin [161] presented design concepts of secure network computing and a secure RPC framework in order to facilitate the basic needs of Internet users.

As seen in [67] the only way of keeping accesses to a database private is to download sections of the database as a duplicate and use private querying. This highlights the advantages of keeping knowledge bases on the client computers.

Bruce Schneider a vocal advocate of cryptography and user privacy, saying that "Cryptography is a technological equaliser" [253]. However, whilst cryptographic techniques are now widely available to all computer users, they are still not often deployed due to the extra effort required to administer and maintain them.

Blaze implemented the Cryptographic File System for Unix, with the encryption services being the responsibility of the file system [40]. It followed the principles of rational key management (the user should only have to enter a key once a session), transparent access (the user should not be able to tell whether or not a file is encrypted, as the only difference will be that encrypted files are nonsense without the correct key), transparent performance (the en/decryption process should not add much time to the storage process) and concurrent access (several users having access to the same encrypted file simultaneously).

2.4 The Internet

The Internet represents one of the most successful examples of the benefits resulting from sustained investment and commitment to technological research and development [165]. From humble beginnings, it is now used by hundreds of millions of people on a daily basis [6] for a wide range of purposes: from entertainment and communication to business and research. Of course, the Internet did not just appear overnight, but is the result of many years of work from researchers and scientists worldwide.

2.4.1 A Brief History

The simple idea of networked computing can be traced back to America in the mid-sixties. A small but dedicated group of telecommunications and computer researchers combined efforts and developed the ideas necessary to make networked computing a reality. Working with Thomas Merrill at MIT in 1965, Lawrence G. Roberts explored the possibility of getting computers to talk to each other. Together, they created the first ever WAN by connecting the TX-2 computer in Massachusetts to the Q-32 computer in California, with low speed dial up [237]. Evolving into the ARPANET (funded by the American military), by August 1972 there were 29 nodes connected to this network in universities and research organisations across America. Electronic mail was invented later in the year by researchers at Bolt Beranek and Newman (BBN) as a way for ARPANET developers to coordinate their implementation efforts [303].

Two years later in 1974, Telnet, (a commercial version of ARPANET) was opened as the first public packet data service. This year saw the first use of the term "Internet", by Cerf and Khan in a paper on Transmission Control Protocol [61]. More computer scientists wanted access to this growing network [129], but only people linked in some way to ARPA were allowed. This caused the National Science Foundation to sponsor a civilian network in 1980 called the CSNET. Other permanent computer networks were also beginning to surface in other parts of the world. In 1982 EUnet (European Unix Network) was created by EUUG to provide E-mail and Usenet services. The original connections went between the Netherlands, Denmark, Sweden, and UK. In 1983 the Department of Defense split the ARPANET into two, MILNET for military use (like the original ARPANET) and ARPANET was kept for civilian research. Meanwhile, in the UK, JANET (Joint Academic Network) was implemented.

For years, these networks grew and flourished, whilst researchers continued to implement new technologies allowing the wide area computer network to develop into a more common occurrence. However, networks were no longer just the domain of military and educational institutions: in 1991 the Commercial Internet eXchange (CIX) Association Inc was formed after the NSF lifted its restrictions on the commercial use of the Internet [8].

In 1992, the term "Surfing the Internet" was coined by Jean Armour Polly, as the number of hosts broke 1 million and 4000 news groups existed. The estimated growth rate of the Internet in 1993 was 341,634%, with 2 million hosts and 600 www sites. In 1994 there were over 3 million hosts, 10000 www sites, and 10000 news groups, as the Internet celebrated its 25th anniversary. Pizza Hut opened its online pizza ordering service, and Virtual First became the first online bank. The World Wide Web overtook telnet as the 2nd most popular online line service (1st was easily ftp/data transfer). By this time the Internet had developed into the system which users today are familiar with, although by comparison it did not yet have such a wide sociological and ubiquitous reach. The Internet has by no means finished its evolution. As modern computing continues to advance, so too will the Internet in order to keep up with the latest developments.

2.4.2 Addressing and Naming

The way in which computers are identified on a network can be compared to the postal system. If you wish to send a letter to someone, you must put the correct address on the envelope. On a computer network, every computer has a unique address in order to identify it to the other computers and network services. Obviously such a system was not developed overnight, but grew from solutions to the problems presented by the beginnings of networked computing. Back in 1965, when the first WAN was implemented, there were only two computers on the network so identification was really not too technical a problem. However, as more computers came online, the need for a unique naming system became increasingly apparent.

2.4.2.1 Network Control Protocol

As more computers were connected to the ARPANET the Network Working Group (NWG) created the first host-to-host protocol, named the Network Control Protocol (NCP) in late 1970. The lower levels of the OSI network layer model were provided by the IMPs (Interface Message Processors), so the NCP provided the transport layer, defining procedures to transmit a unidirectional, flow controlled data stream between two hosts and also the procedure for establishing a bidirectional pair of such streams between a pair of host processes. On January 1st, 1983, NCP was replaced on the ARPANET by TCP/IP, which became the core network protocol suite.

2.4.2.2 TCP/IP

In 1974 Bob Kahn and Vint Cerf (of Stanford) specified the Transmission Control Protocol (TCP), of which the first written version was presented to the International Network Working Group (INWG). TCP was to provide all the transport and forwarding services on the Internet, to allow diverse computer networks to interconnect and communicate with each other. Before this, ARPANET had been using NCP, which only allowed communication between hosts on the same network. TCP comprises part of the transport layer of the OSI network model to regulate network traffic, whilst IP is found in the network layer and is used to handle addressing. The US Department of Defense studied the use of TCP/IP and decided it should be required for use on ARPANET [311], so funded a research project in order to implement it.

The first structure of IP addresses specified in RFC 675 [60] had 8 bits for the identification of a network, and 24 bits for the identification of a computer, which allowed up to 256 networks, each with up to 16,777,216 unique network addresses. Thus IP addresses have 32-bits of address space, allowing 4,294,967,996 individual addresses. At the time of specification, this appeared to be more than enough for every single computer in the world.

However, the explosion in personal computing meant that computers were no longer limited to military or academic institutes. The huge increase in the size of the Internet meant that this original system could no longer cope and so a new hierarchical model of routing was developed: The Interior Gateway Protocol was used inside regions of the Internet and the External Gateway Protocol was used for gateways between networks with different architectures (which was used for linking different regions).

In the mid 90's, with the rapid rate of growth of the Internet, the number of free IP addresses were dramatically reduced. The three different classes of address blocks assigned to organisations meant that the use of existing IP addresses became inefficient. Address blocks come in three different sizes, class A, B and C, that contain 16 million, 65,536 and 256 IP addresses each. Organisations requiring over 256 addresses would be assigned a class B address, even if they only required 257. Although it is possible to overcome this problem using Network Address Translation, a new system of IP addressing was needed.

To provide more IP addresses, the architecture had to be changed to include more bits. Instead of the 32-bits found in IPv4, IPv6 contains 128 bits of address space, written in hex. IPv6 [83] began development in 1995 and has since been worked on by a number of organisations to ensure its widespread implementation.

2.4.2.3 Domain Name Servers

Before the Domain Name System (DNS) was invented, network users had to remember the IP addresses of the computers and networks they wanted to access. DNS provided a mapping between the IP addresses and URLs which made access to the growing number of nodes a lot easier as people no longer had to remember the numbers.

The hostnames of all networks used to be kept in a single file, 'hosts' on an FTP server, managed by a single person. This file contained information on the computers attached to the ARPANET including the hostname and IP address. As the number of hosts on the Internet increased, the responsibility of maintaining and distributing the list was given to SRI. Any changes to the contents of the DNS entries would be emailed to SRI to be updated on the main list, and administrators would periodically download an up-to-date version from SRIs FTP servers. This system did not scale well, and soon the load on SRIs FTP servers as well as the effort required to update the list became too high to deal with.

DNS was created by Paul Mockapetris of University of Wisconsin, which allowed packets to be directed to a domain name that the server database could then translate into the corresponding IP address [196, 197]. This system, still in use today distributes DNS information across the Internet instead of keeping all the data on one host machine. Each domain owner maintains their own information on their host, and a central authority keeps records of where this information is kept.

2.4.2.4 The Internet as a File Storage System

Logically, with each computer connected to the Internet having a unique address, it is possible for each computer to be used as a network file store. However, it is obvious that users wish to have some control over where and how their data is stored. With the emergence of the World Wide Web, many users created personal websites, storing these pages on the servers of their Internet Service Providers. Virtually all webmail accounts now provide an amount of online storage space for both emails and the attachments contained within. Slowly, the Internet has evolved into a hybrid of an information service and a file system. Protocols such as WebDav [304] and programs such as Microsoft's Windows/Internet Explorer [268] have attempted to give the Internet the look and feel of a local computer file system.

2.4.3 Interface Development

Obviously, as the Internet grew in size, its appearance also changed significantly. Primarily the domain of a small subset of academic and military researchers, little thought was given to its usability or the appearance, the more important requirement obviously being that as a network and proof of concept, it was functional. Thankfully, as the number of users of the network increased, so to did the efforts toward fashioning a more usable interface.

2.4.3.1 Early Internet Interfaces

As with almost all computer programs, the first user interfaces to the Internet consisted only of a command line. Whilst only a limited set of commands had been implemented, a more complex interface was not really required. However, as the popularity of the Internet began to increase, users began to demand a more usable interface. A first attempt at a user-friendly interface to the World Wide Web was established when Paul Lindner and Mark P McCahill from University of Minnesota released Gopher [16], a text based menu driven interface to access Internet resources. Although a slight improvement on previous techniques, it was still required that the user remembered or knew how to use the complex user interface.

The hypertext browser Lynx [2] was created independently of the Internet, primarily to distribute campus information across the University of Kansas. When an Internet interface was added by a student in 1993, Lynx became the preferred web browser for text only terminals and is still in use today.

2.4.3.2 Web Browsers

In 1990, the first web browser "WorldWideWeb" (www) was released by Robert Cailliau and Tim Berners-Lee of CERN [37]. It was originally developed as a means to provide a distributed hypermedia system and easy access to any form of information anywhere in the world. Although it was non-graphical (which came later in Mosaic in 1993), it still revolutionised modern communications [137] and from this point onwards, development of web browsers has been inseparably entwined with the development of the Internet.

NCSA Mosaic [15] took the online world by storm in 1993. As one of the first graphical interface browsers, people without computer expertise were able to just point and click to navigate the World Wide Web. It originally ran on Unix but was soon ported to Macintosh and Windows platforms. The creator of Mosaic, Marc Andreessen formed a company that would later be called Netscape Communications Corporation.

October 1994 saw the release of Netscape Navigator and it was also at this point that Microsoft marketed its Internet Explorer, effectively starting the browser wars between the two companies. Both continued to develop their browsers and insert proprietary extensions into their products, which eventually lead to Microsoft gaining the largest share of the market, and Netscape open sourcing their browser, named Mozilla.

The continual development and distribution of Internet Explorer with Microsoft Windows has meant there are strong ties between Internet Explorer and Windows Explorer - the latter being used as a graphical interface for exploring the local file system. Both the interface and functionality are very similar, blurring the lines between the local file system and the Internet.

2.4.4 Searching and Locating Files

With the improvements in interface technologies, more users without specialist knowledge were able to access the Internet, but previous to when search engines becoming widespread people had only limited ways of searching for information they were interested in if they did not know the exact location. The concept that you had to know exactly what you were looking for in order to find it seems absurd by today's standards, yet at the birth of the Internet there were so few resources online no more sophisticated mechanisms were needed.

When the Internet was first developed as the ARPANET, the number of hosts could be counted on two hands. As they were only military or academic institutions, only a very specific subset of information or communications were shared between the different sites. This, combined with the lack of information published over the network meant that no advanced techniques were necessary for locating the information a user was looking for. However, the explosion in the use of the Internet that came about when graphical interface browsers were introduced soon changed this situation.

2.4.4.1 Search Engine Development

The first search engine, named "Archie" [98] ("archive" without the "v") was created in 1990, simply downloaded the directory listings of every anonymous FTP server, creating a searchable database of the filenames available. Similarly Gopher (created in 1991) indexed plain text documents instead of the directory listings. Veronic and Jughead were two programs that both queried the Gopher listings. A new mechanism for indexing and accessing information on the Internet was provided by WAIS (Wide Area Information Servers), which in turn allowed powerful search techniques to be implemented such as the keyword search.

The first web crawler was called the "World Wide Web Wanderer" and was created by Mathew Gray at MIT in 1993. The main purpose of the crawler was to keep track of the size of the Internet, and so the information it stored in the "Wandex" was not overly detailed. The first full-text indexing crawler search engine was WebCrawler that introduced the ability to search for any word on any web page. Since its release in 1994, this functionality has now become the standard of most search engines.

Most modern day search engines use similar mechanisms for finding and indexing the data found on the Internet. Large amounts of information is gathered from webpages using small programs called webcrawlers or spiders. These crawlers consist simply of a very basic automated web browser which follows every hyperlink it encounters. As it traverses the Internet, it analyses each page and gathers data on the title, headings and body content in order to decide how the page should be indexed. The results are stored in a database which is then queried when a user inputs a search term. Depending on the internal criteria of the specific search engine, the best matches for that search term are returned from the index, along with a short summary of the page it was found on. Obviously, the more data a search engine has stored in its index, the more likely it will be to provide an accurate and comprehensive set of results.

The most successful search engine to date is Google [50]. Released in 1998, it took a few years for this simple but powerful search engine to rise to prominence. A large part of the success behind Google is the system called PageRank [51], where the number of pages linked to from other sites is taken into account when sorting the results for relevance. However, PageRank is just a small part of the criteria used to determine relevancy of webpages.

As well as an Internet search engine, Google now also provides a desktop search engine [1] for users to locate their locally stored files. Most users would now find it impossible to find anything on the Internet without the aid of a search engine, not only due to the unimaginable amount of data available, but also because users have naturally become reliant on search engines as the solution to the data location problem. The continual blurring of the line between the Internet and personal computers once again highlights the need for a redesign of traditional file systems.

2.4.5 Security and Growth

The evolution of the Internet and ubiquitous network computing has given rise to some interesting problems in the field of information security. As it is now unusual for a computer to not be connected in some way to a network of some description, it must be assumed that all computers are possible targets for some kind of electronic attack, in much the same way that all houses are liable to be broken into by a burglar. Obviously the techniques of firewalls, passwords and encryption go some way toward protecting users of the Internet from the unwanted attentions of a hacker or cracker, but effective security remains the domain of experienced and knowledgeable users only. Not only do electronic attacks such as viruses and trojans create some kind of disturbance in everyday computer usage (such as causing a malfunction or unexpected behaviour in a computer), as data and information contained in files becomes more valuable there is the threat of losing or compromising the security of such data.

If users currently did wish to store their files on the Internet, to a certain extent they relinquish the control of those files to whomever the relevant server belongs to. As a computer user, the author would certainly not wish to store files in a publicly accessible place without some kind of reassurance that the files would only be distributed to the legitimate parties and would remain completely secure and hidden from anyone else. As the Internet is not policed (in the traditional sense) or organised by any one central organisation (indeed, it is likely the Internet would not have been such a success if this were the case), attempts at security are rather ad-hoc and the responsibility of the individual user.

If users were to use the Internet as a file system without any additional mechanisms to support file storage, the problem of file location and retrieval would only worsen. It would become even more difficult for users to find their files or to perform administrative tasks such as archiving, backup or removal. If users were presented with an infinite file system, although they would never have trouble storing their files, the other problems associated with current file systems would still remain.

2.4.6 Current Uses

At the end of the last century, the majority of Internet usage was related to interpersonal communication [160, 132]. (This could be why there are so many easy-to-use different email systems, instant messengers and chat programs but not many document management functions.) The unsuitability of existing web technology for effective document management has long been acknowledged. A study [227] of the less-than successful document management software used by Xerox highlighted the need for extra tools in order to utilise the infrastructure of the Internet for document management and work group applications.

Adamic and Huberman [11] observed that there are many small elements in the Internet, but few large ones. Their observations enabled them to create mathematical rules for modeling the behaviour of Internet sites, users and the number of hyperlinks by using a power law. There have also been prediction models developed for the likelihood of users following hyperlinks from one page to another [36].

Although the Internet is unstructured, it suffers from several problems [250] such as overloads on network servers, limited ability to control access to sensitive data and the lack of mechanisms for data consistency.

Zhao and Resh [317] explored the transformation of knowledge processes via Internet publishing, a term which refers to the means of distributing publications that has conventionally been done in a paper form. They noted that the Internet has created an unparalleled opportunity for publishers to talk directly with consumers, to fill the needs of user knowledge and to create information products to best service those needs.

Today in business operations there exist four major kinds of web-based systems: Intranets to support internal work; Web-presence sites designed to reach consumers outside of an organisation; Electronic commerce systems to support consumer interaction (such as online shopping); and Extranets, a blend of both internal an external systems for business to business communication [138].

2.4.7 Semantic Web

The next stage in the development of the Internet is believed to be the Semantic Web [38]. It is an extension to the World Wide Web, where data and semantic definitions can be processed by computer programs. More formally, the Semantic Web can be defined as "a web of machine-readable information whose meaning is well defined by standards" [103].

The terms "Semantic Web" and "Web 2.0" are often used interchangeably by those outside scientific and technical fields. However, they have slightly different meanings. Web 2.0 (coined by O'Reilly Media in 2004 [117]) is the name commonly used to refer to the new generation of Web applications, sites and companies that emphasise openness, community and interaction (such as blogs [65], wikis [54]). The number 2.0 does not denote a designed version or discrete evolution, but is used as the new developments are considered by some to be fundamentally different from the original early 90's web [194].

The machine readable information in webpages usually just tells the browser how to render the pages [124]. The idea behind the Semantic Web is that web pages should be written not only for humans to read, but for machines to understand and manipulate [274]. There has been much discussion over how this can be accomplished, with Michael Uschold noting that *"The challenge of developing the Semantic Web is how to put this knowledge into the machine. The manner in which it is done is at the heart of the confusion about the Semantic Web."* [288]. In the Semantic Web, ontologies describe the semantics of data, meaning that machines and programs can more intelligently locate and integrate data for many different types of tasks [86]. The success of the Semantic Web and its applications depends largely on the utilisation and interoperability of well formulated ontology bases in an automated heterogenous environment [281].

2.4.7.1 RDF

The Resource Description Framework (RDF) was designed to standardise the definition and use of metadata [82]. Specified by the World Wide Web Consortium (W3C), and based on existing XML standards, URI and unicode [103], RDF defines how various domains can cross-communicate. By using XML as a transfer syntax, it aims to provide uniform and interoperable means to exchange metadata between programs and across the Web [224]. Each RDF triple is composed of a subject, a property and an object which represents a single fact with well-defined meaning. A triple is the minimum piece of knowledge that can be represented in RDF [118].

In the Semantic Web architecture, the basic layer of data representation is standardised as RDF [215], but this can present various difficulties when trying to layer expressive ontology languages on top of RDF schema [3, 102]. The Semantic Web will have data from many different ontologies and information processing across the ontologies will not be possible without knowing the semantic mappings between them [86].

2.4.7.2 Tagging and Folksonomies

Tagging is associating keywords or "tags" with data objects (websites, pictures, email etc). When trying to find the data, the tags are searched instead of keeping everything organised in a folder hierarchy. Tagging websites have become common on the Internet since 2004 [127] and have the benefits of helping recall and supporting search mechanisms [179]. An example of a successful tagging website is Flickr [4], which lets users upload photos and tag them with metadata to help other users find images they are interested in [194].

There are many different motivations as to why people tag [12], ranging from technical to social. Tags are always personal but not necessarily private [287], a feature for which there is growing support [257]. For example, if one user were to tag an album listing with the name of the artist and the word "owned", that would help one user find all albums (s)he owned but would not be of any use to any other users.

Folksonomies are ontologies that have evolved from community practice [194], collaborative tagging systems where groups of people can define a common vocabulary for a particular domain. They can contain structural knowledge about documents, and assist navigation by providing dynamic hyperlinks among tags, documents and users [313]. Folksonomies can suffer from "meta noise" or idiosyncratic tagging which burdens the user, and users can also change their own vocabularies, meaning tags may no longer match [243]. Another problem is illustrated by a game on the Internet which shows how hard it can be for two people to agree on the same words for a description of an image[295]. Whilst tags are useful for classifying articles into broad categories, they can be less useful in indicating particular content [52], and some users feel that the perceived benefits of annotating files do not overcome the investment [156].

2.4.7.3 Criticism

After the bursting of the dot com bubble, many people are skeptical about the importance of the Semantic Web but many sources continue to herald and sensationalise its arrival, claiming the hype surrounding it is simply a resurgence of interest in applying dismissed previously technologies [287].

Web 2.0 is, in principle, only a collection of other people's proposals and wish lists for the next generation of the Internet. Whilst it is a conceptual framework, it is not a system or technical framework. The term "Web 2.0" can mean radically different things to different people due to the lack of set standards. When asked to define the term, Tim O'Reilly gave the following buzzword-laden and long winded response: *"Web 2.0 is the network as platform, spanning all connected devices; Web 2.0 applications are those that make the most of the intrinsic advantages of that platform: delivering software as a continually-updated service that gets better the more people use it, consuming and remixing data from multiple sources, including individual users, while providing their own data and services in a form that allows remixing by others, creating network effects through an "architecture of participation", and going beyond the page metaphor of Web 1.0 to deliver rich user experiences"* [275].

Meanwhile, Tim Berners-Lee, when asked if the Web 2.0 was all about connecting people replied: *"Web 1.0 was all about connecting people. It was an interactive space, and I think Web 2.0 is of course a piece of jargon, nobody even knows what it means"* [13]. This is a problem compounded by some websites using the term 2.0 for the use of some trivial feature which can cause observers to consider it more an attempt at promotion than providing a useful service. There is also the train of thought that says "Web 2.0" is still just "Web 1.0" as new techniques do not replace protocols such as HTTP but adds an additional layer of abstraction on top of them.

The technologies behind such a system are still relatively young and are certainly not yet widespread [181], the fields of artificial intelligence and knowledge representation still have some way to go before they can operate on a level comparable to a real person attempting the same search and logic tasks that humans find so simple. The Semantic Web will mean assisting users to find relevant data, make appointments, perform complex searches [139] and so on, but it does not address the problems of information overload at all. It would (hopefully) provide more meaningful data and hence allow a user to spend less time searching for correct answers, but at the same time it has not yet been extended to the management of personal documents. Given the expected structure of the Semantic Web, to do so would involve extra workload on users when creating documents.

2.4.8 Discussion

The unstructured nature of the Internet should be considered as one of its strengths as it has resulted in the large scale growth and availability of information available. At the same time whilst this may be a great strength for the dissemination of information, it creates a major weakness for using the Internet as a file system. The Internet as a network spans much of the earth and the problems of supporting a structured or traditional network file system over something of such magnitude are obvious from both social and technological standpoints.

To be able to exist over the Internet, a more flexible approach to file storage is required, however one that is more structured than the World Wide Web itself. Essentially, a large scale flexible file system which employs the previously designed infrastructure and size of the Internet without losing the advantages that this network provides by using a rigid file system.

The current file systems no longer meet the needs of the everyday computer user. The metaphors used for filing systems are outdated and confusing to users whilst the basic hierarchical structure does not have the required flexibility for multiple file classifications. The amount of data users deal with places an extra cognitive load on the user when deciding how to process the data and where to store it, as well as remembering where their previous documents were. This leads to confusion and frustration with the users, and file systems not utilised to their full capacity. The current file classification and search functions for file systems are too slow and time consuming to use and lack complex queries such as similarity matching. Users do not want to spend the extra time and effort inserting metadata to files in order to make these searches more effective. The same can be said of users for back-ups and archiving, the obvious long term benefits still do not encourage the users to invest the necessary time and effort to protect themselves from mistakes (human or computer generated) and save space and time. Although there exist systems for collaborative work environments, they are additional to the file systems and are not transparent. Users have difficulty in setting up encryption on their files and have no desire to memorise the many keys they need to access them. Current systems do not provide adequate, easy-to-use security, organisation and search mechanisms or convenient user interfaces.

The Semantic Web or "Web 2.0" is in principle only a collection of proposals and wish lists for the next generation of the Internet. It is a conceptual framework, but not a system or a technical framework. This strengthens the need for the work presented in this thesis.

The GIFS framework aims to fix these problems by removing the burden of file storage, classification and management. Offering a transparent service for collaborative work, file versioning, metadata assignment and encryption, GIFS removes the burdens associated with the management of increasing amounts of information and data without additional input or effort from the user.

Chapter 3

Information and Knowledge

While this thesis presents a framework of GIFS from mainly a file system perspective, it also encapsulates the essence of information and knowledge processing, which is a fundamental aspect of computer science, and is studied in a broad range of subjects such as artificial intelligence, knowledge based systems, information management, personal assistant, computer supported cooperative work and groupware, and so on. Because of the huge volume of the literatures on these subjects, this chapter is not intended to provide a comprehensive survey of these subjects. Instead, we focus on highlighting the most relevant discussions and developments in these subjects.

Firstly, we present a summary review of the current problems of information overload to illustrate the problems that users currently face in their day-to-day computing lives. This is followed by a brief review of the most relevant developments in the field of artificial intelligence, knowledge base systems and agents technology. We then examine the most relevant developments in personal information management in conjunction with a number of current applications that are available to assist the user in managing their data. Finally we present a discussion of the previous works in comparison to the proposed framework of GIFS.

3.1 Information Overload

Albert Einstein once said "I never waste memory on things that can easily be stored and retrieved from elsewhere" [94]. If only the rest of us could think about information in such a simple and structured way, our computers would no doubt be tidier and more organised. However, in modern day computing, no matter how organised or efficient a user is, there comes a point when the sheer amount of information sent to us via email or the Internet becomes too voluminous to process. If computers are meant to make our lives easier, then why is a considerable portion of most computer operators day devoted to sorting, filing and processing electronic documents?

3.1.1 Overview

In simplest terms, information overload refers to the state in which a user has too much information to either make an informed decision on a topic or to be able to remain informed. There are many reasons that information overload can occur such as a large volume of historical data, contradictions in available information, a lack of mechanisms to compare and process different kinds of information, high rate of new information being released and a low signal-to-noise ratio, all of which make it difficult to identify what is relevant.

The problems of information overload have been well documented by many different sources over the last decade [33, 206, 305, 30, 31], but the term was first used by Alvin Toffler in his book "Future Shock" in 1970 [285]. More recently, the terms "information pollution" and "interruption overload" have surfaced in publications by Jakob Nielsen [208] and the Financial Times [232] respectively. These terms do not just refer to the increasing amount of information available, but also to the huge growth in the breadth of information dissemination. There are several explanations behind the cause of this so-called explosion [159]. These include the development of low-cost computing and storage devices and low-cost Internet access and the availability of easy to use interfaces (such as the development of the web browser). Research into solutions to the problem of information overload is still in its infancy, leaving users without a way to extract the maximum benefit from the Internet. Some researchers have argued that information overload on the Internet is simply an information retrieval problem [198], but with more and more data being added to the Internet every day the techniques for allowing users to search and browse for information should also come under scrutiny.

3.1.2 History

However, this is not the first time society has faced a boom in the amount of information. Over 500 years ago Johannes Guttenberg unleashed an information explosion with his invention of the printing press. No longer was written information the domain of a small intellectual elite. Roughly 400 years later, Alexander Graham Bell ushered in the age of telephony [267]; by 1915 there were more that nine million telephones in use world-wide. Global communication changed from being restricted to a slow medium (e.g. post) to an immediate medium. Toward the end of the Second World War, such was the rate of scientific development and publishing, Vannevar Bush lamented: "The investigator is staggered by the findings and conclusions of thousands of other workers - conclusions which he cannot find time to grasp, much less remember, as they appear" [56]. We can equate the most recent information explosion with those that have gone before: As with the invention of the printing press, the ability to disseminate information has been democratised, the immediacy of information creation and access is similar to that bought about by the invention of telephony, and the huge amounts of information now available to us via the world wide web would be incomparable to the amount that Bush cited [159]. Past experiences have shown that when communication media become easier and cheaper to use, people change their behaviour in order to use such systems more [184].

Looking broadly at the problem of information overload across all available media, it is easy to see why this has become such a problem. Twenty years ago, most houses only had four television channels, a couple of local newspapers, several national newspapers and magazines and around thirty radio stations. Looking for something that interested you from these sources actually took some effort: having to plan a schedule around your favourite TV show, waiting for your favourite monthly magazine to be printed or sifting through books in a library to find the topics that interested you. Today, people have access to more data that does interest them than they can possibly ever consume. Books pile up, inboxes overflow with RSS feeds and emails demanding your attention, digital television boxes keep recording all your favourite shows, and mobile telephones bleep constantly to signal incoming messages. Whilst most of the selectively sorted information will be of interest (but still too unwieldy to digest), there will inevitably also be irrelevant material mixed in with it too, providing users with far more information than they could ever hope to process [206, 314].

3.1.3 Effects

The average disk space on personal computers has been approximately doubling every year, a rate faster than Moore's Law [185]. As the space available to users increases, the existence of larger files becomes possible and electronic file stores become more cumbersome to maintain and manage. A recent study by New Scientist measured the effect that information overload can have upon the mental capacity of humans. During this test, users were asked to perform a set of simple tasks whilst half were bombarded by interruptions such as phone calls and emails. The results of the test showed that information overload can decrease the IQ of a user by around 10 points, which lead to the claim that it was "more harmful than marijuana" [157].

Ten years ago, when a world-wide survey was undertaken by Reuters [297] they discovered that over two thirds of managerial staff suffered increased tension and one third suffered ill-health due to information overload. The term "Information Fatigue Syndrome" [147] was used to describe the social, physical and mental problems seen in the results by psychologist David Lewis. Other effects of information overload can include difficulties with memory, poor decision-making and a significantly reduced attention span [259]. The emergence of these effects are comparable to the health problems caused by a diet too rich in calories. Once upon a time information was scarce and so more of it was seen as a good thing, but now the point of saturation seems to have been reached and users need assistance to limit the ill-effects.

It would be uncommon today to find a computer user who has their files completely organised and can find any given document or piece of information on demand with little effort. As time continues, the problems of information overload look set only to increase, whilst no workable solution has been provided by researchers. The original reason for inventing computers seems to have been forgotten - to make the lives of humans easier by removing the burden of repetitive, simple and boring tasks. Why should it not be possible to apply this thinking to the design of a modern file system? Whilst the idea of an effortless file system may seem like an impossible dream in today's climate of information overload, the technologies required to produce such a system have long existed but have yet to be combined or applied properly to this problem. The GIFS framework sets out to provide a scalable solution to the difficulties of information overload. In order to understand the knowledge-based section of the framework, a overview of the constituent technologies follows.

3.2 Artificial Intelligence

When considering how to alleviate the problems of paper-based information overload in an office environment, secretaries and personal assistants have proved indispensable. Responsible for the file storage needs of other office staff, these employees handle much of the mundane work associated with documents. This scenario can be transferred directly into an electronic environment (especially as most offices run with electric documents now rather than paper ones) and the creation of a personal assistant program to perform the mundane tasks related to file storage. Attempting to produce any kind of machine behaviour to mimic the action of a human counterpart (even if through analogy rather than direct observation) leads us into the jurisdiction of artificial intelligence.

Artificial intelligence is a branch of computer science pertaining to intelligent behaviour, adaptation and learning in machines. However, such is the size and scope of this research area, it could also commonly be classed under the auspices of philosophy, engineering, psychology, biology or mathematics. Humans have long been fascinated by the inner workings of the human brain and classifying exactly what the term "intelligence" actually means. Trying to make machines that can display some kind of intelligent behaviour allows for many different approaches and research areas. Views of researchers in artificial intelligence can be split into two categories, that of strong AI and weak AI [71]. The former refers to the idea that given enough processing power and 'intelligence', a computer program could think, behave and have a consciousness like a human. Many researchers (both computer scientists and philosophers) find this notion to be far fetched and possible only within the realms of science fiction. In comparison, so-called weak AI is the simpler view that computers can use intelligent behaviour to solve complex problems, without being intelligent in the same way as a human.

Despite many advances in the field of artificial intelligence, computers have yet to encompass what we would call "common sense" [166]. Started in 1984, the CYC project is attempting to do just that, by using a giant knowledge base to encode basic human knowledge in order to give computer programs what humans would refer to as rudimentary knowledge about the mechanics of the world. The project is still continuing many years later and remains nowhere near completion [214]. Instead of trying to build one, large, all-encompassing intelligence, other research has suggested that it is preferable to have many, simpler agents each performing a simple task and then combined to provide a more complex answer [195].

The term "AI renaissance" [211] was coined by Daniel O'Leary in 1997, when AI was beginning to play an increasing role in information retrieval strategies. No longer was AI restricted to pure applications, it was now being embedded on other systems for searching, retrieving and analysis of huge amount of data.

For the purposes of this review, we will look further only at the topics within artificial intelligence that are directly relevant to the task and proposed solution in hand as a complete review could fill libraries.

3.2.1 Knowledge Bases

Much of artificial intelligence is in some way concerned with the concept of knowledge. There are the research areas of knowledge representation, knowledge acquisition, knowledge engineering, knowledge bases and knowledge-based systems among others. Defining what can be considered as knowledge is a slightly hazier problem and unfortunately outside the scope of this review. However, a knowledge base can be defined as a special kind of database for the purposes of knowledge management, providing the means for the autonomous collection, organisation and retrieval of that knowledge. Knowledge bases can be split into two major types: machine-readable and human-readable. Machine-readable knowledge bases store their knowledge in a computer parsable form in order that some kind of deductive reasoning can be applied to them. The data collection in such knowledge bases often takes the form of a set of logically consistent rules, meaning that logical operators (such as conjunction and disjunction) can be applied. Employing these techniques, classical deduction may be used to reason about the atomic knowledge instances within a knowledge base. An example of a machine-readable knowledge base was used in a system which built a personal FAQ document [205] by studying the frequency of particular questions asked. By maintaining a knowledge base of rankings denoting the most recently asked and frequently asked questions this prototype system was able to present more personalised content to each user.

Alternatively, human-readable knowledge bases are used primarily for training purposes, providing users with the capability to search and retrieve domain specific knowledge. This kind of knowledge base is used commonly inside large organisations as a way of providing information or solutions on a particular problem to those who may be less experienced in that area. For the purposes of this research, any further references to the term "knowledge base" should be regarded as referring to a machine readable knowledge base only.

In the mid eighties, when knowledge base systems first emerged, there were several key problems. These were: insufficient understanding of the structure of knowledge based systems, the cost of knowledge acquisition, and the focus on complete (but narrow) solutions. Swartout [271] believes that recent advances in key areas (such as a second generation architecture for knowledge-based systems, development of software engineering methodologies, libraries of problem solving methods and ontologies) have gone along way toward addressing these problems.

3.2.2 Data Mining

Knowledge management and IT have long had a symbiotic relationship [210]. As computers essentially process information and data, the internal representations of these data and the manner in which it is retrieved and processed is of great importance. There are several different reasons why people store data [101]: it is becoming easier for them to do so; people store data because they think there are valuable assets contained within it; scientists represent observations about a phenomenon under study; and businesses store data on customers, competitors and markets. These are but a few examples of why people store so much raw data, even when that raw data is of little benefit. This data only becomes useful once it is processed, the science behind which is commonly referred to as data mining [178].

Data mining could be defined as "the nontrivial extraction of implicit, previously unknown and potentially useful information from data" [107] or "the science of extracting useful information from large datasets or databases" [128]. The simplest example of data mining is called the "market basket analysis" [49] which has application in the field of retail and sales. If a shop keeps a record of who buys various different sorts of products from their store, they can identify which customers are likely to be interested in alternative products and target them with advertising accordingly. Some people feel that this is an invasion of privacy, especially in the situation where adware or spyware unknowingly installed on computers track which websites a user visits for marketing purposes. However, others see data mining as a useful tool, particularly in the field of scientific research where it can be used to analyse the enormous datasets that are produced during experimentation.

In recent years, there have been many approaches to personalising content, using software that learns patterns, habits and preferences. By the mid-90's people had begun to realise the potential of data mining large collections of unorganised data, such as found on the Internet [99]. The increasing computational power of computer processors and the size of storage disks are having a positive effect on the accuracy and usefulness of this kind of analysis. Knowledge bases containing data or meta-data (information about data) are particularly suitable, as the data can be stored in a machine-readable format.

3.2.3 Agents

An agent is essentially a computational system which has goals, effectors and sensors, it decides autonomously which actions it should take to improve progress of those goals and has the ability to learn or adapt to improve its effectiveness [176]. Not all of the above criteria need to be present for a program to be considered as a software agent.

The concept of agents is by no means new. Despite this, many of the questions revolving around agent interaction with people and their usage [9] have still yet to be answered [302]. For many years now scientists have studied systems that demonstrate some kind of agent behaviour. Two of the first visionaries were Nicholas Negroponte [202] and Alan Kay [151]. Since their invention it has been suggested and envisaged that agents would be able to reduce the amount of information a user has to deal with [175] and act like a personal assistant.

On an application level, agents can be categorised into four different groups [126]: Buyer or shopping agents, user or personal agents, monitoring or surveillance agents and data mining agents. Buyer agents work in a similar fashion as the data mining scenario presented in the previous subsection. They browse through the Internet, suggesting products or goods that you may be interested in on the basis of what you have previously bought or have in your shopping basket (Amazon [171] is a good example of the uses of a shopping agent). Personal agents can perform a variety of actions on behalf of a user in a range of different situations. For example, a personal agent may check and prioritise your email having been told a set of criteria, fill out webforms automatically and store the information for further use, act as a computer player to patrol an area in a computer game or assemble customised news reports. Monitoring or surveillance agents observe and report on the status of equipment or other factors [247], e.g. monitoring stock levels in a shop in order to effectively schedule ordering and minimising stock wastage. Perhaps predictably, data mining agents operate over a data warehouse or knowledge base and discover information, find patterns or shifts in trends. Of course, these groups are by no means mutually exclusive and it is possible for an agent to be classified under more than one category. For example, an information agent used to help users locate relevant information in large, unstructured collections of documents [24] could be classed as both a data mining agent and a personal agent.

Mentioned in the previous chapter, web crawlers or spiders are independent software agents that crawl the web to gather information, and are commonly used by search engines. The spiders of one such search engine, Lycos [119], have now evolved to a multiagent system of cooperating components that visit over 10,000,000 web pages each day. The Lycos system consists of 3 main components: the spiders, a URL server for managing the future locations for the spiders to visit, and a catalog update server which retrieves the information gathered by the spiders and stores it in a repository.

There have been several attempts to combine artificial intelligence systems with databases. One such attempt was the design of a cache-based DBMS interface that accepted queries and returned tuples in order to provide AI systems with efficient access to databases [187]. Waltz and Kasif [298] presented a framework for memory-based reasoning, which combined the strengths of case-based reasoning and probabilistic reasoning in order to create agents to work in largely autonomous adaptive systems. The idea of a ubiquitous media agent was introduced by Wenyin et al [302]. The purpose of this agent was to gather information on user's actions and multimedia file accesses in order to build up personalised semantic indices of multimedia data. The collection of data in this system does not include metadata that is commonly associated with each file, and looks to manually-created attached text files in order to find semantic information.

3.3 Personal Information Management

Personal information management (PIM) is the umbrella term used to describe the collection, storage, organisation and retrieval of digital objects (e.g. files, addresses and bookmarks) by an individual in their personal computing environment [163]. The distinction between personal information management and general information management is that in the latter a professional organises things for other people (such as a librarian) whilst in the former it is the onus of the individual to manage their own information [34].

PIM is a growing research area [142] especially as researchers sources have identified PIM as a burden to users [177, 305]. In 1995, Barreau classified five component sub-activities of PIM [22]: acquisition, organisation, maintenance, retrieval and presentation. However, Barreau's definition is by no means perfect. Boardman [42] suggested several changes, including the removal of the presentation category as by default, most PIM tools will automatically contain such functionality. This disagreement over categorisations is not uncommon [306] and many similar (but varied) definitions are in use by different members of the research community.

The activity of organisation (or categorisation) has been identified as a cognitively hard operation for users [163], which may go some way to explaining why previous studies have identified that users do not like to spend time categorising their data [114, 90]. Improving PIM would result in better uses of resources such as time, money, attention and energy (or in the case of a commercial organisation, better employee productivity) [35]. Thus much of the research into PIM focuses on making categorisation easier [89, 143, 148, 221, 240]. In comparison, the GIFS framework removes this burden from the users by taking responsibility for both file classifications and storage.

There have been many user studies into the different application areas of PIM such as email [19, 305], bookmarks [144, 10], photographs [239] and general files [312, 22, 23]. Different researchers have employed a variety of techniques [153] when evaluating PIM technologies and studying user behaviour. A cross-tool study of personal information management by Boardman and Sasse [43] found that users do not display the same behaviour when organising different digital formats. It was not possible to define users as displaying only one behavioural characteristic for personal information management across different domains as where different tools are used for each organisational activity, the user behaviour also changed. Kaye et al [152] also had difficulty in identifying standard behaviour in users in cross-domain personal information management. The number of different tools available can also result in lack of full utilisation by users due to the limitations of each domain and single content type [141].

Even within a singular domains, users can have different personal information management strategies leading to different retrieval techniques [153]. Several user studies have attempted to classify user behaviour into different categories with respect to their filing and archiving preferences: filing or piling [177] denotes whether or not a user categorises and stores files in an ordered hierarchy or an unsorted structure; Abrams et. al. defined no filers, creation-time filers, end of session filers and sporadic filers [10] dependent on when (if at all) users reorganised and maintained their personal collection of URL bookmarks; email filing behaviour was studied by Whittaker and Sidner [305] who defined frequent filer, spring cleaner and no filer. The "no filer" category identified in the original study was later split further by Bälter into folderless spring cleaner and folderless cleaner [19]. The different behaviours observed in these user studies support the need for a Virtual Secretary to have multiple personalities in order to be an effective assistant.

As well as domain specific applications, several tools have attempted to provide a complete solution for the searching and display of personal information stores. Stuff I've Seen [90] by Microsoft Research presents a search interface for locating information of different formats which the user has seen before. The search starts over a broad area and then provides the user with contextual clues (such as screenshots or keywords) so the search can be refined [78]. MyLifeBits [109] works in a similar fashion with more support for user annotation, but only over multimedia files.

LifeStreams [104] replaces the desktop metaphor with a searchable, time-ordered stream of information. Until recently, the user interfaces for personal information management tools had changed relatively little since the invention of the desktop metaphor [70]. This has changed with the recent research activity and as a result, most of the research systems created are file browsing systems; interfaces which lay over the top of the current file storage system technology [180] instead of whole new file systems. As this is the case, the user interface was not considered as a part of the scientific contributions of this work.

3.3.1 Computer Supported Cooperative Work and Groupware

The terms "groupware" and computer supported cooperative work (CSCW) were coined in 1984 by Irene Greif and Paul M. Cashman [122]. They are often used interchangeably but have distinctly different meanings. CSCW is a multi-disciplinary field which studies the use of technology to support group activities [299], observing how people work in groups as well as the hardware, software, services and techniques needed to support that activity [310]. A more succinct definition was given by Wilson:

"CSCW [is] a generic term which combines the understanding of the way people work in groups with the enabling technologies of computer networking, and associated hardware, software, services and techniques." [309]

CSCW can refer to the study of a single user application (as in HCI research) or applications designed for organisations [123]. CSCW applications can be labeled with a number of names: groupware, group support systems, collaborative computing, workgroup computing and multiuse applications [110]. However they are referred to, these tools are designed to enable many participants to collaborate and work toward a common deliverable. Spanning a wide range of software that enables teams of people to work together efficiently [62], groupware usually runs over a network, allowing a group of people to access the same data or work on the same project [93].

The size of a group can be small (e.g. two people conducting a meeting via video conferencing) or considerably larger (e.g. the general population using an electronic voting system). Groups can be tight-knit with shared goals, tasks and common knowledge or alternatively very loose and amorphous with no explicit shared goals or knowledge of the other members. This type of group is the kind commonly found on the Internet [299] using one of the many web-based collaborative tools. One such tool is Wikipedia [231], an online encyclopedia that anyone can edit. This particular type of online collaborative system illustrates the problems of provenance or illustrating where a piece of data came from [106]. Groth et. al. [121] defined the provenance of a piece of computer data as "the process that led to that piece of data." Tracing the origins of data on the web and particularly in collaborative environments is a popular ongoing research area [55] with unanswered questions too numerous to mention here.

Groupware can be considered as software tools for CSCW and despite the name, there are many single-user software tools which have been upgraded or updated to become "group enabled". For example, a text editor designed for a single user which has an integrated electronic mail feature could be considered groupware, or at least to have a groupware aspect [299]. Obviously in this example, the text editor is a less collaborative tool than, for example, an electronic whiteboard which is used concurrently by several different people to exchange ideas. In this way, it is easy to see that producing a strict definition of what is (and isn't) groupware would be near-impossible. Instead, a spectrum of common task dimensions can be used to evaluate how tightly-coupled the software and extent of collaboration are [95]. The text editor example given above would be at the low end of the groupware spectrum, as it provides few environmental cues. A system such as an electronic meeting system [169] with projectors, interactive whiteboards, video conferencing, access for remote users and so on would rank much higher up the groupware spectrum. However, the addition of extra functionality to groupware can increase tool complexity (termed "bloating") [186]. In order to analyse and evaluate groupware systems, software usability inspection techniques [216] and task analysis schemes [217] have had to be updated in order to be flexible enough to model the collaborative aspects which were absent from previous software.

CSCW research into groupware deployment has studied both the impact on interpersonal relations between users [230, 252] and also the effects upon organizational efficiency [84]. Within a commercial or organisation setting, groupware and associated technologies have the advantage of generating a continuous record of exchanges via the electronic communication mediums, providing an account of project/company progress far more detailed than that of traditional minutes taken in meetings [170]. In this way, groupware can contribute to a support the ability of an organisation to retain and archive its own history, referred to as "organisation memory" [154]. This is very similar to the technique the GIFS framework uses to produce a complete record of a file's history.

By using one or more of the artificial intelligence and personal information management techniques mentioned in this section and §3.2 many pieces of software have been developed, (with varying degrees of success) to assist users with the problems of information overload and file storage. Whilst it would not be possible to include every single attempt at assisting users in a personalised format, the remainder of this section looks at several different domain-specific applications.

3.3.2 Personal Assistants

Personal assistants are a research area that has captured the imagination of computer scientists for many years [74]. There are several characteristics desirable in a digital personal assistant [75]: a common user profile for use by all one user's agents, the ability to adapt to the user's preferences, the capability to share information with other agents and to collaborate with each other, and privacy of information.

A study into "Open Sesame!" [136], one of the first commercially available personal assistants for use on Macintosh computers found that it lacked the requisite flexibility needed in order to assist the user, as it tried (and predictably failed) to surmise what caused a user to make particular decisions from past events and actions alone. Equally, the Lumiere project [134] is the basis for Microsoft's "Office Assistant", and uses Bayesian user modeling to predict user behaviour in context of the suite of Microsoft Office applications. One aspect that the office Assistant does not help with is filing and document management, and many users have questioned the value of this particular assistant in reference to its intrusive and often annoying behaviour. It is desirable for the user to have the final say in actions suggested by an autonomous system, for example when Smartlook [245], an e-mail classification assistant, organises user's emails into a set of hierarchical folders it offers the most likely 6 folders and allows the user to make the final choice. A study of this technique showed that users can tolerate errors from a personal assistant system (as building a 100 percent accurate user model is rarely possible), so long as the assistant achieves reasonable performances. This technique also works well for those situations where an agent cannot find enough data to make a confident suggestion. Bauer et al. [25] proposed a set of agents called "Trainable Information Assistants" that generated scripts to extract information from websites, even when the format and layout of these websites changed over time. When encountering problems the agent makes the user aware of its limitations and asks for suggestions.

3.3.2.1 Office Assistants

It has been acknowledged for some time that there should be comprehensive systems in place to help office workers in the more basic information management aspects of their job [96].

Office Assistant [315] is an implementation of an agent that interacts with people at the door of an office and manages the office owner's schedule. It is context aware, changing behaviour dependent on current situation of the office owner and works in conjunction with their schedule and those of other people to organise meetings. There are several other similar systems: KautzBot [150] and SelmanBot [149] are both software agents that communicate via e-mail in order to arrange meetings and to automatically negotiate convenient times for all involved participants. Within an office environment it is not just scheduling systems that have incorporated agent technology. The Multi Agent Referral System (MARS) [316] was created on the basis that much knowledge within an organisation or office environment may go unpublished, so there should be a system to help manage the social network of a user. Each user is assigned an agent and assists them in obtaining and following referrals in order to find the person in the organisation who would be most likely to help satisfy the user's informational needs.

The term "Virtual Secretary" was introduced by Bellika et al [28], who aimed to combine user models and software agents to create adaptive user interfaces. It was developed to assist and imitate the user's computer associated actions, for which it needed some knowledge about the way the user performs his or her tasks. Part of the functionality of the secretary was to locate files on computers that a user had previously been working on, given that the computers were connected to a global network [29]. Searching only via a given set of keywords introduced many problems in itself, least of all those of keyword assignment when the document is written. Whilst it was apparent that user model-based software agents were a useful tool for software personalisation, the concept only applies to long-term information interests which go beyond a single session need for information. A later implementation of this project [39] used a twin-based set of cooperative agents in order to manage distributed knowledge. Each agent acts as a domain expert, and if it does not know the answer, asks another agent on the network. Thus the set of agents can work together to answer user queries even if the original agent does not know the answer. Although very similar in concept and ideology to GIFS, the examples seen in the second paper were far more generalised, and had moved further away from the concept of filing and more toward querying knowledge bases.

3.3.3 Personalised Views and Searches

Given that it is already widely acknowledged that a hierarchy is an unsuitable display metaphor for the contents for a file system [43, 174, 23, 158, 108], much research has revolved around the best way to present this information to users and how best to personalise interfaces. An observational study of file accesses [23] showed that users preferred to search for a file visually rather than trying to recall the filename they assigned to it previously, even though this method was less efficient. The search function by name was only used as a last resort even in the cases where it would have considerably reduced the task time. A prototype system of 3D document organisation called "Data Mountain" [238] took advantage of 3D spacial cognition abilities of humans. Users were able to place their "favourite" websites contents at arbitrary positions on an inclined plane in a 3D desktop environment. Another system, MSpace [254] searches for information about music/media files by integrating metadata from the files and displaying "time slices" of the data.

Customisation of user interfaces can already be seen in many of today's commonly used applications: the ability to select which menus to display, to add toolbar buttons, to define macros and to add custom functionality through the use of a scripting language. However, naïve users lack the required skills to program their own scripts, and advanced users do not have the time to spare to do so, even if it would save them time in the long run [301]. The Information Programming Toolkit (IPtk) [100] is a framework which sits between applications and low-level information storage to enable the creation of personalised and adaptive user interfaces. This allows users to refer to documents by semantic structure, relationships and context but does not change the underlying problems with the rigidity of a hierarchical files system. Creating interfaces that adapt to users without direct instruction can also be problematic. Microsoft's SmartMenus are disorientating [105], and as mentioned previously, the Office Assistant falls far short of the mark in the stakes of helpfulness. Hence the main aims of intelligent user interfaces should be to provide a more efficient, effective and natural interaction between a human user and computer [183]. Mulvenna et al [201] proposed that there are three principle components to this observational personalisation: analytics, representation and deployment. Once such interface [172] provides just-in-time assistance by predicting the user's most likely plan and performing parts of the plan in the user's behalf. André and Rist [14] illustrated three different systems utilising animated virtual characters in order to personalise the web browsing experience. In these tests, it was found that users preferred to have an interface which provided them with some kind of social context.

Although alternative and personalised views of file systems and information stores can help ease the problems of information overload [44], they obviously do not provide a complete solution. Also of great importance are the methods for searching through and retrieving files and data. When searching for and retrieving documents, keyword searches on their own are not enough [284]. Imagine going into a library and searching for a book on the criteria that it has the word "computers" somewhere in the text. Asking a librarian would add a more sophisticated specification using other information like genre, time frame and content description, and thus so should computers assist in the location of data. Most of the time when people are searching for something, it is because they do not have sufficient knowledge themselves to solve the query already. However, it can be argued [27] that users do not know the best way to search, (that is, the internal representation of data) in order to form the most effective query. Agents have not only been used in conventional search engines on the Internet, but also in systems where user preferences can be expressed in order for more relevant documents to be highlighted [24].

Jansen [140] created a taxonomy of 26 interactions split into 9 groups for users performing web searches, a number of actions which is greater than most users would expect. Perhaps this large number of actions all happening via the same interface has some bearing on the studies [212] which have shown that both inexperienced and advanced search engine users still have difficulty in efficiently completing a search task on the Internet. Many different sources have suggested that the accuracy and efficiency of search results on the Internet could be improved by incorporating user feedback [125], and the addition of personalisation agents [218] in order to provide the service of "intelligent information retrieval" [26].

Teaching a system to the point where it can competently predict a user's needs is a task that is manually undesirable, the set-up costs of such a system means that a user may not save time by using it. It is therefore far more desirable to unobtrusively observe a user's behaviour in order to learn functions of value [114, 90]. An alternative to this can be seen in a system named Apt Decision [258], used to find real estate rentals. A constant feedback process allows a user profile to be built up without redundant or unnecessary effort from the user. Phlat [77], an interface for personal searches using user created keywords and metadata suffers from the similar problem of requiring the user to invest time in training the system.

3.4 Discussion

With information overload already a sizable and frustrating problem for almost all computer users, new ways of organising, processing and displaying data are obviously needed. Using a computer now means that unavoidably some time will have to be devoted to the administration and housekeeping tasks that are associated with file storage. It is unlikely that the problems and effects of information overload will decrease naturally, and so new solutions are needed in order to help users manage both their personal data and time.

Using computers to store and analyse large amounts of data and metadata can provide very useful information to use when personalising interfaces or suggesting courses of action to a user. As computers become progressively more powerful it is possible to store and process data with increasing speed, so the information that can be deduced and extracted from this data is of great importance, but only if it is managed, processed and analysed in a meaningful way.

Whilst artificial intelligence is not yet at a level comparable to that of humans, there are many agent systems that can make life easier for users by performing a small subset of actions. However, almost all of the systems described in this review chapter are either incredibly domain specific and inflexible, or not specific enough and incapable of performing any tasks well. Another problem, even for the small and specific systems, is that of scalability. There does not seem to be any data on the deployment of personalised systems for a large number of users or in some cases, over non-personal data stores (e.g. the Internet instead of one single computer or local network) In order for these systems to be successful over a large network or for a large number of users, the architectures and procedures would need to be very carefully planned and most likely redesigned. As the preceding discussion illustrates, despite the huge number of ideas and research papers behind intelligent agent systems and personal file storage mechanisms, many of these systems have never been implemented.

The research areas of personal information management and computer supported cooperative work support the ideologies of this thesis and several constituent parts of the GIFS framework. However, the research systems developed within these fields are either restrictively domain specific or focus on creating new user interfaces over existing file systems.

There are a plethora of agent systems which are described as secretaries or personal assistants for use in an office environment, but this description is slightly misleading as the majority of them concentrate on tasks such as meeting scheduling and diary management rather than those concerned with file storage. Whilst it may be the case that these systems are appreciated more in the office environment, all computer users would benefit from automated assistance with their file manipulation activities.

Previous agent or otherwise personalised assistants have sometimes hindered rather than helped. Using a system which produces incorrect or inconsistent results is far more frustrating and time consuming to a user than not using such a system at all even if most users are willing to tolerate a reasonably small amount of errors from their assistant. The best way to train a computerised personal assistant is arguably through observation of the user's actions, although this on its own is not a good basis for making predictions as some kind of more complex analysis will be required so as to provide consequential results. Where these results have been used in order to change the interface of file storage systems, they have all concentrated on simply displaying files to users or organising search results in alternative ways and have not addressed the system level problems of file storage.

No system has yet combined these technologies in order to provide a solution for file storage, retrieval and organisation in the same way as GIFS. Previous attempts at creating document management or file storage systems have all used existing file systems as the underlying storage mechanism, hiding the current problems from the user rather than solving them.

Chapter 4

GIFS Overview

4.1 Introduction

This chapter presents an overview of the Virtual Secretary (VS), a knowledge-based user interface. The Virtual Secretary forms the front-end of a system named GIFS, a Global Intelligent File System. A user instructs the VS to perform tasks for managing and storing files instead of manipulating them manually via windows as with many conventional operating systems. The VS manages all the technical details related to each task, including the actual location of the files and storage format, security and access permissions in addition to handling all encryption and key distribution activities. Whilst performing each task the Virtual Secretary gathers raw knowledge about the user's actions in a knowledge base which is then managed and processed by a collection of agents. These agent programs work behind the scenes aiding the analysis and gradual proliferation of knowledge. For example, a grouping agent can establish relationships between similar files from existing knowledge (such as their names, sizes, dates, keywords etc.), and group them in various ways. With the support of the knowledge base and agents, the VS is able to provide the user with a more intelligent service than current systems. For instance, when the user requests a document the VS may also thoughtfully bring a folder containing all related files. Over a long period of service and with the addition of more agents, the VS has the potential to be evolved into a highly intelligent assistant.

From the perspective of the user, a Virtual Secretary is an adaptive user interface running in the background of their computer. From time to time, the user instructs the VS to perform certain tasks, such as storing away a file, retrieving a group of files related to a specific context, or informing colleagues that a shared file is available for viewing. The VS acts as the connecting link between the user and the computer's conventional interface for file storage whilst gathering raw data to be managed, processed and proliferated by the collection of agents to facilitate an adaptive service.

4.2 Background

A user operates a computer largely through manipulation of various files. The needs for secure and intelligent file management are two-fold, namely *file organisation* and *sharing*.

File organisation is a perpetual activity for every computer user. Yet the level of difficulty in this activity is becoming increasingly noticeable, largely due to information explosion and deficiencies in current file systems. For example, many managers and secretaries are constantly looking for extra disk space for storing documents, or looking for files previously created on their computers. The hierarchical tree structure available in most file systems (directories and folders) is a satisfactory mechanism for short-term and small-scale document organisation. However, it becomes often clumsy, problematic and less user-friendly when dealing with a large volume of files that are to be maintained over a long period.

File sharing is perhaps the most common activity in a collaborative environment. A user may create a document, and wish to distribute it to a group of other users. There are usually some additional requirements associated with this task, such as read/write access permission and transmission security. Despite the fact that a variety of mechanisms may be used for supporting this activity, they lack in either user-friendliness or security, and in comparison with a human secretary leave a lot to be desired. Email attachments, a very commonly used mechanism, may incur unnecessary duplication and excessive space wastage. Many organisations and computer users are being inundated with email attachments sent to them endlessly and often pointlessly. Uploading a document onto a web site is another technique typically for read-only file sharing. This however requires some technical skills to set up the service on the server. Networked operating systems such as Windows also offer procedures for shared user spaces and network drives. Nevertheless, most of these methods are based on file owners or groups, rather than on individual files. To most users, creating a mechanism in order for others to share a specific file is not really a trivial task.

This work is primarily motivated by the needs for developing a globally integrated management system for administrating the storage, access, sharing, manipulation and security of files across interconnected computers. In the UK First Workshop on Grand Challenges for Computing Research, a vision of such a system, called Global Intelligent File Tele-System (GIFTS), in action was presented [63].

4.3 Overview

A Global Intelligent File System (GIFS) is a globally integrated management system for administrating the storage, access, sharing, manipulation and security of files across interconnected computers. The Virtual Secretary (VS) is the knowledge-based user interface for such a global system. Whilst it is not feasible to address the wide range of difficulties in prototyping a GIFS within this project, the effort has been focused on the development of the VS user interface and its supporting knowledge framework based on the current operating system and data communication technologies. Figure 4.1 illustrates the overall system infrastructure with two Virtual Secretary systems operating in a networked environment.

A typical environment where a VS may operate involves a remote file server which is supported by a larger volume of disk space than a typical desktop or laptop computer. In such an environment, a VS will use the remote server as the primary storage area for file warehousing. The communications between Virtual Secretaries who serve different users are handled by using socket-based messaging system. A VS may also communicate directly with any user using the email system although the preferred method of communication is alerting a user via their VS. We also believe that it is technically feasible to integrate a VS user interface with the email system, enabling emails to be processed as files, because each email is essentially a text segment in a very large file (i.e. mbox or pst). A Virtual Secretary hides from the user much of the technical detail concerning file organisation, grouping, directory/folder structure, attribute setting, and so on. It is also a gate keeper for the user for safe-guarding his/her files. When the user instructs the VS to store a file away the file itself is stored in a location unknown to the user. A public key encryption algorithm is applied to each document before it is transmitted to the external storage area, and if it is to be visible to other users their own Virtual Secretaries are automatically informed of the key they will need to view that file.

The concept of GIFS and its main features are significantly different from those of NFS and WWW. Some of the new features expected are summarised in Table 4.1.

The transformation of file systems and the World Wide Web to a new, integrated, global, intelligent technological infrastructure for information distribution is a dramatic idea, which requires and promises a major paradigm shift in our approach to the design of operating systems and global communication infrastructure. The history of operating systems (including distributed operating systems) and World Wide Web provides a convincing answer to this question. Although there are commercially successful operating systems and web-browsers, academic and industrial research has played a leading role in the development of these technologies. Most major technological innovations in these areas were delivered through non-profit-making systems (e.g. Unix, NFS, TCP/IP and HTTP). In addition, the society, computer scientists, software engineers and users will have serious concerns about a proprietary GIFS.

Between 1977 and 1988 there were over 12 different distributed operating systems (including Cambridge DCS, Newcastle and NFS) developed. In the 1990's, the focus was shifted to the Internet and WWW, and there were no remarkable advances in

	NFS	WWW	GIFS
Design Approach	user-centred	URL-centred	file-centred
Notion of Files	reasonably generalised	restricted	generalised to include most communication channels
Access Management	user-centred password control; controllable operations include read, write, execute, list	mostly open access, with application-specific access control, most files are read-only, with limited uploading, listing and script execution	mostly open access, with file-centred access control; controllable operations include read, write, append, delete, execute, list, search and encrypt
User Interface	WIMP based GUI, no built-in intelligence	WWW browser, generally no built-in intelligence	virtual secretary
File Attributes	location path, permissions, dates	location address, content-based links	location mapping, access privileges,version control, user groups, search keys, encryption keys, life-span, attribute-based links, actions, ...
Closely Integrated Tools	limited plug-ins for file display	email, ftp and various plug-ins for file display	email, ftp, mobile devices, voice recognition and various plug-ins for file display

Table 4.1: Features of NFS, WWW and GIFS

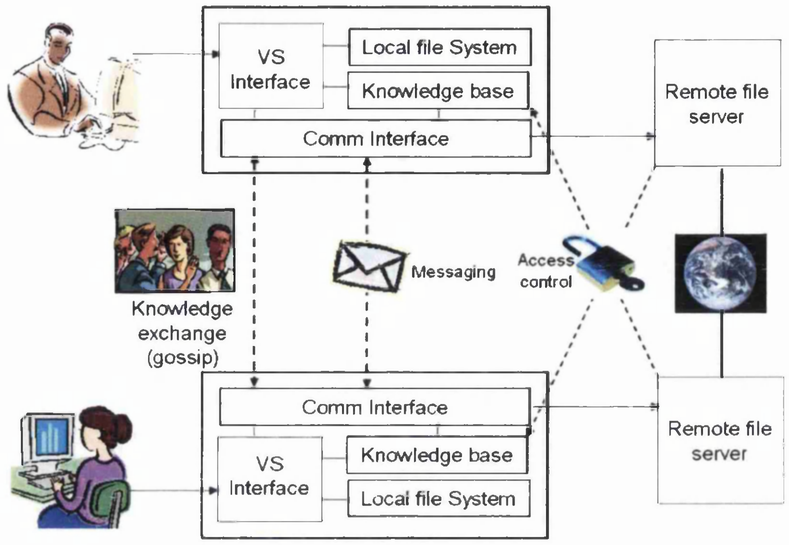


Figure 4.1: The GIFS Infrastructure

operating system technologies. The lack of development in this area may attributed to (i) the focus of research resources on WWW in the 1990's, (ii) the dominance of proprietary operating systems in the computer industry, (iii) the difficulty to manage knowledge and intelligence in an open knowledge framework.

The notion of GIFS was formulated through a recent project conducted at Swansea. Although much smaller in terms of size and achievements, a final year undergraduate project demonstrated the feasibility of a global file system. Due to time constraints at only a small subset of basic features were implemented, however the main objectives and vision of a global file system remained constant throughout.

To help us to identify needs of this work, we can visualise the following futuristic scenario:

A user called Jo starts a working day with writing a memo for a committee (`create file`). She then drag-and-drops the memo onto a Virtual Secretary icon on the desktop (`store file`), which is a standard GIFS utility just like the recycle bin. The Virtual Secretary suggests to Jo how the file will be classified, and how the committee members be informed of the availability and access key. After Jo selects and confirms the options, the memo is sent to a GIFS server (`send file`), and a set of correspondences are sent to the committee members via their Virtual secretaries. Jo then asks the Virtual Secretary to display all tele-meetings which she has registered her interest and are authorised to attend today. The tele-meetings which matched the criteria are displayed as folders. As she is unable to choose which meetings to attend, she asks her Secretary to compare them to the meetings she attended yesterday, and which are likely to be the most similar on content, based on other attendees and literature currently available (`compare file`). Two meetings are returned as a result, and Jo decides to attend both simultaneously by double-clicking the two corresponding folders, (`open file`) each leading to a window containing all documents (such as presentations and agendas) and devices (such as camera and slide projector). Jo activates her own camera (`open communication stream`) and participates in the meetings, taking her own notes and giving them to her Virtual Secretary to store along with the other relevant documents. After the meetings, Jo's real-life Secretary brings her a cup of coffee. The Virtual Secretary notices that it is 11am and Jo has no meetings scheduled or documents open, and asks if Jo would like to listen to Classic FM whilst she enjoys her coffee break.

Task	User	File System
file placement	✓	
semantic file naming	✓	
physical file naming	✓	
metadata tagging	✓	
file storage		✓
encryption	✓	
key distribution	✓	
file distribution	✓	
locating files	✓	
archiving	✓	
version control	✓	
file relationships	✓	

Table 4.2: Task assignment of users and previous file systems

The above scenario exemplifies a new way of managing various files and communications over the Internet which cannot be supported by today's operating system and Internet technology. As most of the operations performed by Jo are essentially operations on files, this suggests a new approach to the user interface design for an operating system is necessary.

There now approaches a paradigm shift between the tasks of users and those undertaken by a file system. Whilst previously users may have been assisted in certain tasks by their computer (such as versioning or encryption), the user had to specifically request them via an additional program external to the file system itself. Tables 4.2 and 4.3 show the divisions between the tasks of user and file system both for traditional file systems and GIFS.

Obviously, the more of these tasks that a user is responsible for, the more of their time and effort is required to manage their data and file collections. As computer usage has increased most users are probably not aware of how many extra tasks they are having to perform which can be seen listed in Table 4.2. There is no technological reason why a computer should not be responsible for these tasks as seen in Table 4.3, unburdening the user.

The aims and objectives for the design of this system are set out as follows:

- to store a file away in a transparent and secure manner
- to decide where and how the file is stored
- to enable a wide range of search functions over the files including similarity groupings
- to retrieve the file when some information about the file is given

Task	User	GIFS
file placement		√
semantic file naming	√	
physical file naming		√
metadata tagging		√
file storage		√
encryption		√
key distribution		√
file distribution		√
locating files		√
archiving		√
version control	√	√
file relationships		√

Table 4.3: Task assignment of users and GIFS

- to assist the user in distributing the file in a collaborative environment
- to inform all users who are allowed to share the document about its availability
- to provide additional security to the document using encryption
- to manage file access privileges, user groups, passwords and encryption keys
- to analyse data to provide more accurate and faster searches and just-in-time results for the user
- to protect the user from errors causing accidental file deletion and overwriting
- to automatically version each file and allow any version to be accessed
- to archive older versions of documents

There are many technological challenges associated with the concept of a global file system, the main ones being:

- To generalise the notion of files to accommodate multimedia communication over the Internet, and define an international standard that supports an open architecture for a globally distributed GIFS.
- To make a paradigm shift in file system design possible by adopting a file-centred design approach.
- To support multi-level file security and access management, incorporating public key encryption and digital signatures into the set of file attributes managed by GIFS.
- To develop intelligent and adaptive user interfaces that act as Virtual Secretaries with knowledge of users, files, their common correspondences and their

working environments.

- To develop a generic knowledge representation and management strategy that can be supported by generations of operating systems in an evolutionary manner.
- To deploy advanced technologies in GIFS for providing file services through mobile communication and voice-driven interfaces.
- To provide direct support to a wide range of applications involving data and information management, including e-business and e-government.

4.4 File Lifecycle

During the early stages of research, a presentation was given to the computer science undergraduate students during the annual colloquium. In order to help their understanding of this novel concept, a short video project was produced. This video (which can be found on the DVD disk marked "Appendix C" at the back of this thesis) took a tongue-in-cheek look at the operations of the Virtual Secretary as if it were not a computer program but still a real-life secretary. For those people who are unable to view the DVD, some of the basic features can be seen as screen shots in Figure 4.2.

The first two screen shots show a user who is obviously suffering from information overload, both on his computer desktop and in real life (it is worth noting here that this particular location was used 'as-is' for filming, and the desk was piled up with papers before filming started). In row 2 we see the Virtual Secretary (or in this case, the not-so-Virtual Secretary) taking files from users to be stored elsewhere, retrieving those files, and delivering files to other users. Row 3 shows the secretary preventing unauthorised access to files by keeping them out of reach of certain users and by 'fighting off' intruders. Finally, row 4 shows the secretary encrypting a file by tearing it up into small pieces. If a user were to intercept these pieces they would be meaningless and impossible to understand, however with the aid of the Virtual Secretary the file can be reassembled perfectly at its destination.

The video also shows how the Virtual Secretary 'interface' can be customised to the user's needs, as seen in Figure 4.3. Picture 1 shows the option to have faster results delivered (perhaps at the cost of accuracy), picture 2 shows the secretary performing tasks more independently and without the request of the user. The picture marked 3 shows the secretary being overly intrusive whilst picture 4 shows the secretary demonstrating a worrying amount of stealth. Pictures 5 and 6 show the secretary being friendlier and more forgiving toward the user or alternatively becoming more strict and bossy with respect to the user's actions.

1: Information Overload

2: File storage and retrieval

3: File security

4: Encryption

Figure 4.2: Some basic actions within GIFS

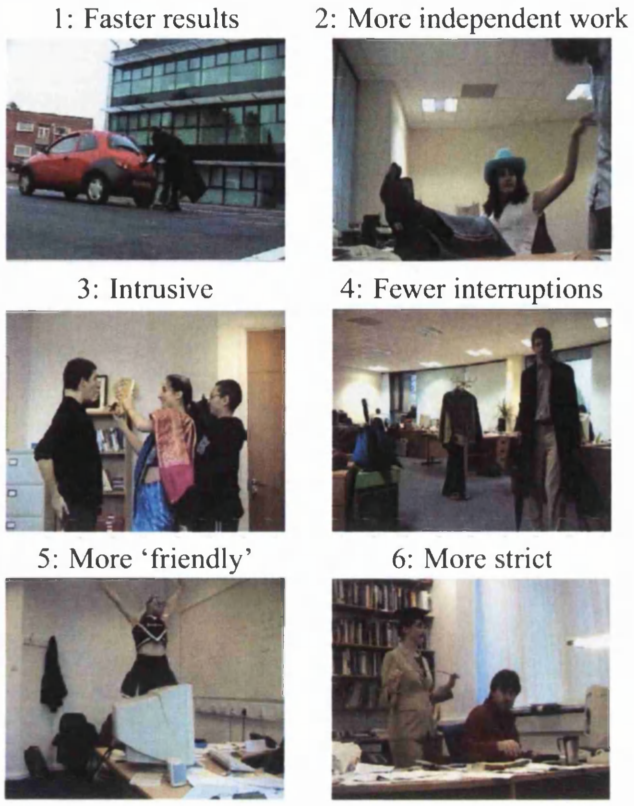
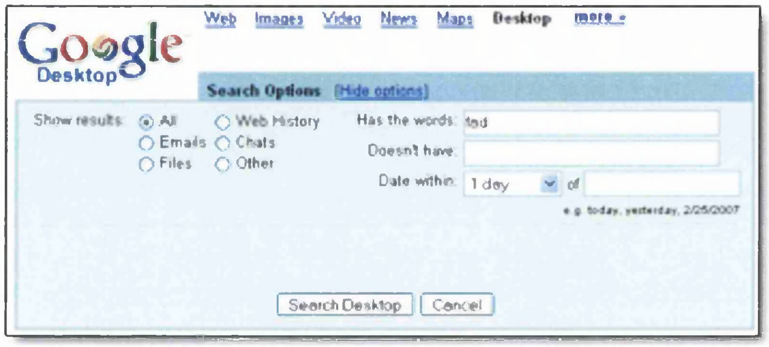


Figure 4.3: Personalisation of the Virtual Secretary

Whilst some of the features seen in the video are not wholly accurate representations of this research, it provides a good (and not entirely serious) overview of how a Virtual Secretary might work and the problems that users may encounter, without the need for any in-depth technical knowledge. Since the making of this video the focus of research has shifted slightly as it became apparent that searched-based interfaces were already being produced in commercial software. Instead it was decided to focus the research on the more challenging scientific problems that had been unearthed. However it still remains advisable to watch this video if possible to help overall understanding of the long term development goals.

Since the proposal of our grand challenge in 2002 [63], and our proposal of the concept and system framework in 2004 [64], the concept of search-based has appeared several pieces of commercial software developed independently from this work.

For example, Figure 4.4 shows the interface of Google Desktop Search [1] which enables users to search their personal files by maintaining an index of all compatible files on their computer. Figure 4.5 shows the user interface of Spotlight, a feature of the MacOS X 10.4. Similarly to Google Desktop, Spotlight maintains indexes of file metadata in the background. Also produced by Apple, for use with their Ipod music players, the interface of iTunes can be seen in Figure 4.6. Although it only runs over a subset of file types (e.g. media files), the interface provides a variety of searching mechanisms based on the metadata of a user's personal media library. In much the same way to the above approaches, Microsoft research produced Stuff I've Seen (see Figure 4.7) and its successor, Phlat [79]. Stuff I've Seen [90] helps users to locate files, documents, calendar entries and so on that they have previously accessed. Information on these objects are all stored in the same index, regardless of what format the file or object originally took. The user interface also gives rich contextual clues for users to narrow down their search results. In comparison, Phlat focuses more on the design of the user interface and as a search tool for personal information management. The search interface for Phlat can be seen in Figure 4.8.



Show results: [✓ All] Web History Emails Chats Files Other

Has the words: ted

Doesn't have:

Date within: 1 day of

e.g. today, yesterday, 2/25/2007

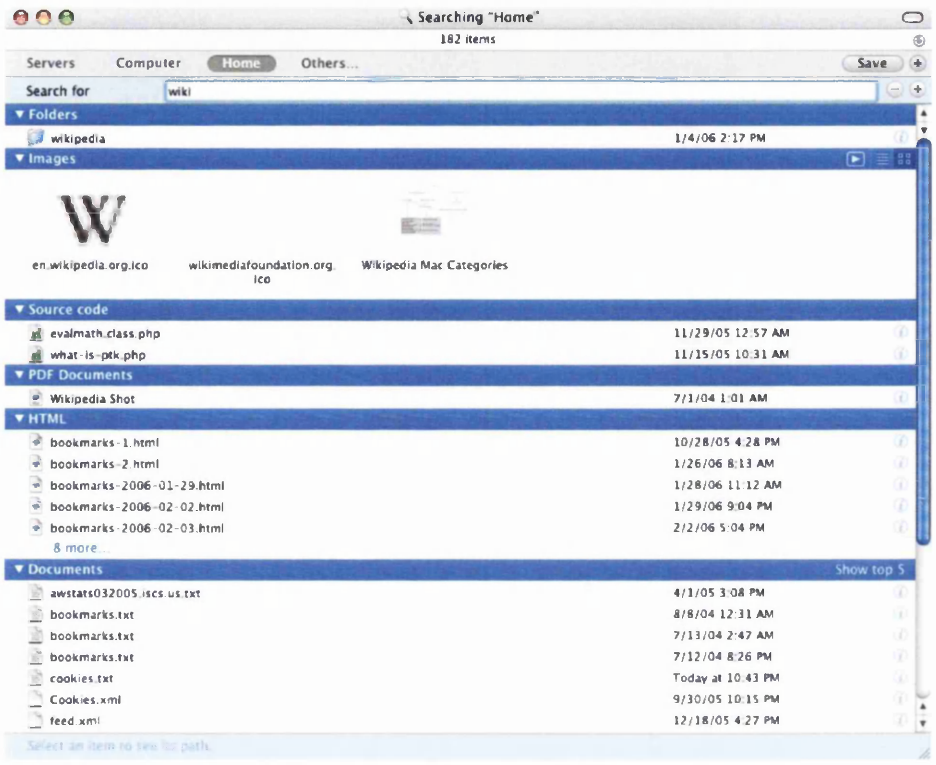
Figure 4.4: The user interface of Google Desktop Search

An earlier feasibility study, JoFS [115], which was undertaken by the author in 2001, had a basic web-based interface as shown in Figure 4.9. It provided the search functionality for the network file system that was developed and worked well as a proof of concept but was not closely bound to the workings of the operating system.

The user interface of GIFS is further integrated with the operating system in order to give users more of a feel of a native file system. However, as user interaction was not the focus of this research, the example interfaces presented are designed to facilitate the basic file access flow and knowledge transaction flow, which are the main focus of this work. In order to show how these interfaces might appear several example screenshots can be seen later in this chapter, but were not considered to be a main contribution of this work.

4.4.1 User Action Taxonomy

Many of the advanced features of GIFS and the Virtual Secretary are hidden from the user in a transparent fashion. In order to give the reader an overview of the system



▼ Folders

wikipedia	1/4/06 2:17 PM

▼ Images

en.wikipedia.org.ico — wikimediafoundation.org.ico — Wikipedia Mac Categories

▼ Source code

evalmath.class.php	11/29/05 12:57 AM
what-is-ptk.php	11/15/05 10:31 AM

▼ PDF Documents

Wikipedia Shot	7/1/04 1:01 AM

▼ HTML

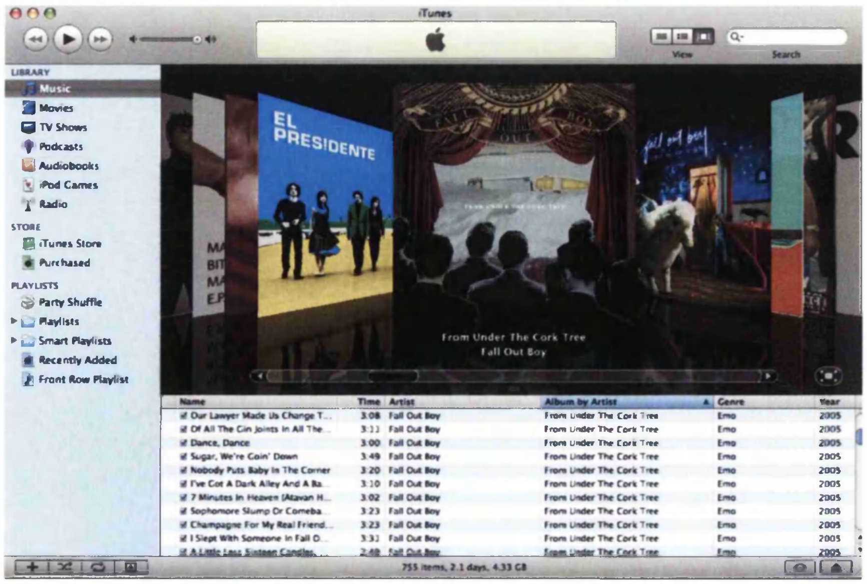
bookmarks-1.html	10/28/05 4:28 PM
bookmarks-2.html	1/26/06 8:13 AM
bookmarks-2006-01-29.html	1/28/06 11:12 AM
bookmarks-2006-02-02.html	1/29/06 9:04 PM
bookmarks-2006-02-03.html	2/2/06 5:04 PM

8 more...

▼ Documents

awstats032005.iscs.us.txt	4/1/05 3:08 PM
bookmarks.txt	8/8/04 12:31 AM
bookmarks.txt	7/13/04 2:47 AM
bookmarks.txt	7/12/04 8:26 PM
cookies.txt	Today at 10:43 PM
Cookies.xml	9/30/05 10:15 PM
feed.xml	12/18/05 4:27 PM

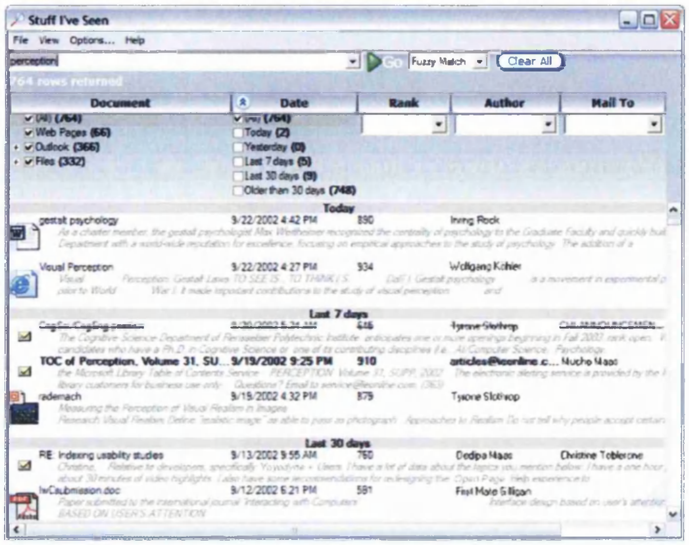
Figure 4.5: The user interface of Spotlight



Name	Time	Artist	Album by Artist	Genre	Year
Our Lawyer Made Us Change T...	3:08	Fall Out Boy	From Under The Cork Tree	Emo	2005
Of All The Gin Joints In All The...	3:11	Fall Out Boy	From Under The Cork Tree	Emo	2005
Dance, Dance	3:00	Fall Out Boy	From Under The Cork Tree	Emo	2005
Sugar, We're Goin' Down	3:49	Fall Out Boy	From Under The Cork Tree	Emo	2005
Nobody Puts Baby In The Corner	3:20	Fall Out Boy	From Under The Cork Tree	Emo	2005
I've Got A Dark Alley And A Ba...	3:10	Fall Out Boy	From Under The Cork Tree	Emo	2005
7 Minutes In Heaven (Atavan H...	3:02	Fall Out Boy	From Under The Cork Tree	Emo	2005
Sophomore Slump Or Comeba...	3:23	Fall Out Boy	From Under The Cork Tree	Emo	2005
Champagne For My Real Friend...	3:23	Fall Out Boy	From Under The Cork Tree	Emo	2005
I Slept With Someone In Fall O...	3:31	Fall Out Boy	From Under The Cork Tree	Emo	2005
A Little Less Sixteen Candles...	2:48	Fall Out Boy	From Under The Cork Tree	Emo	2005

755 items, 2.1 days, 4.33 GB

Figure 4.6: The user interface of iTunes



Today

gestalt psychology 9/22/2002 4:42 PM 890 Irving Rock
As a charter member, the gestalt psychologist Max Wertheimer recognized the centrality of psychology to the Graduate Faculty and quickly buil Department with a world-wide reputation for excellence, focusing on empirical approaches to the study of psychology. The addition of a

Visual Perception 9/22/2002 4:27 PM 934 Wolfgang Köhler
Visual Perception: Gestalt Laws TO SEE IS... TO THINK I S... Gestalt psychology is a movement in experimental p prior to World War I. It made important contributions to the study of visual perception and

Last 7 days

~~CogSci/CogEng position~~ ~~9/20/2002 5:34 AM~~ 646 Tyrone Slothrop ~~CHI-ANNOUNCEMEN...~~
The Cognitive Science Department of Rensselaer Polytechnic Institute anticipates one or more openings beginning in Fall 2003, rank open. candidates who have a Ph.D. in Cognitive Science or one of its contributing disciplines (i.e., AI/Computer Science, Psychology

TOC of Perception, Volume 31, SU... 9/19/2002 9:25 PM 910 articles@ieonline.c... Mucho Maas
the Microsoft Library Table of Contents Service PERCEPTION Volume 31, SUPP, 2002 The electronic alerting service is provided by the library customers for business use only. Questions? Email to service@ieonline.com (363)

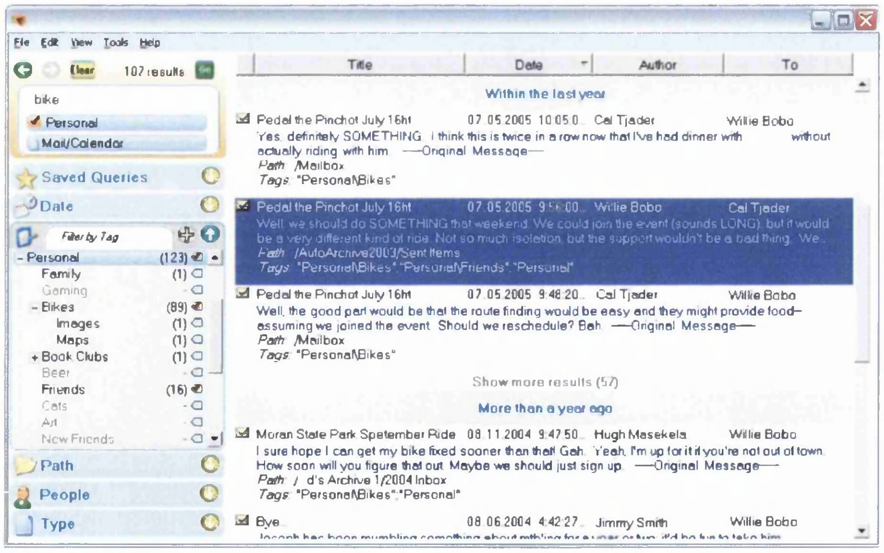
rademach 9/19/2002 4:32 PM 879 Tyrone Slothrop
Measuring the Perception of Visual Realism in Images
Research Visual Realism Define "realistic image" as able to pass as photograph Approaches to Realism Do not tell why people accept certain

Last 30 days

RE: Indexing usability studies 9/13/2002 9:55 AM 760 Oedipa Maas Christine Toblerone
Christine, Relative to developers, specifically Yoyodyne + Users. I have a lot of data about the topics you mention below. I have a one hour, about 30 minutes of video highlights. I also have some recommendations for redesigning the Open Page. Help experience to

IwCsubmission.doc 9/12/2002 6:21 PM 581 First Mate Silligan
Paper submitted to the international journal "Interacting with Computers" Interface design based on user's attention BASED ON USER'S ATTENTION

Figure 4.7: The user interface of Stuff I've Seen



Within the last year

Pedal the Pinchot July 16ht 07.05.2005 10:05:0... Cal Tjader Willie Bobo
Yes, definitely SOMETHING. I think this is twice in a row now that I've had dinner with without actually riding with him. ——Original Message——
Path /Mailbox
Tags "Personal\Bikes"

Pedal the Pinchot July 16ht 07.05.2005 9:56:00... Willie Bobo Cal Tjader
Well, we should do SOMETHING that weekend. We could join the event (sounds LONG), but it would be a very different kind of ride. Not so much isolation, but the support wouldn't be a bad thing. We...
Path /AutoArchive2003/Sent Items
Tags "Personal\Bikes","Personal\Friends","Personal"

Pedal the Pinchot July 16ht 07.05.2005 9:48:20... Cal Tjader Willie Bobo
Well, the good part would be that the route finding would be easy and they might provide food—assuming we joined the event. Should we reschedule? Bah. ——Original Message——
Path /Mailbox
Tags "Personal\Bikes"

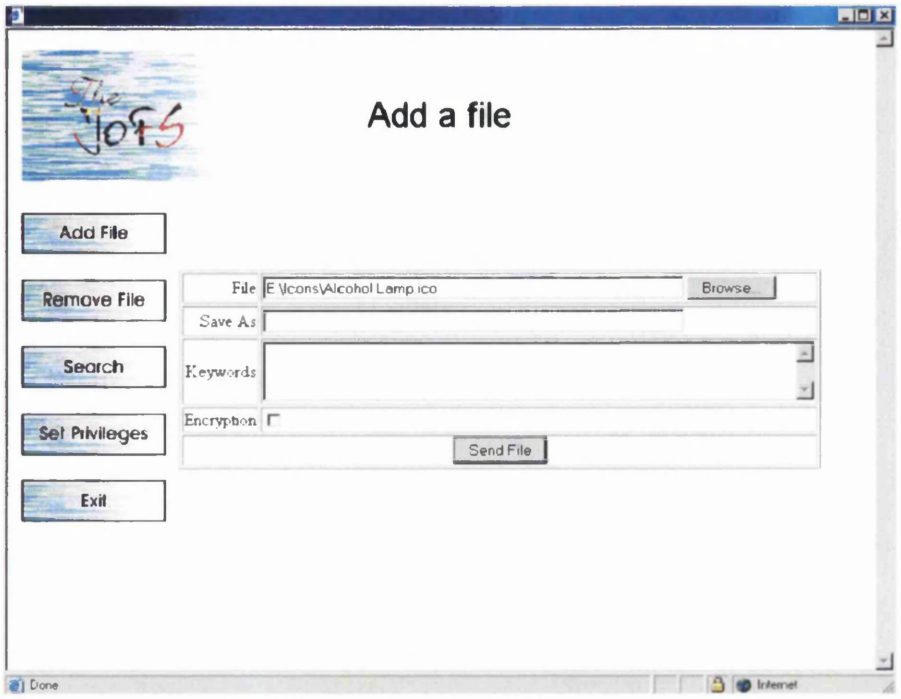
Show more results (57)

More than a year ago

Moran State Park Spetember Ride 08.11.2004 9:47:50... Hugh Masekela Willie Bobo
I sure hope I can get my bike fixed sooner than that! Geh. Yeah, I'm up for it if you're not out of town. How soon will you figure that out. Maybe we should just sign up. ——Original Message——
Path / d's Archive 1/2004 Inbox
Tags "Personal\Bikes","Personal"

Bye... 08.06.2004 4:42:27... Jimmy Smith Willie Bobo

Figure 4.8: The user interface of Phlat



File: E:\Icons\Alcohol Lamp.ico
Save As:
Keywords:
Encryption: ☐

Figure 4.9: The user interface of JoFS

from an operational viewpoint a user action taxonomy and examples are provided below. It should be noted that in these examples the behaviour of the system is documented through the eyes of the user, not a system designer and therefore there is no mention of the processes that happen behind the scenes (including network transport, encryption and archiving). Instead these examples concentrate on what the user actually sees as the system behaviour, with the more in-depth and technical details studied in the later chapters.

There have been nine actions identified for users, as seen in the following taxonomy:

- Execution

  Store a file

  Retrieve a file

  Delete a file

- Searching

  Individual search

  Proximity search

- Viewing

  View search results

View notifications

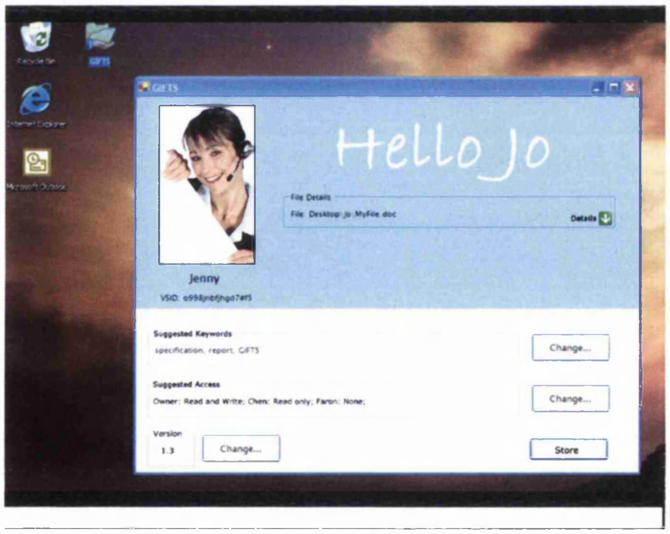
- Optional actions

  Set keywords

  Set permissions

4.4.1.1 Storing a file

A user creates a new file such as a word processing document, as they would normally. GIFS would employ a drag-and-drop interface similar to that displayed in graphical operating systems to allow the user to instruct the Virtual Secretary to store this file away. The user would drag and drop the file onto the VS/GIFS icon to store this file remotely. Depending on the settings of the VS, this action may cause an interface to appear for the optional input of more details, see Figure 4.10 for an example. In other cases, the VS may just take the file and store it without any further need for input from the user.



File Details
File: Desktop\jo\MyFile.doc

Jenny
VSID: o998jnbfjhgd7af5

Suggested Keywords
specification, report, GIFTS

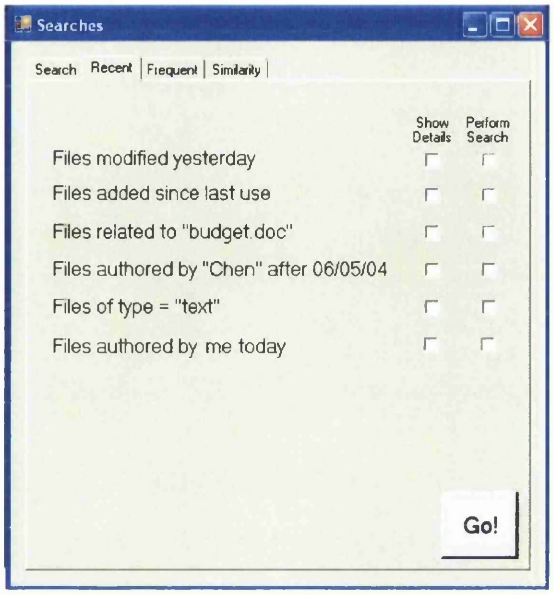
Suggested Access
Owner: Read and Write; Chen: Read only; Faron: None;

Version
1.3

Figure 4.10: The optional settings interface

4.4.1.2 Retrieving a file

Later in the day, the user wishes to retrieve the file that they created in the morning. They open the VS program and have a choice of methods for retrieval. One such technique is by using the list of recently used searches that have been generated by the VS (Figure 4.11). Upon seeing the relevant search in the most recent list, the user could select to see the results and from there mark the correct file(s). This



	Show Details	Perform Search
Files modified yesterday	☐	☐
Files added since last use	☐	☐
Files related to "budget.doc"	☐	☐
Files authored by "Chen" after 06/05/04	☐	☐
Files of type = "text"	☐	☐
Files authored by me today	☐	☐

Figure 4.11: The adaptive user interface

instructs the VS to bring the file back to the user's local machine, and open it with the appropriate program.

4.4.1.3 Storing a new version of a file

Once the user has finished editing a file, they add the file back to the system in much the same way as they did upon creation of the file. The VS handles the version control and so adding an edited file back to the system will not cause the original to be overwritten. If the user wishes to increase the version to a specific number they can do so from within the VS interface.

4.4.1.4 Deleting a file

As is the case with all the file storage mechanisms within GIFS, file deletion is out of the hands of users. If the user requests a file to be deleted, then as far as all users of the system are aware, that file no longer exists. In reality, this is not the case but will be discussed further in Chapter 7. The user is able to undelete a file in the future if they find it is still needed or was deleted in error, but up until that time the file will be unavailable.

4.4.1.5 Searching for files

There are a variety of ways that a user can locate their files for retrieval through the Virtual Secretary. If a file is seldom used, or has not been accessed for a long period of time, then it is unlikely that it would appear in either the most frequent or most recent search lists within the VS interface. When this is the case, the VS allows the user to search for a file using the details about the file that they can remember. The search options are natural and expressive allowing powerful queries. For example, a user can ask the VS to find a file that they worked on a week ago, that may have been about a committee meeting and originally written by Mr Smith by using the example interface seen in Figure 4.12. The VS could then return a list of files that most closely match the specified criteria, shown in Figure 4.13.

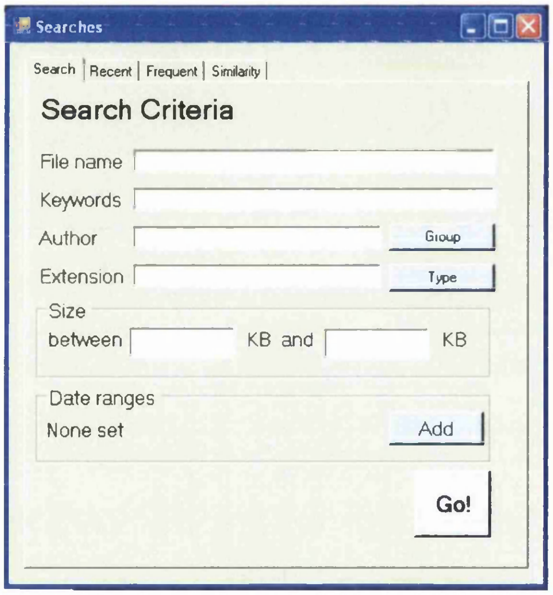
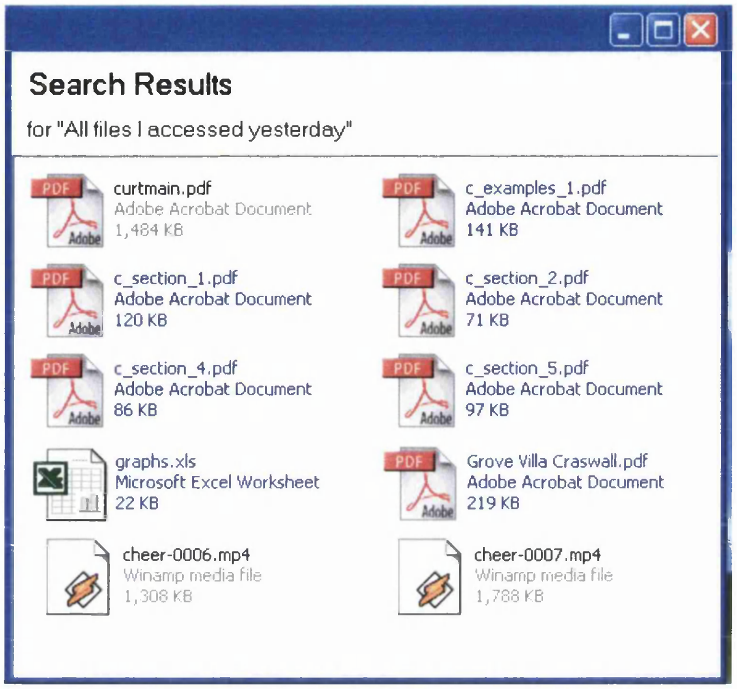


Figure 4.12: The search interface

The VS also allows proximity matching between files. For example, if the user wants to find all the files that could be related to a report they have written on the space usage of an office building, they can instruct the VS to find all files similar to this report. The results returned are not restricted to files only written by the user, as the search will span all files that the user has permission to access. Thus a complete subset of files on similar/related topics can be recovered from just one file. It is likely that the VS will also return some less-relevant files. However, over a period of time the VS will develop more knowledge that highlights the unsuitability of these files and will exclude them from the results.


Search Results

for "All files I accessed yesterday"

curtmain.pdf
Adobe Acrobat Document
1,484 KB

c_examples_1.pdf
Adobe Acrobat Document
141 KB

c_section_1.pdf
Adobe Acrobat Document
120 KB

c_section_2.pdf
Adobe Acrobat Document
71 KB

c_section_4.pdf
Adobe Acrobat Document
86 KB

c_section_5.pdf
Adobe Acrobat Document
97 KB

graphs.xls
Microsoft Excel Worksheet
22 KB

Grove Villa Craswall.pdf
Adobe Acrobat Document
219 KB

cheer-0006.mp4
Winamp media file
1,308 KB

cheer-0007.mp4
Winamp media file
1,788 KB

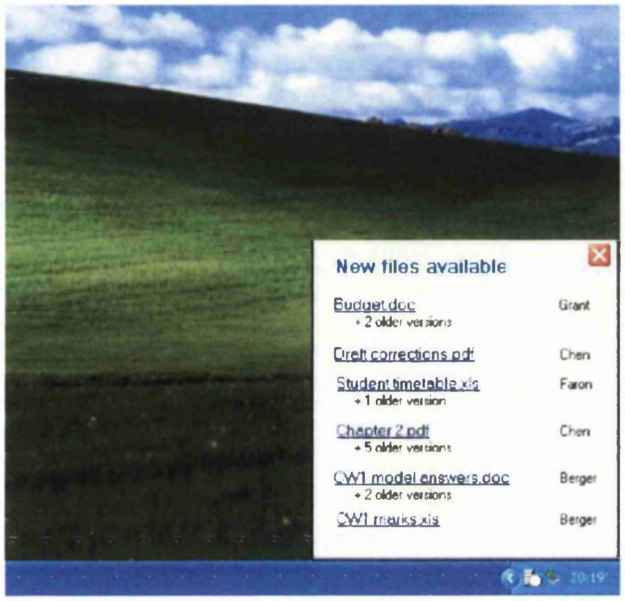
Figure 4.13: The search results interface

4.4.1.6 User notification

Files may be added to the system that are available to multiple users (such as many-authored documents, committee files or otherwise shared files). In these cases, depending on the user's preferences the VS could alert the user to the existence of these new files. This can be done in a variety of ways, ranging from a small icon appearing in the user's task bar, a once-a-day alert through the VS interface or a weekly email. (Obviously the type of alert is personal preference and the time span is dependent on the amount of traffic for a particular user. An example alert is shown in Figure 4.14)

4.4.1.7 Setting file permissions

The Virtual Secretary assumes all files are to be kept private unless it has identified an access pattern within the knowledge bases. If user wishes to share their files with others, it is simply a case of telling their Secretary which users are to be granted access, and to what level (e.g. read, write or joint ownership). The VS may make suggestions for access priviledges to save the user effort. These permissions can be changed or revoked at any time by using the optional settings interface within the VS. (Figure 4.10).


New files available

Budget.doc + 2 older versions	Grant
Draft corrections.pdf	Chen
Student timetable.xls + 1 older version	Faron
Chapter 2.pdf + 5 older versions	Chen
CW1 model answers.doc + 2 older versions	Berger
CW1 marks.xls	Berger

Figure 4.14: Example user notification alert

4.4.1.8 Setting keywords

When a text-based file is added to the system, the Virtual Secretary automatically searches through it and provides suggestions for keywords. If the file is not text-based a user can add their own personal keywords to the file by entering them into the appropriate box in the "file settings" interface. The same method is used for editing the keywords that have been suggested by the VS. Provided the user has the write permission for a file, the keywords can be altered or updated at any time.

4.4.2 Summary

The Virtual Secretary interface aims to remove from the user the burdens associated with file storage and retrieval. In order to provide the features seen in this chapter, the Virtual Secretary must be supported by a collection of agents, a set of knowledge bases and a file system. Seeing the Virtual Secretary from the viewpoint of a user gives a convenient and simplified perspective from which to start examining the system as a whole. As much of the technical detail of the operations relating to file storage is abstracted away from the user and handed to the computer to process, the way in which this data is stored and processed becomes critically important.

Chapter 5

Data and Knowledge

5.1 Introduction

The strength and perceived intelligence of GIFS and the Virtual Secretary system comes from three different sources. The *data* it collects from a user's actions, through day to day use of the system; the *information* that can then be extracted from the raw data when it is analysed; and the *knowledge* resulting from combining both the raw data and information. The task of collecting, storing and analysing large amounts of data is an expansive research topic in its own right, and this chapter covers only those parts required for the operation of the Virtual Secretary. Once the data has been gathered, it is analysed by a collection of intelligent agents, resulting in further data and information generation. From this information the knowledge that alters the behaviour and interface of the Secretary can be developed.

5.1.1 Technical Questions

File systems typically store information in the form of flat records (such as i-node tables or FAT tables). There is little use of databases in file systems, and even less use of knowledge bases. Hence, in order to realise GIFS, the main technical questions include:

- How can a knowledge representation scheme be designed so that its basic framework lasts for years, without the need of middleware or patching of the software to update it frequently whenever there is a need to introduce a new feature into the file system?
- How can each piece of knowledge be structured such that it is uniquely identifiable within the system?

- What kind of knowledge processing utilities will be needed to implement the functionality of the GIFS framework?
- How can the knowledge processing utilities be implemented efficiently in order to avoid on-demand processing for most events?
- How can the knowledge base be updated without the need for a system redesign?
- How should files be stored locally?
- How should data be collected from users?
- How can user's interactions with the file system be specified?
- How should files with multiple versions be handled?
- How should each Virtual Secretary be identified?

The above questions will be answered in the rest of this chapter, as the knowledge-based architecture is presented.

5.1.2 Assumptions

The design of the knowledge-based architecture relies on the following assumptions:

- Disk sizes will increase faster than knowledge base sizes, as will be discussed in §6.2.4.5.
- An average user will be situated within a networked environment as this is a common phenomenon nowadays, and networking capability will continue to expand.
- Users will interact with the Virtual Secretary for all their file storage and retrieval, as this presupposes that the Virtual Secretary technology proposed in this thesis will in the future replace the traditional user interface for file systems for a very large number of ordinary users. By that time only a small number of technical specialists will know how to handle traditional user interfaces (an analogy to the text-based command windows), who will also be aware that such interactions may not be transferred to the knowledge base of the Virtual Secretary.
- Users will create 5000 files a year or more as will be seen in §6.2.4.5.
- Users will access or search for some files considerably more frequently than others. This is as certain files or types of files will form part of a user's daily computing routine or be used frequently for reference purposes and so will be accessed more often than files which are created for one-off occasions.

- Users will wish to collaborate with colleagues on some files. Working in groups is now a very common task in the workplace, even for simple tasks such as asking colleagues for suggestions or improvements to a file. The increase in personal websites on the Internet also supports the assumption by proving that people wish to share their data (such as photos etc) with others.

5.1.3 Approach

To address the above technical questions, the best approach is through research and development. In this work, we conduct a relatively comprehensive design exercise that provides an overall design of the knowledge framework for GIFS, and we will selectively implement the main technical components of the framework to demonstrate the technical feasibility of this design. In particular, we present the overall design of the knowledge base in 5.3.4, and a collection of processing utilities in the form of agents in 5.5. As detailed in 5.5.2, we implemented 24 agents which covered a wide area of operations such as communication, knowledge capture, user profiling and searching. The data flow between the knowledge base and other parts of GIFS will be detailed in 7.2.1.

5.2 Acquisition of data from files via human interaction

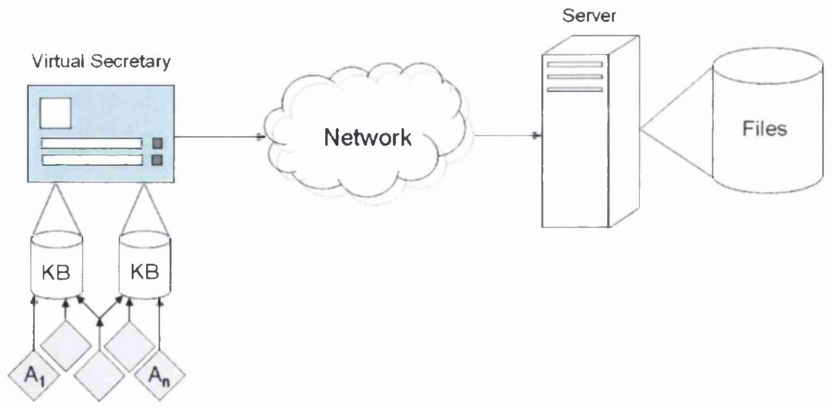


Figure 5.1: A Simplified GIFS Architecture

A user works on and operates a computer largely through the manipulation of files. In most current operating and file systems, a limited amount of metadata is stored about each file. The systems which include more metadata do not have appropriate searching facilities attached, and still only record a small subset of data. Computers are ideal for processing large amounts of data, as well as recording all actions that a user performs.

The Virtual Secretary is the user interface for GIFS. By using this interface, the computer can record a wide variety of metadata on the way a user creates, manipulates and accesses their files. Previous data-gathering or learning systems have failed as users were uninterested in investing the necessary time or effort to train their assistants. The Virtual Secretary (VS) removes this problem as every single action a user performs will expand the knowledge base the Secretary works across, without requiring extra input from the user. As seen in the previous chapter, a user can perform a variety of actions through the VS. Each of these actions will generate data, which will then be stored in a knowledge base, and processed by a collection of intelligent agents. Some of these agents will simply sort the data, whilst others will produce new knowledge to assist the system and thus the user. Before looking in detail at the architecture of the agents and interface, it is worth understanding the design principles and aims of the knowledge base.

5.3 Design of the Knowledge Base

5.3.1 Design Principles

The following design principles will be followed:

- The data collection process should be separated from the process of knowledge processing. It is a common mistake that the decision of what data to collect is based on what can be processed with the current technologies and what is considered useful in the context of one person's knowledge about applications.
- Every piece of knowledge must be time-stamped according to its creation time. This helps to ensure the provenance of each piece of data, by being well-documented and unique enough to allow reproducibility [121]. It is necessary to consider the validity of this time-stamp carefully, and how it is used, as most time-stamps will be based on local clocks. Time-stamps can be adjusted by comparison with the server clock, or a central clock such as `http://greenwichmeantime.com`. Alternatively, it could be necessary to only synchronize the clocks of Virtual Secretaries that belong to the same person across different machines, with other Secretaries clocks dealt with by an algorithm similar to one used with timing problems and synchronicity in operating systems.

- The knowledge representation scheme should not be constrained by "versions" wherever possible, especially in terms of knowledge structure, elements of attribute set, size and ordering of attribute list, etc. Hence new attributes can be introduced and old attributes can be replaced without referring to the notion of version.
- Using constants to represent attributes that are not intrinsically numerical is to be avoided.

5.3.2 Terminology

The terms, data, information and knowledge are often used interchangeably in related literature, though they often have a different emphasis:

- **Data** is the raw knowledge collected through various user interaction, and communications. We use the term "raw knowledge base" to avoid it being misinterpreted as a "database" which usually has a predefined structure that is fixed throughout the life cycle of the database (or its particular version). However, we are also aware that raw knowledge is commonly referred to "data".
- **Information** is extracted from the data, and is often an abstraction of the data. For example, an email may be considered as data or raw knowledge, the keywords, sender name, and so on that can be extracted are information. This can also be referred to as "metadata".
- **Knowledge** is commonly referred to as the information resulted from a process of reasoning and normally has more semantic meaning. For example, a link between two files called "2003budget.doc" and "2003_budget_sheet.xls" is considered as knowledge.

The term "file" is often used as a generic term for both ordinary files and folders (directories). Ordinary files may also be often referred to as documents, although this terminology is avoided where possible as whilst most computer users would understand a word processed file as a document, the same cannot be said for sound or media files.

5.3.3 Definitions

In addition to the previously presented principles, several other guidelines are used to provide a consistent and logical development environment for the knowledge bases and agent system.

Each VS has a unique identifier (VSID), which is advantageous for licensing and security reasons (for example, when VSs exchange data directly). Although it may be advantageous for each agent to have a unique identifier, they are simply given a name which is consistent under each Secretary. Thus each agent can be identified from the VSID and the agent name.

Each action is given a unique (between actions) identifier, and any other pieces of data related to that file will also have the same identifier. As each of these pieces of data refer to a different attribute, combining the attribute name and identifier gives a unique identifier across the system. The identifiers are constructed from the Virtual Secretary ID (VSID) and a timestamp. Other fields were considered for inclusion in the identifier, however they were not unique (such as author, filename and hostname), and network addresses were avoided as they are prone to change.

The structure of the knowledge bases and the knowledge representation language is a question of personal preference. There are several suitable alternatives. XML is a natural choice due to its expandable nature, and its widespread usage means that knowledge represented in such a way may have a longer shelf life. However, XML does not suit knowledge processing, including retrieval and reasoning. Formats such as Prolog predicates are much more suitable for processing. The ideal design would have XML as an external and permanent (long-term) representation, and Prolog predicates for internal and short-term representation. It would be possible to store and transmit the data using the Resource Descriptor Framework [272] model and submit queries via a query or inference language. However, most of these query languages were not developed or documented sufficiently at the time of project implementation to be considered a viable This would require a parser to translate between the XML and Prolog (a trivial piece of software). As the solution is so simple in terms of implementation, the knowledge bases have been stored as Prolog, with the communications protocols and external knowledge base communications written in XML.

For file retrieval, the Virtual Secretary creates a temporary local store whose name is derived from the current date within the user's native file system which will contain all the files retrieved on that day. When the VS is told to file the document away, it is removed from the directory. Either at regular time intervals or at the end of the day (depending on set preferences) the VS files away the files automatically.

Data, or raw knowledge, can be collected in many ways, including users' interactions with the operating system and emails. In principle, there are no constraints on what data will be collected as long as it is collectible. The first implementation of GIFS will concentrate on those data collected through users' interaction with a file system.

Definition: An *action* is an operation performed by a user, typically via a Virtual Secretary, who is then able to collect attributes related to this particular action. An action may have sub-actions. An ordered list of actions may be combined to form a super-action.

Definition: An *attribute* is a data or information entity of a specific data type (or attribute type) associated with an action. The attribute can be a distinguishable item in user's direct input (i.e. data such as filename), the status provided by the operating system (i.e. data such as a timestamp), or information extracted from the raw data associated with the action (i.e. information such as the document title from an MS-word document).

Actions considered in GIFS version 0.5 are:

- Save a file, or more precisely, instruct a VS to store away a file. This action may have the following attributes:
  - filename (direct input)
  - timestamp (OS status)
  - If the object to be filed away is a folder, the list of all files and sub-folders it contains (OS status)
  - the owner/user of this Secretary (information)
  - author(s), the default is the owner but this can be extracted from a document, its metadata, or be entered by the user (information/direct input)
  - keywords, which can be extracted from a document, its metadata, or be entered by the user(information/direct input)
  - access permissions and security level with separate read and write access, for individuals and also groups(direct input/information)
- Search a file by giving a set of search criteria, which may include:
  - date, time, size authors, title, keywords etc.
  - most accessed, or similarity to another file
- Select a file from a list of files returned by a search.
- Retrieve a file or several files.
- Delete a file or several files.
- Modify attributes of one or more files.
- Change the security password for the Secretary

Attributes that can be modified include:

- keywords
- password
- access permissions

- filename
- author
- version number
- encryption key - note this is not set by the user but can be changed by the VS when requested

Attributes that cannot be modified include:

- date of creation
- date of modification

As many or as few of these actions can be combined into one knowledge entry, although only ≤ 1 action can be selected from the set of direct actions on the file (e.g. save or delete, not both). There is no action for moving a file, as a directory/folder no longer reflects the physical organisation of files in GIFS and physically moving a file would cause problems with version control. The user has no concept of or control over the physical layout of the disk. *Folders* within GIFS are defined only as the graphical representation of a grouping of search results. Files can be renamed, although the file names set by the user are used purely as an attribute in the metadata. The Virtual Secretary has the "knowledge" that a file is renamed and thus can still keep track of previous versions.

This leads us to the question about the definition of a "version". There are several alternatives.

Definition 5A A *version* is an explicit numbering system associated with each file and it can only be changed by the user(s) who has the write access permission of the file. The number can just be an attribute, or coded into the file name.

Definition 5B There is no explicit specification of version required. The Virtual Secretary maintains the knowledge about how a file is changed or evolved. When the VS is asked about the history of a file, the VS will show a list of actions on a file and its relationships with others (e.g. open as A and saved as B, and A and B become related). When the VS retrieves a file for the user, it automatically archives it. When the file is returned, the relationship between the files is mapped if changes to the file have occurred.

Definition 5C Both 5A and 5B are available to facilitate version management.

Whilst definition 5A provides a good structural base for version control, it is too restricted and relies too much on the user versioning their own files correctly — precisely what the system is trying to avoid. Definition 5B requires a large amount of disk space if files are edited and consequently versions are changed frequently. However it may produce complications for files that can be edited by a group of people as there will be no easy way for the users to tell if they are editing the most up to date version of the file. This definition is also useful if the user accidentally destroys the most recent version of their file and provides a perhaps primitive mechanism for backing up files. Although definition 5C is unlikely to be the most elegant solution, it combines the rigid structure of definition 5A and the flexibility and vast knowledge propagation of definition 5B.

5.3.4 Knowledge Base Structure

A knowledge base is associated with a specific Virtual Secretary (VS), and contains the following general knowledge:

```
virtualSecretary(VSID, Timestamp, [Attribute List])
```

where `VSID` is the licensing/unique ID string. The same VS can be installed on different computers for the same user, the VSID will be unique in each installation but there will be an entry in the knowledge base linking each of these VSIDs so data can be shared between them with no restrictions. The VSID string should be carefully designed to accommodate free licensing, site licensing, single, and multiple installations. `Timestamp` contains the date and time when the VS was installed, as local clocks can be entirely inaccurate this should be provided by the server the first time the VS connects. The `Attribute List` is currently unused, but may be needed for future extension.

```
knowledgeCount(LastID)
```

`LastID` denotes the knowledge sequence number, which is a 16-letter string, and each letter can be chosen from an ordered set of 64 characters [#, *, Z, Y, ..., A, z, ..., a, 9, 8, ..., 1, 0]. Hence this gives us ($16^{64} = 1.158 \times 10^{77}$) combinations. If one VS can create 10^{10} pieces of knowledge each day, it will create $365 \times 100 \times 10^{10}$ $= 3.65 \times 10^{14}$ pieces of knowledge in 100 years, so this numbering system should be sufficient.

Data in the main knowledge base takes the format of:

```
action(ID, Timestamp, [SubActionList], [OList])
```

where `ID` is the identifier that binds all the sub-actions in the `SubActionList`, and the `Timestamp` is the time at which this entry was added. The `OList` contains the predicates relating to the origin and expiry of the data.

Each action that takes place should be uniquely identified not only for one user, but also across the entire system for the sharing of data. However, this is difficult to manage in a world-wide distributed system. GIFS is designed to ensure that each action is uniquely identified with a Virtual Secretary's knowledge base. Provided that each VS has a unique VSID in a domain, we can ensure that each action can be uniquely identified in the domain. It is not difficult to enforce a unique VS through a licensing mechanism. The concatenated ID (VSID-ID) format ensures that each ID is unique domain-wide.

The SubActionList takes the form of:

```
[subAction1,..., subActionN]
```

where each `subAction` has a generic form:

```
subActionName(ID, [Attribute1,..., AttributeN])
```

with `ID` being the same as the ID from the main action. This list of sub-actions is fully expandable.

The OList has the form:

```
oList(ID, origin, [Attribute1,..., AttributeN])
```

The `origin` tag merely denotes whether or not this data was created autonomously by an agent or with user input. If the data was agent created, then the attributes in the list will be filled with details on that agent, e.g. the name of the agent and the ID of the actions that were used in the deduction.

A file storing request activated through a VS may result in the following knowledge to be added into the VS's knowledge base:

```
action(0000000000000001, 1158161802, [FileAway, OriginList])
fileName(0000000000000001, "MyDocument.doc")
fileType(0000000000000001, "MS-Word")
fileSize(0000000000000001, 128698)
fileTimeCreated(0000000000000001, 1158161802)
fileLastAccessed(0000000000000001, 1158164523)
fileLastModified(0000000000000001, 1158164523)
userSetVersion(0000000000000001, 1)
keywordsVS(0000000000000001, ["specification", "report"])
keywordsU(0000000000000001, ["specification", "GIFS"])
access(0000000000000001, UserAID, read, write)
access(0000000000000001, UserBID, read, write)
access(0000000000000001, UserGroupA, read, noWrite)
oList(0000000000000001, user, [all])
```

Due to the extensible design of the knowledge base, further knowledge can be added in later. For example, the following data shows how keywords found by some agents using new software programs could be incorporated:

```
action(000000000000008b, 1158169534, [Agent, File])
agent(KeySearch, [Software])
software(000000000000008b, [KeySearchLite, MS-WordProfile])
```

```
refAction(000000000000008b, 0000000000000001)
keywords(0000000000000001, 000000000000008b, [budget,...])
```

Search operations produce an action entry in the following format:

```
  action(ID, Timestamp, [Search, Results],[OList])
```

where `ID` is the identifier for this action, `Timestamp` is the time at which it was initiated and `OList` contains the origins of the data as before. The `search` predicate takes the following form:

```
  search(ID, [SearchCriteria])
```

with the `SearchCriteria` variable holding all the criteria set by the user for this specific search. The results of each search are stored in the format of:

```
  results(ID,[[FileID1,Score1],[FileID2, Score2],
          ...,[FileIDN, ScoreN]])
```

where `[FileID(1...N)]` are variables holding each file ID that was included in this search and `[Score(1...N)]` holding the corresponding scores.

The reverse of the FileAway action, Filehome retrieves a file for the user.

```
  action(ID, Timestamp,[File,Filehome], [OList])
```

where `ID, Timestamp, File` and `OList` follow the same format as in previous predicates and the `Filehome` predicate has the structure:

```
  filehome(ID, [RemoteServer, LocalPath])
```

where `RemoteServer` contains the name and network address of the server that the file was retrieved from and `LocalPath` contains the temporary location of the file on the user's local machine.

As GIFS is designed to protect users and data from deletion errors, although from the point of view of the user the data is deleted, from the point of view of the system it is archived. Thus the following data is added to the knowledge base:

```
  action(ID, timestamp, [File, Delete],[oList])
  delete(ID, FID)
```

5.4 Life cycle of data and knowledge

A knowledge base is associated with a specific Virtual Secretary, that is, a VS interface servicing a specific user. Each knowledge base, `K`, is composed a set of knowledge modules, `K1, K2, K3, ..., Km`. In a distributed VS environment,

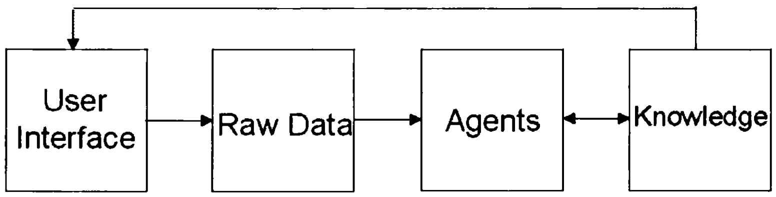


Figure 5.2: The life cycle of data

modules can be located on different computers, and are maintained, sometimes individually and sometimes collectively, by a group of knowledge management agents. The modular structure of a knowledge base facilitates the ownership of knowledge stored in each module, and a unique identification of each piece of knowledge within the module. A knowledge module, `Ki`, contains the following general knowledge:

```
virtualSecretary(VSID, Timestamp, Attribute_List)
knowledgeCount(LastKSN)
```

where `VSID` is a licensing string, `Timestamp` is the date and time when the corresponding VS was installed, and `Attribute_List` is an extensible list of attributes characterising the VS software. The predicate `KnowledgeCounter(LastKSN)` maintains a counter for knowledge sequencing numbers, each of which uniquely identifies a compound knowledge instance, which will be detailed in the following subsection.

5.4.1 Compound Knowledge Instances

A Compound Knowledge Instance (CKI) is one or several pieces of knowledge gathered or generated in the context of a specific action or task, such as filing away a document. Each CKI is represented in a knowledge module as a collection of Prolog predicates in the following general format:

```
mainKI(KSN, Timestamp, SubKI_List)
<subKI1>(KSN, ...)
<subKI2>(KSN, ...)
```

where `KSN` is the knowledge sequencing number that binds principal knowledge instance (`mainKI`) with all the supplementary knowledge instances (subKI's) in the `SubKI_List`. A `SubKI_List` consists of a list of names (called functor in Prolog) of supplementary knowledge instances `[<subKI1>, <subKI2>, ...]`. The ordering of these functors is insignificant in a CKI, hence the format of each type of CKI in terms of the number of attributes can evolve freely without necessity for defining a version in every evolution stage. The unification capability of logic programming enables pattern matching with little programming effort.

5.4.2 Example of Data

Listing 5.1: An example of a data created through the addition of a file

```
mainKI(0000000\#00000001, 1158169534, [file, fileAway])
file(0000000\#00000001, [fileName, fileType, fileLocalPath,
          fileSize, fileTimeCreated, fileLastAccessed,
          fileLastModified, userSetVersion, keywords,
          access])
fileName(0000000\#00000001, "MyDocument.doc")
fileType(0000000\#00000001, "MS-Word")
fileSize(0000000\#00000001, 128698)
fileTimeCreated(0000000\#00000001, 1158169534)
fileLastAccessed(0000000\#00000001, 1158175391)
fileLastModified(0000000\#00000001, 1158175391)
userSetVersion(0000000\#00000001, 1)
vsVersion((0000000\#00000001, 1)
keywordsVS(0000000\#00000001, ["specification", "report"])
keywordsU(0000000\#00000001,["specification", "GIFTS"])
access(0000000\#00000001, 9lksjlkjslf9840, read, write)
access(0000000\#00000001, exampleUser1, read, write)
access(0000000\#00000001, exampleUser2, read, nowrite)
access(0000000\#00000001, group1, read, write)
access(0000000\#00000001, group2, read, nowrite)
group1(0000000\#00000001, [exampleUser3, exampleUser4])
group2(0000000\#00000001, [user1, user4, user5])
```

In the example seen in Listing 5.1 the principal knowledge instance (`mainKI`) gives the functors of two supplementary knowledge instances, `file` and `fileAway`. The `file` predicate provides a further list of supplementary knowledge instances for a range of file attributes. The `fileAway` predicate records a set of operations performed for the task. Some of the above knowledge (e.g. `fileName`), is gathered from user interaction by the Virtual Secretary; some (e.g. `group2`) obtained from previous knowledge in the knowledge base; and some (e.g. `keywordsVS`) generated by relevant agents. However, a substantial amount of the knowledge (e.g. `fileSize`) is acquired from the operating system by the VS. Hence the overall burden for the user to provide the Virtual Secretary with raw knowledge is very limited.

5.5 Agents for Knowledge Capture and Manipulation

5.5.1 Overview

Within GIFS, agent programs perform a variety of tasks in the background to support the Virtual Secretary interface. As seen in Chapter 3, artificial intelligence still has a long way to go before it can produce a system that contains common sense or is intelligent in the classic sense of the word. By using many small agents in GIFS that each perform very specific roles a useful service can be maintained without the worry of creating one large "intelligence".

Some agents within GIFS are responsible for communications and interfacing with other parts of the system, whilst others solely perform simple deductions or calculations in order to help a more complex agent. Primarily, the agents operate over the raw data in the knowledge bases that is produced by the user interacting with the Virtual Secretary. By using deductive reasoning new knowledge can be created which is used both by the Virtual Secretary interface and can also then be analysed by further agents. This is a continual process as data is added with feedback from the Virtual Secretary and new knowledge is created.

5.5.2 Classifications of Agents

There are 3 main classifications for agents within GIFS. These classifications are broad and not necessarily unique (e.g. it is possible given these classifications that an agent may belong to more than one group).

- Communications: Agents whose main task is to provide communications services to the system
- Processing: Agents whose main task is to perform calculations which are necessary for the more complex tasks
- Creation: Agents whose main task is to analyse the knowledge bases and create new knowledge

A list of the agents and their main types can be seen in Table 5.1. It is worth noting that almost all of the processing agents also display the behaviour of a creation agent. When their operations are performed the results of the intermediary calculations are stored within the knowledge bases. It is this data which is then combined or used by a creation agent in order to deduce further knowledge.

Name	Type
Archive	processing
Author	processing
AuthorList	creation
Favourites	creation
Filename	processing
FilenameList	creation
Friend Profiling	processing
Interface comm	communication
Keyword	processing
KeywordList	creation
Knowledge Transplant	communication
Permission	creation
Recent	creation
Search updater	processing
Similarity	processing
Time and Date	processing
TimeList	processing
Type	processing
TypeList	creation
Union	processing
User analysis	processing
Vars	processing
Version	processing
VersionList	creation

Table 5.1: List of agents

5.5.2.1 A Brief Description

Archive: The Archive agent is responsible for reducing the size of the working knowledge base. It will move any data that is obsolete (e.g. when a file has been "deleted" by the user), unlikely to be of use (e.g. when a file has not been accessed for over a year) or is superfluous (e.g. there are over 5 versions of a file, with earlier versions not being accessed for over 3 months).

Author and **AuthorList:** The Author agent has two main functions. Given a file, it can compute a similarity score (as a percentage) of that file based on author by comparing with all other files. Alternatively, given a file and a list of files, it will compute the similarity score of the author for all the files in the list. In comparison, the AuthorList agent maintains lists of files, categorised by author.

Favourites: The Favourites agent maintains a list of the search operations requested and performed by the user most often.

Filename and **FilenameList:** The Filename agent computes the similarity of all files to a given filename and returns a score between 0 and 100. It also calculates this similarity score given a filename and a subset of files for comparison. The FilenameList agent maintains and returns lists of files ordered by filename. As these agents work on a text-based field, they make use of the external libraries provided by the communications agent.

Friend Profiling: The Friend Profiling agent creates a list of the most contacted users. For example, those with which knowledge is shared via the knowledge transplant agent and those whom files are shared with. It is also responsible for the maintenance of groups for the purposes of file sharing.

Interface comm: This agent provides communications services between the Virtual Secretary and the agents (where needed). More importantly, it also acts as a bridge between the agents and external third party programs (such as those used for text matching) which have been written in alternate languages.

Keyword and **KeywordList:** The keyword agent calculates a similarity score for all files, given a set of keywords. It can also perform the same calculation over a given set of file IDs, returning a percentage score between 0 and 100. The KeywordList agent maintains lists of files sorted by keywords. Both agents use the external libraries for string matching.

Knowledge Transplant: The Knowledge Transplant agent handles the incoming and outgoing of knowledge to and from other Virtual Secretaries.

Permission: The Permission agent maintains the suggestions for file permissions by working together with the user analysis agent.

Recent: The Recent agent maintains a list of the most recent search actions performed by the user.

Search updater: The Search updater keep the lists of precomputed search results up to date either by running at a specified time interval or as the contents of the knowledge base changes.

Similarity: The Similarity agent is the main agent for coordinating the search results when comparing a file to others. It calls other agents to provide it with similarity scores before collating and ordering the results to be returned.

Time and Date and **TimeList:** The Time and Date agent performs similarity testing over either a set of or all files using three different functions - linear, polynomial and bounded cosine function. It returns the results as a list of scores between 0 and 100. The TimeList maintains lists of files, grouped and ordered by time in periods of 1 day.

Type and **TypeList:** The Type agent computes a similarity score between 0 and 100 for a given file over all other files or a subset of files. The score will be 100 for an exact match (e.g. "doc" and "doc"), 50 for a type match (e.g. "jpg" and "gif") and 0 if there is no match. The TypeList agent maintains the lists of files categorised by extension.

Union: The Union agent is used solely by other agents to perform unions or intersections over sets of results to give a combinatorial set.

User Analysis: The user analysis agent calculates the probabilities of a user being granted access (read, write or co-author) to a file.

Vars: The Vars agent is used for directly querying the knowledge bases in order to retrieve file metadata such as names, sizes and types. It is not used in any of the searches by the interface uses it extensively when displaying results or other options.

Version and **VersionList:** The Version agent computes the similarity between one file's version and all other files. The score is returned as a value between 0 and 100, and is largely dependent on the range of versions that are available. The VersionList agent maintains lists of all files ordered and sorted by version number, as well as per-file lists containing the numerical representations of each file.

5.5.3 Agent Creation

All the agents listed above have been written in Sictus Prolog 3.12.0. As the knowledge bases were written in a Prolog/XML extensible style language, Prolog seemed the most natural choice as an implementation language. In previous versions of GIFS, the agents utilised a pre-defined architecture called "The Open Agent Architecture" [68] which required Sicstus Prolog to run. However, as implementation and research progressed, it became apparent that OAA was not a suitable choice. Although well documented and researched, the OAA architecture introduced too many overheads for a large scale agent network and too many complexities for a smaller sized research project implementation. The agents were thus slightly rewritten to function without the help of aforementioned architecture, but remained in Sicstus Prolog.

For certain tasks (e.g. string matching), it was more efficient to use a 3rd party set of functions that were readily available than to write them from scratch. The string matching function that was used was called FuzzySearch written by SoftComplete Development [1]. It provides a variety of fuzzy matching algorithms to be used over strings. The source code for these functions had been written in C++ so first a C wrapper program had to be written. A DLL was then created to allow Prolog to call the functions referenced in the C code by using the Prolog $\rightarrow$ C bridge. In this manner the Prolog agents were able to call the string matching functions that were provided in the 3rd party C++ code.

5.5.4 A Closer Look

In the following section four agents are looked at in more detail. They provide a range of services in order to facilitate the Virtual Secretary interface as well as supporting the operations of other agents.

5.5.4.1 Filename Agent

The filename agent compares a file name to all the other file names in the knowledge base in order to help calculate the proximity score of all files. It uses 3rd party fuzzy matching algorithms to generate a percentage score denoting the similarity of all filenames in the knowledge base to a given filename. For example, the filenames "mydocument" and "yourdocument" would return a high similarity score, whereas the filenames "mydocument" and "cabbages" would result in a low similarity score.

Listing 5.2 displays a section of code from the filename agent. The predicate `comparefilename/2` which starts on line 1 takes a filename `N` and returns a set of `(FileID, Score)` tuples in `Results`. Line 2 shows the agent first building a set `S` of all the filenames and corresponding file ids that a user has access to. Once the list has been built, the data is extracted into two singular lists: `Abs` containing the filenames and `Files` containing the file ids. Line 4 performs the calculation of the proximity score of each filename in `Abs` to the filename `N`, with the results stored as a list in `Scores`. Finally, line 5 contains the call to the predicate which builds the results into a list which has the format of `[(FileID1, Score1),...,(FileIDN , ScoreN)]`.

[1] `http://www.softcomplete.com`

Listing 5.2: A predicate within the filename agent

```
comparefilename(N, Results) :-
            setof((ID,Name), fileName(ID, Name), S),
            makelists(S, Files, Abs),
            doclever1(N, Abs, Scores),
            rebuildlists(Files, Scores, Results).
```

5.5.4.2 User Analysis Agent

The user analysis agent performs many mathematical calculations over in order to provide the individual probability scores of file access permissions to the permissions agent. These values are calculated by using agent-created knowledge that has been extracted from the raw data of user actions and interactions with the Virtual Secretary. The code in Listing 5.3 shows one such calculation which processes the data of previously made correct suggestions. If a user has been suggested to have read access previously and this suggestion has been accepted by the user, the returned score will be high. Alternatively, if this user has never been correctly suggested to have write access for this file then the score will be 0.

One of the definitions for the predicate `crrs/4` is included in Listing 5.3. It takes a user ID `UserId`, the number of previous versions of a file `Vno`, a list of file ids for each previous version `Vlist` and returns `Result`, a positive number. In lines 3 and 4 all the previous correct suggestions for a user `UserId` to have read access for previous versions of this file are identified. The relevant knowledge ids are then stored in `S`. Line 5 shows a cut which prevents Prolog from backtracking and performing the `setof/3` operation again if a future line fails. Lines 6–9 form an if-then-else statement. If set `S` contains $\geq$ 1 element (e.g is not empty) then the result returned is the length of `S` divided by the total previous versions, `Vno`. If `S` is empty, the score assigned to `Result` will be 0.

Listing 5.3: A predicate within the User Analysis agent

```
   crrs(UserId, Vno, Vlist, Result) :-
       (
           setof(ID, (ID^(correct(ID, _, Rlist, _,_)),
               member(ID, Vlist), member(UserId, Rlist)), S),
5          !,
           length(S, N),
           Result is N / Vno
           ;
           Result is 0
10     ).
```

5.5.4.3 Search Updater

The search updater is a more complex agent than those already seen above. This is because not only is it required to call the processing agents to perform searches, but it must also cache and sort the search results, ensuring these results are up-to-date by running at a predefined time interval or as new data is added to the knowledge bases. By working in collaboration with the recent/favourites agent, it calculates the results for these searches even when not requested directly by the user so that should the user request these results they are already up-to-date.

When the user requests a search, the Search updater first checks the cache of search results. If the Search updater is running continually it will already have the most recent results for all the searches previously performed in a knowledge base as a complete set (so requesting N results and then N+5 results will not require the whole search to be performed again). If this particular search has never been run before, operation is passed over to the search builder agent.

When any new data is added to the knowledge base the search updater agent works down the list of most frequent and most recent searches and calls the respective processing agents to re-perform each search so that the new data will be included. Listing 5.4 shows a section of code from the search updater agent. In this small segment, the agent takes a set of search criteria `S`, and locates the time that this search was last performed `T`, and the corresponding results `R` by using the predicate `getsearch/3` seen on lines 1–4. Once the previous search has been found, all new files that have been added since are identified using the `setof/3` predicate on lines 8 and 9, and sent as a reduced search space `Files` to one of the search agents.

Listing 5.4: A predicate within the search agent

```
getsearch(S, R, T) :-
            setof((ID, S, Results, Timestamp),
              search(ID, Timestamp, S, Results), Set),
            maximum(Set, (x,x,x,0), (ID, S, R, T)).
5
findnewdata(S, R, Files) :-
            getsearch(S, R, T),
            setof((ID, Time), fileLastModified(ID, Time),
               AllTimes),
            picktimes(T, AllTimes, Files).
```

5.5.4.4 Permissions Agent

The permissions agent maintains the lists of predictions of file access control lists. It works in conjunction with the user analysis agent to provide the user with a suggestion of whom should have what permissions for any given file. Once the suggestion has been created it is then fed back into the Virtual Secretary interface to be presented to the user.

Listing 5.5: A predicate within the permissions agent

```
calcr([H|T], Vno, Vlist, Filewords, Now,
    N1,N2,N3, N4, N5, N6, N7, N8, T1, T2, FID) :-
            cs(H, Vno, Vlist, S1),
            pas(H, Vno, Vlist, S2),
            crrs(H, Vno, Vlist, S4),
            wrrs(H, Vno, Vlist, S6),
            ols(H, Vlist, Now, S7),
            las(H, S5),
            wlsr(H, Filewords, S3),
            (
                S3 > 0,
                !,
                S3a is S3/100
                ;
                S3a is 0
            ),
            Pnew is (S1*N1)+(S2*N2)+(S3a*N3)+(S4*N4)+(S5*N5)
                -(S6*N6)-(S7*N7),
            k(K),
            (
                verlist(FID, _, [FID, T|_]),!,
                p(_, T, H, read, Pold)
                ;
                Pold is 0
            ),
            R is ((1-K) * Pold) + (K * Pnew),
          (
                R>=T1, !,
                assert(predict(Now, FID, H, read, certain, R
                   ))
                ;
                (
                    R>=T2,!,
                    assert(predict(Now, FID, H, read,
                       perhaps, R))
                    ;
                    true
                )
            ),
         calcr(T, Vno, Vlist, Filewords, Now,
                N1,N2,N3, N4, N5, N6, N7, N8, T1, T2, FID).
```

Listing 5.5 shows a section of code from the permissions agent which calculates the probability score of a user being granted read access to a file. The predicate `calcr /16` takes a list of users `[H|T]`, the number of previous version of this file `Vno` and a list of their ids `Vlist`. `Filewords` is a set of strings which have been identifed as related to this file, `Now` is the current timestamp, whilst atoms `N1..N8` contain personality values of the Secretary and `FID` is the file id of the newly-added file.

The individual scores for each category `S1..S7` are calculated by the user analysis agent on lines 1–9, and are then adjusted according to the Secretary's personality settings (lines 17–18) and the scores of previous comparisons (lines 21–26). If the resulting score is over a threshold value (line 29) then that user is added to the list of those who are likely to have read permission.

If the score is below the highest threshold but above the second, the user is added to a possible list (line 32). If this user has score below both thresholds no action is taken and the agent processes the remaining users in the list `T` (lines 38 and 39).

5.5.5 Discussion

This chapter has defined the knowledge framework used to support the Virtual Secretary. The technical questions presented in §5.1.1 have all been addressed in the design, and as such the feasibility of the design has been demonstrated. In addition to illustrating the data that is collected from the Virtual Secretary's interactions with the user, the internal format and design principles of the knowledge base were presented. A series of agents were described that operate over this data in order to provide dissemination and proliferation of knowledge. Several agents were examined in detail to give a view of the processes that run in this system behind the scenes. In the next chapter the feasibility and scalability of this adaptive service is demonstrated.

Chapter 6

Case Studies

6.1 Introduction

In order for GIFS to provide a beneficial service to users, it must assist in the day-to-day file organisational activities without hindering the performance of the computer. In other words, the system should remove the burden of file storage and organisation from the user. Such improvements would be worthless if the system became unusable after a period of time due to the amounts of data being produced and analysed, or if the system was not capable of providing helpful suggestions to the user after analysing previous data. Therefore to prove the feasibility of the Virtual Secretary and knowledge-based approach for file storage presented previously, a variety of simulation tests over computer-generated datasets are needed to demonstrate the two major performance issues of this work: *Scalability* and *Adaptability.*

6.2 Case Study 1

6.2.1 Hypothesis

With appropriate techniques, the average search time for a Virtual Secretary can be achieved in a scalable manner.

The amount of data gathered and produced by the GIFS framework over a long period of use means that in order to provide a useful service, the Virtual Secretary must be able to return search results in a time which has a slower rate of increase proportional to the size of the data set. In order to be considered scalable, the time taken as a function of the size of the dataset must not grow faster than a polynomial of small degree.

6.2.2 Assumptions

The following assumptions are made for the purposes of this case study:

- File sizes will continue to grow at a rate of 100KB per year on average and this assumption will be discussed in detail in §6.2.4.2 and §6.2.4.3.
- Disk sizes will continue to increase by 40% per year, as will be seen in §6.2.4.5.
- The size of the knowledge base for a typical Virtual Secretary will increase by 70,000 entries per year in a best-case scenario, as will be explained in §6.2.4.1.
- The size of the knowledge base for a typical Virtual Secretary will increase by 300,000 entries per month in a worst-case scenario as will be discussed in §6.2.11.
- A typical user will have some knowledge of the file they wish to search for. It would be impossible for a user to try and search for a file that they had absolutely no knowledge of in either paper-based or current electronic systems. Therefore it is assumed that the user will have knowledge of some subset of attributes about a file but not necessarily complete information.
- A typical Virtual Secretary will search the entire knowledge base unless otherwise instructed and will return an ordered list of the complete results. This is due to the implementation of the search function and will be discussed further in §6.2.7.
- A user can select to only view the top specified number of results. Users are unlikely to want to review the score for every single file in the system when the file they are looking for would be more likely to be near the top of the results.

6.2.3 Approach

In order to explore the hypothesis, data was collected and reviewed to support the above assumptions. As well as a review of the literature of previous user and disk studies, data was *collected* from anecdotal interviews with colleagues. Another approach used was that of *simulation*. Given the amount of time a study would need to be conducted in order to run over appropriately sized datasets, data had to be created using a simulation program. Both approaches were used in this instance to provide evidence in order to support the assumptions and allow the case study to be conducted.

6.2.4 Scalability of the Knowledge Based Approach

Collecting large amounts of data over an extended period of time is not likely to result in performance advantages for the user if the knowledge bases are so large that searching through them takes hours. Whilst waiting a couple of seconds for some search results or for a file to be located may seem reasonable for a day-to-day activity (some people may argue that this would still be a frustrating wait), if performing the same action on a much larger knowledge base took 3 hours, the user would be unlikely to use the system at all.

6.2.4.1 General Problem of Knowledge Base Growth

Each file addition from a user generates between 10 and 14 entries of raw data in the knowledge base (depending on which attributes are set). A survey of fellow postgraduate students showed that at the time of writing, they each created around 5000 files per year. This would produce between 10*5000 and 14*5000 entries in the knowledge base (50,000 and 70,000 respectively). Over a period of 10 years, this equates to at least 700,000 pieces of data created entirely from file additions. Data would also be created from file deletions, changes in permissions and groups, as well as the extra knowledge created by the agents on analysis of this raw data. Thus it can be expected for this number to double in order to include these extra actions. Over a period of 50 years it would not be unexpected for over 7 million pieces of data to be created.

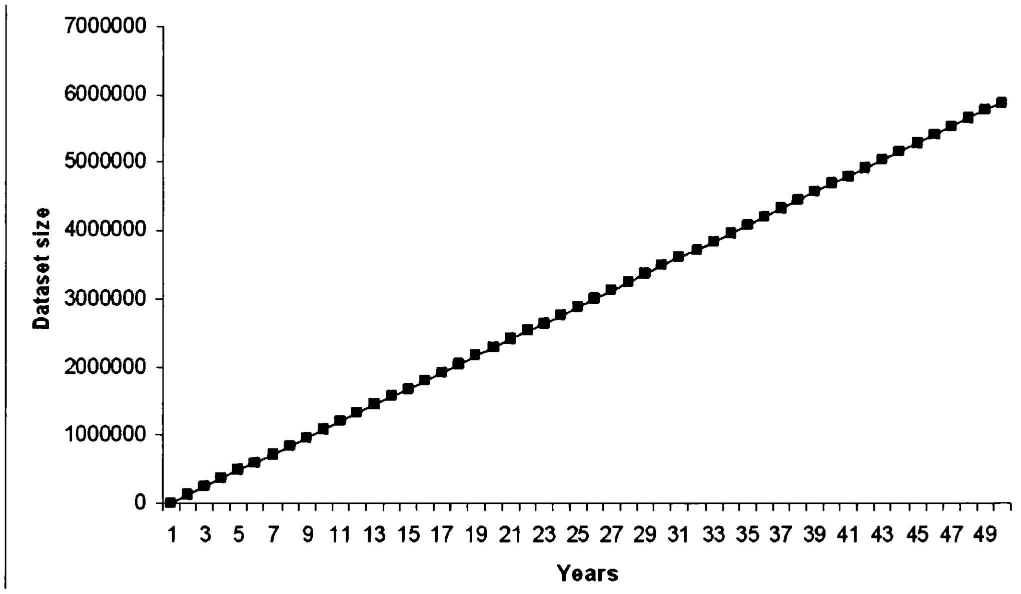


Figure 6.1: Dataset growth over a period of 50 years

Year	Files	Users	Files per user	Source
1981	86000	-	-	[248]
1984	19978	-	-	[200]
1991	304847	200	1524	[32]
1994	23000000	1845	12466	[263]
1994	429995	7500	57	[113]
1998	-	-	24000–45000	[293]
1998	140000000	10568	13247	[87]

Table 6.1: File and user statistics from previous studies

6.2.4.2 Previous Studies on the Growth of Files

The numbers represented in Figure 6.1 show the best-case scenario as it assumes that as time continues users will not create many more than 5000 files per year. There is little or no data available on the number of files belonging to a user over a period of time. Whilst there have been many studies into file system performance [248, 266, 18, 264, 200, 256], only a few of them included details as to the number of files examined in the study and the number of users that those files belonged to [87, 32, 113, 293, 263]. The studies were all performed on differing types of file systems (local, server and distributed) running on a variety of operating systems mostly by taking a snapshot of the file system status. The data can be seen in Table 6.1. Whilst these numbers certainly show an increasing trend in the number of files per user on a file system, the differences in data gathering techniques, file and operating systems studied and the lack of data over multiple snapshots means that these results do not give a sufficient basis to form generalisations for future file statistics.

6.2.4.3 Swansea File Growth Data

Instead, it was decided to analyse file usage statistics from the servers of the computer science department. The server, named "cs-svr1" services the academic staff, support staff, clerical staff, research assistants and postgraduates of the computer science department, currently totaling 161 users in all. By executing a simple script, the server administrators were able to provide a breakdown on the number of files per user and associated timestamps over a period of several years. The file timestamps can be altered not only by the file system, but also any program that uses the files on the client machine. Thus if the client machine or process has an incorrectly set time, so then will the i-node data for that file. Some files were found to have modified times of 1901 or 2028, which can be attributed to the wrapping around of the POSIX time system (which counts the number of seconds since the 1st January 1970). Entries with obviously incorrect timestamps were removed from the sample,

Year	Files
1996	17865
1997	38037
1998	71586
1999	58221
2000	62983
2001	113269
2002	167176
2003	335375
2004	273768
2005	403511
2006	390142*

Table 6.2: Last modified dates of files of all users on cs-svr1

Year	Files	Avg file size (bytes)
2001	23816	26806
2002	37651	17725
2003	40759	34554
2004	46466	125776
2005	46643	215023
2006	74150*	360792

Table 6.3: The last modified file times of User A on cs-svr1

the remainder of the results can be seen in Table 6.2.

Of the 161 users identified, the number of files per user ranged between 1 and 256853. Whilst all members of staff and postgraduates within the department are granted access to cs-svr1, many also have their own personal machine which some users prefer to use instead of storing their files on the network. The most active user (e.g. had the most files) was then identified and their file statistics individually examined to show the number of files they worked on during a year and the total sizes of files per year. This particular staff member, referred to as user A, joined the department in 2001 so there is no file data beforehand. The results can be seen in Table 6.3 and Figure 6.2.

As these figures clearly show and as supported by previous works, the number of files used per year increased. These statistics were created mid-way through a year, so the final result was doubled to allow for the remaining 6 months (marked with a '*'). By extrapolating the line seen in the graph, it can be estimated that in 50 years time, over 3 million files could be created by a user per year.

Following on from the estimate of number of files, a fifty year knowledge base of raw file data alone could contain 1,000,000,000 (one billion) entries. This is still

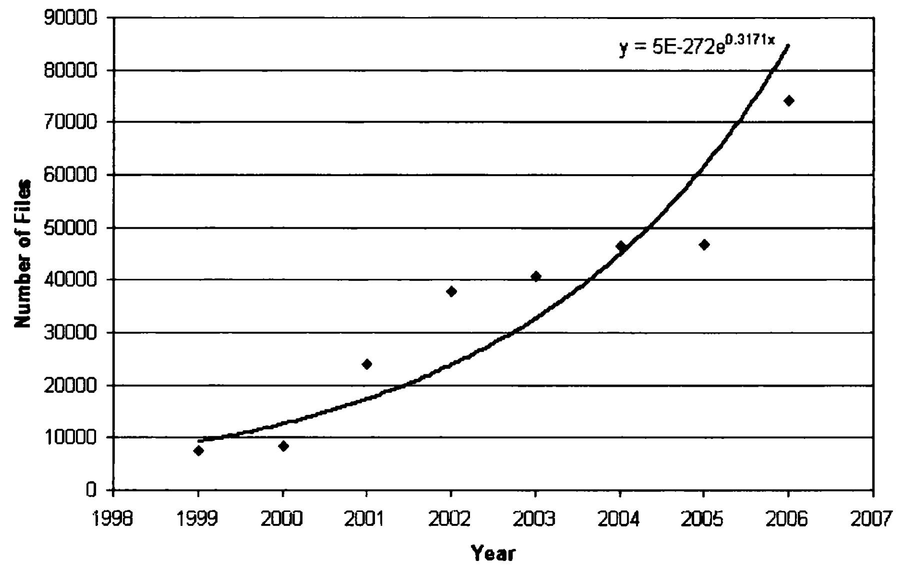


Figure 6.2: Number of files modified per year by UserA

a best case scenario, as it is likely that as the system evolved more data would be gathered to assist in the knowledge processing activities.

6.2.4.4 Increase in File Sizes

Table 6.4 contains the average file size in bytes of all files on cs-svrl. Although it shows a steady increase in growth from 1990–2003, there is a sharp increase over the last 3 years. As the number of total files on cs-svrl has increased at a more steady pace seen in Table 6.2, average file sizes doubled between 2003 and 2004, and have almost doubled again between 2004 and 2006.

In order to examine the changes in the sizes of files over a period of years, a second set of user file statistics was selected. User S is one of the longest serving members of staff in the department and has also used cs-svrl for primary file storage so provided an acceptable spread of data over a number of years. As seen in Table 6.5 and Figure 6.3 it is unsurprising that the average file sizes have increased since 1994. In comparison to the overall increase in average file size on cs-svrl, user S has a relatively small average file size, but one which still follows the trend of increase. The introduction of larger disks, more powerful computers and larger storage formats (such as unicode) have all contributed to this increase.

Year	Avg file size (bytes)
1990	7045.6
1991	20265.4
1992	17477.1
1993	15005.5
1994	18044.9
1995	91175.9
1996	36321.1
1997	43270.3
1998	54980.8
1999	66400.1
2000	112891.6
2001	88426.6
2002	93614.0
2003	92144.9
2004	180495.3
2005	235763.4
2006	309444.1

Table 6.4: The average file sizes of all files on cs-svr1

Year	Files	Avg file size (bytes)
1994	254	6443.8
1995	827	15631.1
1996	516	10101.6
1997	617	14359.9
1998	1794	8362.1
1999	3763	18421.8
2000	755	12918.6
2001	2285	15086.8
2002	2851	20271.8
2003	3082	31195.7
2004	3070	38954.7
2005	3960	45453.3
2006	1996	46087.8

Table 6.5: The number and average size of files for User S on cs-svr1

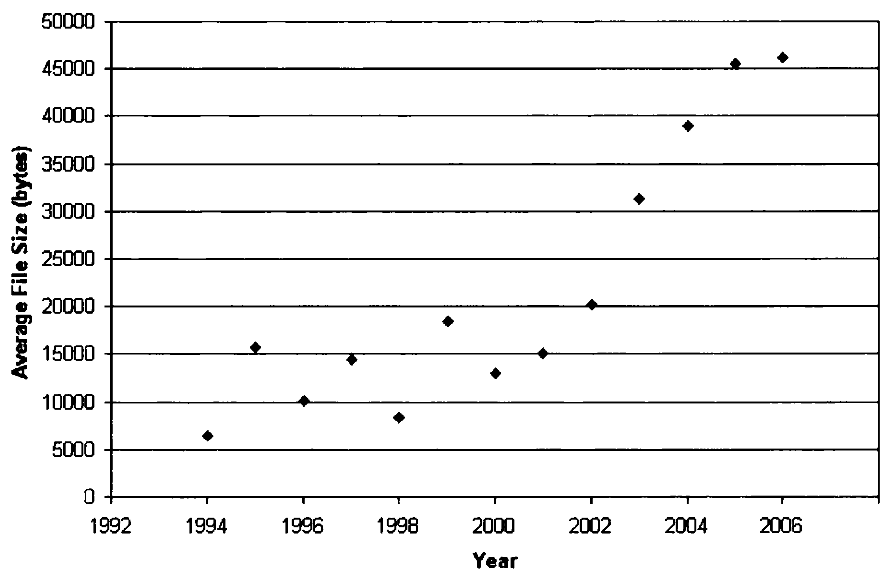


Figure 6.3: Average file sizes for user S

6.2.4.5 Increase in Disk Sizes

The development and capacity of hard disk sizes is considerably better documented. Figure 6.4 shows the capacity of disk drives over time, with a logarithmic scale. The data for this graph was extracted from [262], and shows a trend of exponential growth possibly due to the utilisation of error correcting codes and the magnetoresistive effect [133]. However, in the last few years the sizes of new disk drives have plateaued noticeably. Therefore, a more realistic prediction of disk sizes in the following years comes from Seagate [66], who predict a 40% increase in disk capacity per year. Following this prediction, terrabyte disks will be commonly available in the next 2 to 3 years, and the capacity in 50 years will be over 7,000,000,000 GB.

As the projected development of disk sizes is well over the projected knowledge base size, disk space can be disregarded as a problem in the GIFS framework.

6.2.4.6 File Growth Simulation

As it is not practical for the purposes of this project to use real data gathered over an extended period of time (as no such data is currently available), a data creation program was implemented to assist in the testing process. Although it is near impossible to accurately predict numbers involved in the future of computing, by using some arithmetics and probabilities the data creation program generated what can be considered as accurate as possible. Within this test, the size, volume and general spread

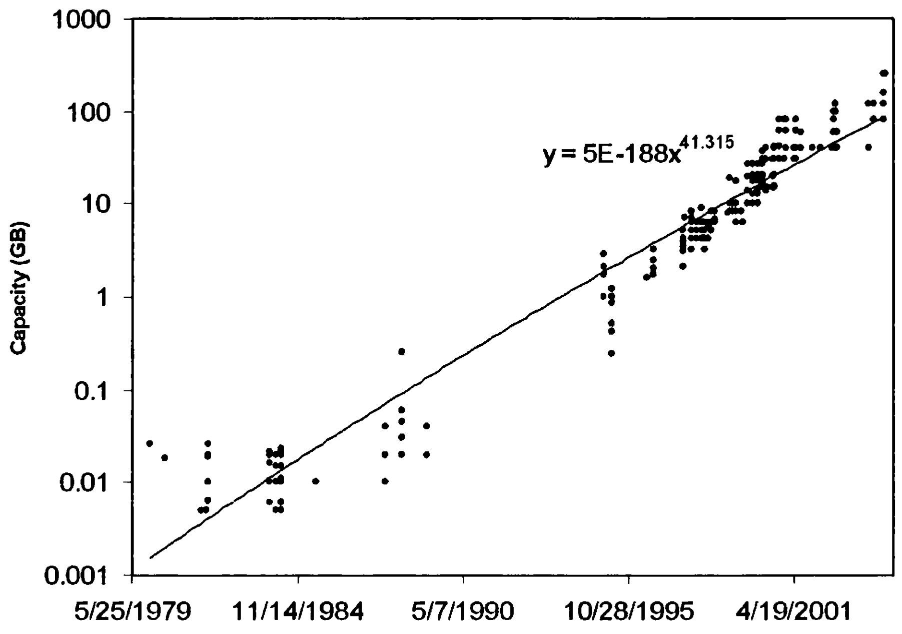


Figure 6.4: The size of disks over time on a logarithmic scale

of the data are considered to be far more crucial than the accuracy of each individual piece of data. Should a higher accuracy of data be required, several user studies and time-consuming statistical processes would have to be used over a cross section of computers. It was decided that for the purposes of the scalability test, accuracy of the contents of the knowledge base came second to the volume of knowledge and data itself.

6.2.4.7 Data Creation Theory

There have been many studies into the contents of file systems, however they were mostly concerned with Unix file systems and have not been conducted recently. Those conducted after 1994 stated several factors which influenced the data creation process.

A file usage study of software developers was carried out in 1994, of 7500 users connected by a LAN [113]. This study observed that:

- Regularly accessed files are larger
- Files are least mostly accessed and least often modified
- 50-60% of the files were shared between workgroups

- As sharing activity increases the modifications decrease
- The average file size for source code increases as sharing increases, and less-so for non-source code files

The first large-scale study into Windows file system usage was undertaken in 1999 by Doucer and Bolosky [87]. This study agreed with the findings of previous research, with the additional discoveries that:

- The mean file size ranged from 10KB to 40KB
- The number of files per user had increased by an order of magnitude
- Most files are small but most bytes are in large files
- Median file age is 48 days, but lifetimes vary widely
- File name extensions are strongly correlated with file size

Whilst both of these studies provide useful guidelines on file system data production, another study undertaken at approximately the same time [293] found that there was extreme variance in all of the traced usage characteristics. As all these studies were undertaken over 6 years ago it is likely that file sizes will have continued to increase, and user behaviour will have changed to reflect the increase in available data and media from the Internet and local networks. It is always difficult to accurately predict the development of computers and storage technology over a period of time. Therefore the findings of the previous research was kept in mind when implementing a data creation program, but were not followed religiously.

6.2.4.8 Data Creation Implementation

The file usage simulation program was written in C#.NET. It loaded in pre-set parameters from external files, such as 'words' (used for keyword and file name generation), 'authors' (a set of known authors) and 'extensions'. In addition to the list of file extensions, the upper and lower bounds for file sizes were loaded as a 'txt' file will most likely be smaller than an 'mpeg'. These values were used for calculating file size growth.

The number of files for which data was created was reliant on the time across which the data was to be produced. The total number of files produced was calculated by multiplying the number of years by 5000 (which was the current state/best case scenario for files created per year).

The program then filled in the data attributes for each file by using the pre-defined values that were loaded from the external files at initialisation. For one user, the majority of files available were authored by that user and a smaller amount would be available from other users. Authors were designated by weighted selection, with a 75% probablility of one specific author and 25% probability of an alternative author.

In order to assign the number of keywords for each file, between 0 and 10 were selected from those loaded from the external file, with duplicate keyword selections removed. The file's title was generated in the same manner with the weighted probabilistic addition of a keyword and the author's name. Once the file title had finished being built the length was between 1 and 13 words (with up to 10 of those words also being keywords, plus the author name).

The file extensions were allocated to each file from a set of 17 different file types. Given the extension, the file size was assigned by producing a value between the lower and upper bounds of file size associated with that extension.

The creation and modification times for the first version files were assigned by selecting a number between 0 and the time span entered by the user. The accessed time was created by adding an interval of up to 1 year to the modified time.

Following the above steps, the details for each file were filled in. At this stage the program will have only created the first generation of files (version 1). The program then iterates through the list of files, spawning a new file for each old one using a weighted ratio of 80:20 (e.g. out of 100 first generation files, 80 will have at least one other version, and 20 will not).

When a new version of a file was created, the filename, extension, creation time and keywords were duplicated directly from the older version. To reflect changes in the contents of the file, new keywords were added combined with the possibility of others being removed. The file size of a new version was manipulated accordingly to fit the trend of file sizes increasing over time, it was also assumed that the majority of files would increase in size after each version. The accessed time of the new version was created by generating an appropriate interval to the previous version's modified time, with the new modified time being set between the new access and old modified times.

Once the data for all second generation files had been produced, the above process for determining future versions of a file was repeated again for the newest files. As this is done in a probabilistic fashion, for each data set generated for identical time spans there is likely to be a natural variance in the number of files contained within.

The data creation program also had to produce data for users deleting files as well as creating them. Of course, as previously mentioned, in GIFS files are not removed, simply flagged as deleted so they can be retrieved as necessary. To reflect this operation data from the deletion of around 1500 files per year was created, plus a small percentage of files which would then be undeleted. This number was decided upon after reviewing a set of user's anecdotal data as well as the contents of their "Recycle Bins".

As the projected development of disk sizes is well over the projected data set size, disk space can be disregarded as a problem in future file systems. However, a large knowledge base will take a considerable amount of time to search and process. In order to combat this problem, GIFS uses a variety of different mechanisms and agents: archiving, caching, and combined archiving and caching.

6.2.5 Example Case

In order to show how the deployment of a Virtual Secretary would help users store and retrieve their files, the following problem is presented showing typical actions in an academic/administrative setting.

User P is working on the budget request documentation for the Computer Science department (a common annual task). He starts by reading some recommendations from User J contained in a document called "Budget2006.doc"(1). He makes his own notes in "draft_budget_2006.txt"(2) and generates some rough numbers in "budget05.xls"(3). Other members of staff read these notes and numbers and go on to create their own versions, each adding comments/changes etc. For 4 members of staff (typically the senior departmental staff), each one creates around 2 extra versions of each file (=3*4*2 = 24 files). Then the final drafts of the files are made to be taken to a faculty meeting, so approximately 3 more versions of each file are made (24+3 = 27). Someone then produces a presentation on the budget request over several days, creating several versions of this file. (27+6 = 33).

Each member of the Faculty of Science within the university creates their budget requests in a similar manner. The Faculty members are Computer Science, Chemistry, Mathematics, School of Biological Sciences, Physics and Psychology. If each department produces their budget request in a manner similar to that shown in the Computer Science department, this makes 33*6 = 198 files created for one budget meeting. The reports are made to the university for the faculty budget, to be presented at another meeting (so several more versions of each budget file following the meeting, plus a new set of documents created for this meeting, 250 files). The university administration then makes reports on the budget requests, including drafts, notes and reports from various members of the administration at different levels (500 files).

Consider that each file creates around 20 separate pieces of information. If there are 300 files available to one user over the course of a year on the topic of budget requests, this makes 6000 pieces of information to be searched. (Although the budget request above produced 500 files, it is unlikely that one user would have read access to all these files.) If the last 10 years worth of budget information is kept on the system, this makes over 60000 pieces of information relating to budget requests.

These numbers are simply for files that are related to department finance or the budget requests. Also considering that there will be other files on the system (on average, members of the Computer Science department were found to produce around 4000 files per year), this makes a total of files about the budget + (number of files per user * number of other user's data that gives us read/write access) = 300+(4000*20) = 80300 files with 1606000 pieces of data produced per year.

If disk and storage size for the knowledge bases is not considered to be a problem, then the time needed to search and process such large amount of data becomes critical for the usage of such a system. In order to show that GIFS is scalable, several tests were carried out over various sized datasets.

6.2.5.1 Different Approaches for Implementation

For the first implementation, the data for every file was all kept in one large knowledge base. However, this caused problems due to restrictions imposed by the implementation languages chosen. Sicstus Prolog version 3.12 can only use the first 256MB of RAM for stacks. The stack is a section of memory which Prolog uses to store user-defined predicates and also the recursion calculations. When processing large amounts of data, the stack can become full, causing the program to crash. When a knowledge base containing over 10000 files was queried there was not enough stack space to sort the results. When a knowledge base of over 250000 files was queried the stack ran out of space almost immediately the knowledge base was loaded.

In order to find a better solution to this software-imposed problem, several alternatives were investigated and implemented.

6.2.5.2 Approach 1

The first solution was to optimise the search algorithm (in Prolog) and reduce the stack size by working out the complete results for one file at a time (instead of on a per-attribute basis). Although this improved the running time for the program and allowed the smaller datasets to be sorted, it was still unstable when collating the results of the medium and large datasets due to the stack limitations.

6.2.5.3 Approach 2

The second approach involved changing the structure of the knowledge base slightly, by splitting up the one large knowledge base into several smaller knowledge bases, each containing the complete data on one set of attributes (such as author, keywords etc). This worked effectively for the mid-sized datasets, but still had little success with the largest ones. Further examination showed that after running each query the stack was not as efficiently freed as expected. The simple solution to this problem was to kill each prolog process programatically after it had computed the set of results, thus clearing the stack space and allowing for the results to be returned.

6.2.5.4 Approach 3

Approach 3 reduced the stack size by computing the results for each file one at a time, and then discarding it should the result fall below some pre-set threshold (for example, if the user requested the top 50 files if the result was lower than that of the file already in position 50). Although this approach seems an elegant solution, the recursive nature of Prolog meant that the knowledge base had to be read through twice to gather ranges (for those agents that required ranges of numbers to calculate upon), putting strain on the stack once more. Also, whilst returning the top N number of files is efficient for a single search, should the user next request the top N+1 files, the whole search would have to be performed again.

It was decided that approach 2 provided the most functional solution.

6.2.6 Test Conditions

The following tests were performed on a single machine with a Pentium 4 2.4GHz processor and 480Mb of RAM. It contained a 40 GB hard disk and was running Windows XP with Service Pack 2. The interface program ran under version 1.1.4322 of the Microsoft .NET framework. The agent programs ran under Sicstus Prolog 3.12. and incorporated the FuzzySearch library version 3.105. During the entirety of the process the computer was disconnected from the network to prevent any external or unintentional processes from running. The computer automatically rebooted itself after each set of results had been computed to ensure that each test was run under exactly the same conditions.

6.2.7 Direct Search

In order to compare the different mechanisms provided by GIFS to manage large amounts of data, a simulated search was performed. This particular search was borne from the example case described in §6.2.5, which requested the Secretary to find the first 50 files that were similar to a file written by "Jo", with the word "budget" in the name and the keywords of "budget", "finance" and "committee", with a "doc" extension.

For the control set of results, a brute force search is run over 5 different data sets for each time frame 5 times. These times were then averaged and the standard deviation

Dataset	1	2	3	4	5	avg	SD
1	23.000	21.625	21.172	21.953	21.453	21.841	0.632
2	21.813	21.484	21.734	21.594	21.063	21.538	0.262
3	21.844	21.703	21.516	21.719	21.500	21.656	0.130
4	21.359	21.188	21.469	21.281	21.078	21.275	0.134
5	21.734	20.938	21.609	21.172	21.375	21.366	0.288

Table 6.6: Brute force results for data over 1 month (seconds)

Dataset	Time
1 month	21.535
3 months	47.710
6 months	1:27.186
9 months	2:07.604
1 year	2:47.178
5 years	13:43.313
10 years	27:55.130
20 years	57:57.609
30 years	1:42:02.192
40 years	3:05:21.675
50 years	4:31:08.592

Table 6.7: Averaged brute force results (hh:mm:ss)

calculated as seen in Table 6.6. The full results for each dataset can be seen in the appendix in tables A.1- A.10, and the averaged results can be seen in table 6.7.

The results show that the search took approximately 20 seconds for a 1 month dataset (Table 6.6), 45 seconds for 3 months (Table A.1), 90 seconds for 6 months (Table A.2), 130 seconds 9 months (Table A.3) and 165 seconds for a year (Table A.4). The standard deviation across all these times is low and shows high consistency of all results. As the dataset size increased to between 5 and 20 years (Tables A.5- A.7) the standard deviation increased slightly. As the search times also increased this is unsurprising as the deviation still represents only a fraction of the overall search time. On the larger datasets the search took around 1 hour and 40 minutes for 30 year's worth of data (Table A.8), 3 hours for 40 years (Table A.9) and almost 5 hours for 50 years (Table A.10). The averaged results for the brute force searches can be seen in Figure 6.5.

The results from the brute force search show a linear (O)n increase in the length of search for the datasets up to 30 years where the standard deviation also increases dramatically. The datasets which are smaller than 30 years all produced consistent results of the search times with small standard deviations. However, the larger datasets show an unexpected exponential increase in search times with large stan-

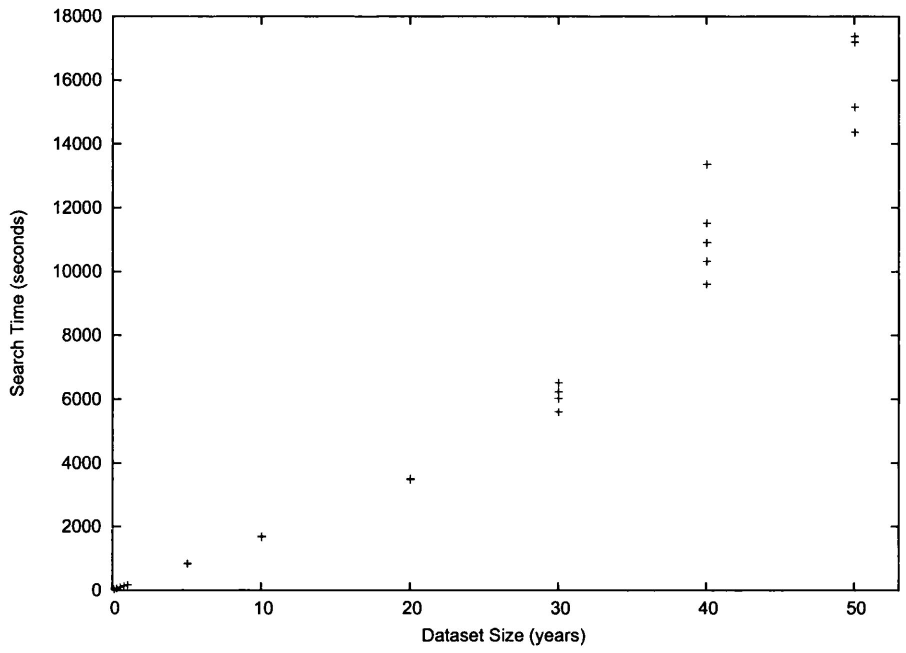


Figure 6.5: Average brute force search times

dard deviations. This is most likely due to the swapping/pagefile errors caused by the restrictions of the test machine hardware.

6.2.8 Three Approaches for Improving the Direct Search

With the results from the brute force search technique gathered and analysed we can now look at the techniques employed by GIFS in an attempt to reduce these times.

6.2.8.1 Caching

The first technique is based on the axiom that when a search is repeated, the only scores that need to be calculated are those for files which were added since the search was previously performed. These new scores are then combined in a result set with the old scores to give a complete list of results. Searches which have been identified as 'favourites' would be performed automatically without the need for user interaction at least once a day. To simulate the situation the appropriate amount of data (e.g one day's worth) was added to each dataset before running each search a second time.

6.2.8.2 Archiving

Even with the search only being undertaken over the last day's data, the times still proved too long to allow practical use. In the long-term datasets, the knowledge bases are bulky and cumbersome, taking hours to be searched through even for just a single day's worth of data. The archive agent can be used in this instance to remove old knowledge and data from the knowledge bases that may no longer be relevant. Note that this knowledge is not deleted, but moved to a different storage location so complete searches could still be performed if they were required. The archive agent will move any data which has a last modified and accessed timestamp of over a year. For files with multiple versions it will move all but the most recent 5. The knowledge base containing search results and records will have duplicate searches removed, with only the most recent remaining. The archive agent would run over night or on a daily basis when the computer is idle and so is not timed (as such timings would be meaningless). The resultant datasets after the archiving process had completed were used with the brute force search to show the improvement in search times. However, by combining this archiving method with that of result caching, the search times can be cut even further as seen below.

6.2.8.3 Combinatory

As we already have a list of the favourite searches (e.g. the most popular) it would be advisable to run each search in the background whenever new file data is added to the knowledge base. In this case, only one file has to be processed at a time, and as a background process (unless the user has immediately requested a search). The time taken for a background search over one file's data should remain almost constant for all dataset sizes, as the knowledge would be processed by the secretary at the same time it is added to the knowledge base. Using this method of updating the cache continuously gives the 4th projected improvement for search times.

6.2.9 Results

The brute force search mechanism showed an increase of O(n) as dataset size increased. The algorithm which used caching showed a similar complexity in time increase although to a lesser degree. Using archiving, the search time for each dataset leveled out at around 160 seconds, and combining this approach with caching further reduced the time to 15 seconds. The worst-case scenario data showed the same trends as the best-case scenario, with obviously larger search times.

Dataset	Brute Force	New Data	Archived	Combined
1	21.841	09.360	21.372	09.500
2	21.538	09.600	21.325	09.480
3	21.656	09.420	21.303	09.340
4	21.275	09.560	21.409	09.460
5	21.366	09.360	21.456	09.560

Table 6.8: Results for data over 1 month (seconds)

Dataset	Brute Force	New Data	Archived	Combined
1	47.569	10.120	47.347	10.400
2	47.475	10.120	47.406	10.460
3	47.647	10.100	47.175	10.220
4	47.772	10.100	47.390	10.280
5	48.088	10.120	47.187	10.220

Table 6.9: Results for data over 3 months (seconds)

6.2.10 Comparison of Best-Case Scenario

As seen before in §6.2.7, the search is run on data sets of 10 different sizes. Within each, there are 5 different data sets and each test is repeated 5 times to give an average. For brevity, only the average times are shown in Tables 6.8– 6.18, the full tables of results for each test can be found in Appendix A, Tables A.11– A.43.

Figure 6.6 shows each of the average times for a search using the caching method. It shows a quadratic increase in time over the different datasets. The deviation between these results is minimal as each test within a dataset produced similar results. Even over the largest datasets the deviation remains small, so the 'swapping' effect seen in the brute force tests has unlikely effected these results.

The next graph, Figure 6.7, plots the average search results for the archived datasets. This approach initially shows a very high increase in the rate of search times, but then once the datasets reach the size of 5 years, the times level off to a value between approximately 140 and 190 seconds. The spread in the individual results can be

Dataset	Brute Force	New Data	Archived	Combined
1	1:27.072	0:10.560	1:26.569	0:10.540
2	1:27.653	0:11.000	1:26.569	0:10.640
3	1:27.419	0:10.780	1:27.069	0:10.820
4	1:25.641	0:10.840	1:25.844	0:10.980
5	1:28.147	0:10.860	1:26.978	0:10.620

Table 6.10: Results for data over 6 months (minutes:seconds)

Dataset	Brute Force	Cached	Archived	Combined
1	2:07.962	0:12.300	2:06.231	0:12.560
2	2:07.506	0:12.220	2:03.503	0:12.240
3	2:08.131	0:12.120	2:03.612	0:12.460
4	2:08.038	0:12.520	2:02.750	0:12.120
5	2:06.385	0:16.780	2:02.428	0:15.920

Table 6.11: Results for data over 9 months (minutes:seconds)

Dataset	Brute Force	Cached	Archived	Combined
1	2:46.769	0:13.900	2:05.441	0:13.200
2	2:47.644	0:14.740	1:57.281	0:12.760
3	2:46.878	0:14.420	1:57.719	0:13.240
4	2:47.588	0:14.180	1:55.435	0:12.980
5	2:47.009	0:14.140	1:56.588	0:13.020

Table 6.12: Results for data over 1 year (minutes:seconds)

Dataset	Brute Force	Cached	Archived	Combined
1	13:41.422	0:53.240	3:05.825	0:14.960
2	13:43.006	0:53.400	3:06.316	0:14.600
3	13:38.919	0:54.520	2:46.622	0:15.040
4	13:42.278	0:54.900	2:48.216	0:15.300
5	13:50.941	0:54.260	2:26.760	0:14.320

Table 6.13: Results for data over 5 years (minutes:seconds)

Dataset	Brute Force	Cached	Archived	Combined
1	27:50.125	2:36.460	2:30.594	0:15.080
2	27:41.831	2:37.820	2:37.875	0:15.240
3	27:53.347	2:54.720	2:31.122	0:15.500
4	27:58.819	2:54.360	2:47.522	0:15.120
5	28:11.531	2:45.920	3:14.866	0:15.540

Table 6.14: Results for data over 10 years (minutes:seconds)

Dataset	Brute Force	Cached	Archived	Combined
1	58:17.941	9:10.260	2:46.781	0:15.480
2	57:49.353	8:46.460	3:17.353	0:16.720
3	57:39.406	9:05.240	3:01.010	0:15.400
4	57:45.663	9:08.820	2:55.294	0:15.540
5	58:15.681	8:46.960	2:37.141	0:15.180

Table 6.15: Results for data over 20 years (minutes:seconds)

Dataset	Brute Force	Cached	Archived	Combined
1	1:40:13.553	0:20:02.480	0:02:24.975	0:00:15.000
2	1:48:12.712	0:21:09.020	0:02:43.803	0:00:15.340
3	1:33:04.472	0:20:15.560	0:02:36.372	0:00:14.980
4	1:40:07.397	0:20:26.460	0:03:07.825	0:00:16.000
5	1:43:32.825	0:20:03.880	0:02:26.912	0:00:15.920

Table 6.16: Results for data over 30 years (hours:minutes:seconds)

Dataset	Brute Force	Cached	Archived	Combined
1	2:39:47.609	0:36:17.220	0:03:10.972	0:00:16.080
2	2:51:36.066	0:36:18.260	0:02:51.297	0:00:15.500
3	3:01:29.097	0:34:20.240	0:02:46.100	0:00:15.340
4	3:42:17.091	0:36:13.840	0:02:54.078	0:00:15.560
5	3:11:38.512	0:36:12.580	0:02:51.731	0:00:15.480

Table 6.17: Results for data over 40 years (hours:minutes:seconds)

Dataset	Brute Force	Cached	Archived	Combined
1	4:12:13.706	0:50:26.260	0:02:33.025	0:00:15.460
2	3:59:01.884	0:50:30.480	0:02:37.735	0:00:15.500
3	4:49:03.769	0:50:28.540	0:02:51.625	0:00:15.480
4	4:49:14.656	0:50:11.700	0:02:17.081	0:00:14.880
5	4:46:08.947	0:51:22.320	0:02:35.488	0:00:15.560

Table 6.18: Results for data over 50 years(hours:minutes:seconds)

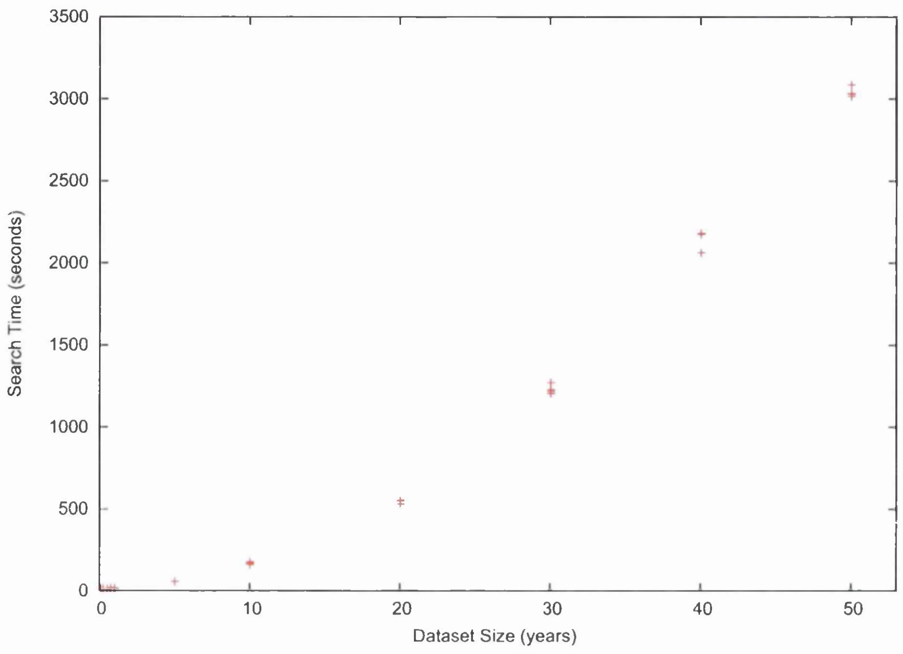


Figure 6.6: Search times using cached data

attributed to the way in which the archive agent functions. Each dataset will have obviously contained different numbers of versions for files leading to some datasets being reduced more than others by the archiving process.

Figure 6.8 shows the search times obtained when combining the archiving and caching techniques. Although the results in this graph follow the same trend as seen in Figure 6.7, the times have been greatly reduced.

Figures 6.9 and 6.10 show all four approaches plotted together. It is easy to see in Figure 6.9 the swapping problem encountered by the brute force search, but it is harder to see the archiving and cached results. Figure 6.10 plots the same data but on a logarithmic scale. From this graph we can see that brute force and caching both show quadratic increase. Caching gives better search times than archiving for datasets under 10 years in size, where the caching times continue to increase and the archive times remain reasonably constant. The most successful approach was the combination of archiving and caching, which resulted in a search time of around 15 seconds for any of the datasets regardless of size.

6.2.11 Comparison of Worst-Case Scenario

As mentioned in §6.2.4.1, whilst 5000 seems like a reasonable estimate for the number of files created per user per year, it is likely that the number of files will increase,

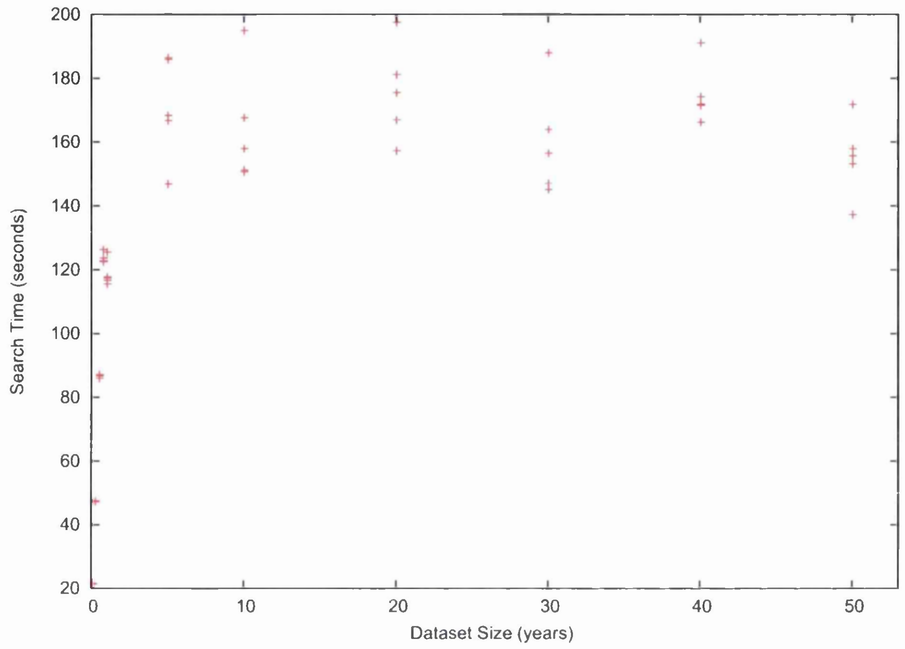


Figure 6.7: Search times using archived data

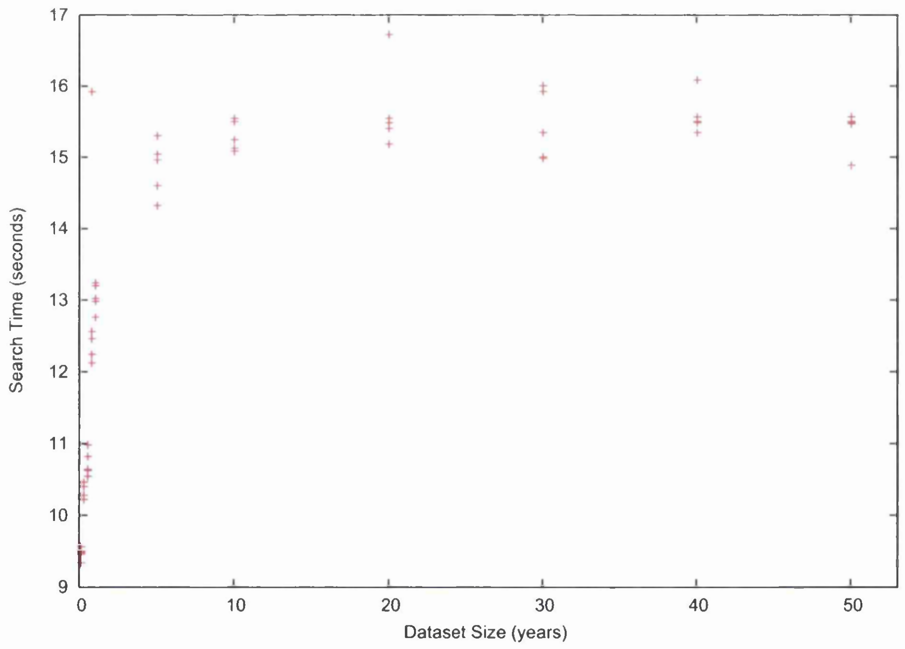


Figure 6.8: Search times using caching and archiving

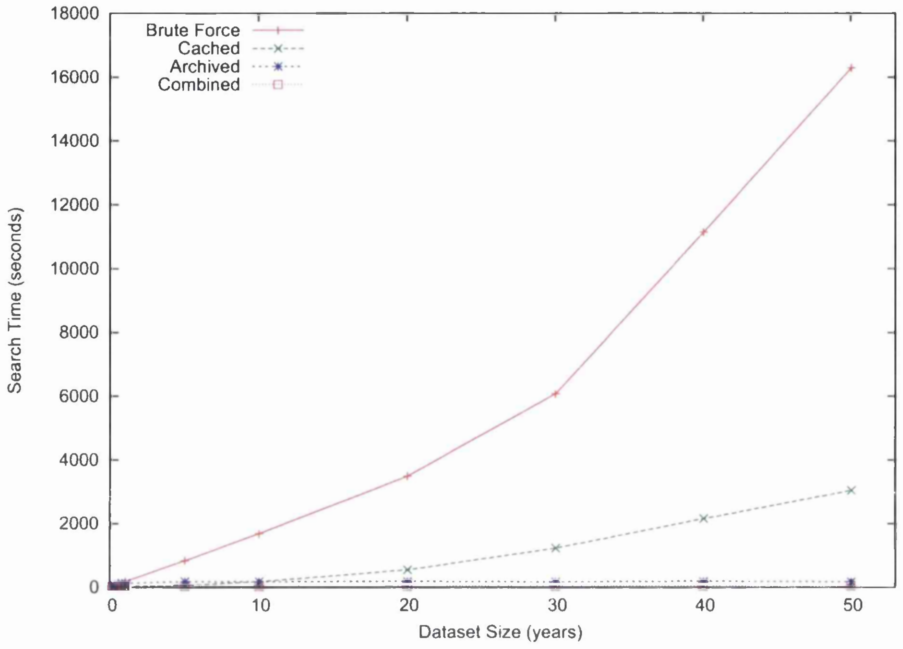


Figure 6.9: Comparison of search times

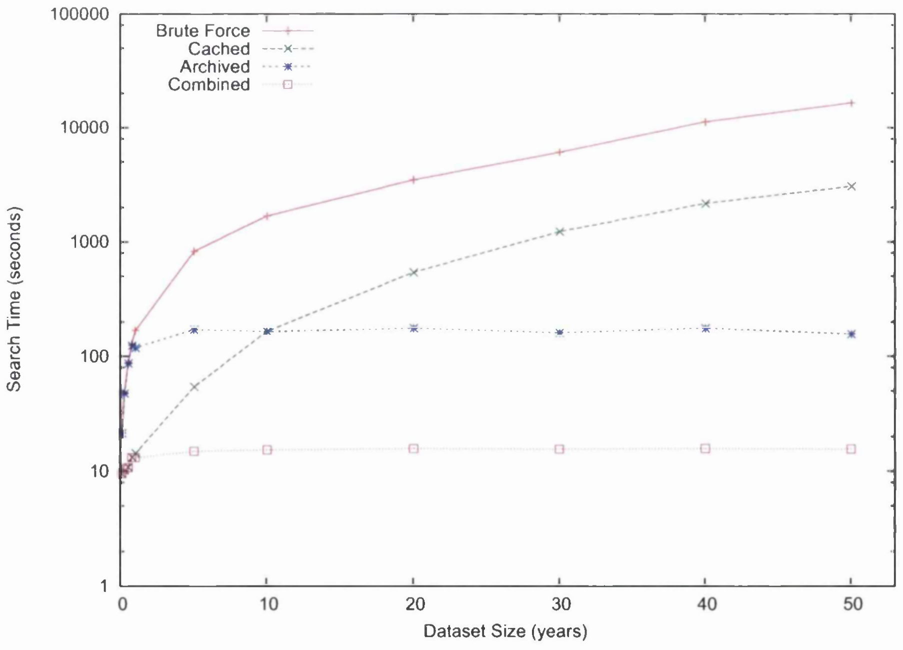


Figure 6.10: Comparison of search times with logarithmic scale

Dataset Size	Brute Force	Cached	Archived	Combined
1 year	15:06.392	01:14.432	04:51.240	0:20.550
5 years	32:52.972	03:51.692	05:40.556	0:22.998
10 years	1:29:45.220	18:49.412	05:11.596	0:22.371

Table 6.19: Average Search Times for Worst-case datasets(minutes:seconds)

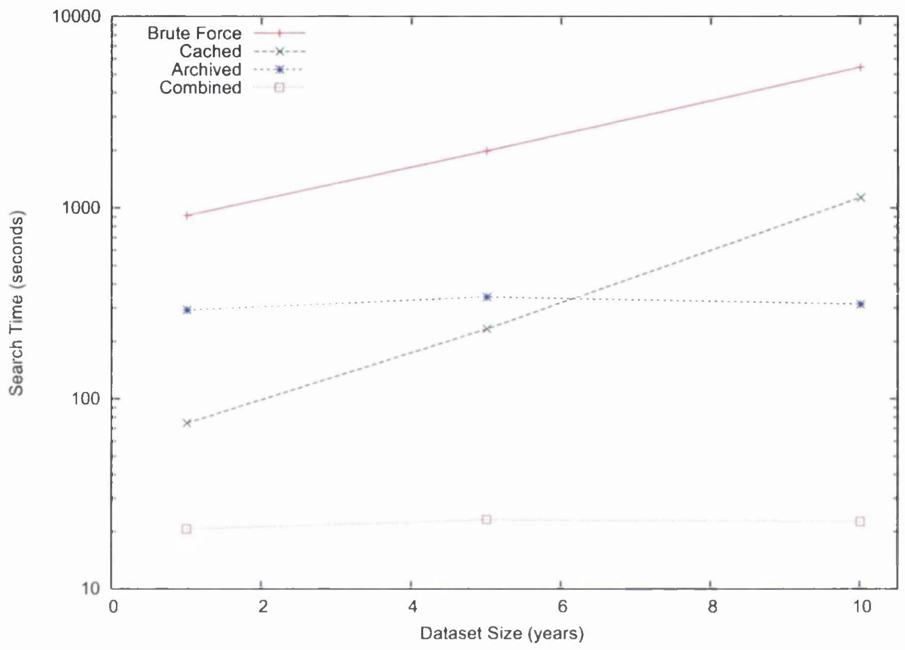


Figure 6.11: Comparison of worst-case scenario search times with logarithmic scale

hence the first set of tests are referred to as the best-case scenario. Using the projections created from the short study of cs-svr1, a second set of datasets was produced to reflect the likely increase and present what could be referred to as the worst-case scenario. The full results can be found in Appendix A (Tables A.44- A.55), whilst Table 6.19 shows the average search times for each worst-case dataset size.

The previous tests that had been run on the best-case datasets showed the increase in the time needed to process a dataset of a certain size at a rate higher than expected due to the limits in the capacity of the test machine memory. As this had been observed to happen for the smallest set of data (e.g. for the 10 different best case scenario datasets), it was decided that only the datasets in the range of 1 to 30 years would be used for the worst-case scenario test in order to reduce the effects of memory swapping. The worse case scenario tests were obviously much larger than those that had been previously used, and as a result the test computer was unable to process the datasets of 20 years and over. For this reason, only the 1, 5 and 10 year datasets were used for the worst-case scenario test.

The worst-case scenario datasets produced similar results to those of the best-case scenario, with understandably increased search times. Whilst it is not possible to extrapolate data so easily as only 3 datasets were used, these datasets displayed the same behaviour as their smaller counterparts. That is, the brute force search took the longest, the caching technique was faster than that of archiving until approximately 6 years. Archived data search times stayed level at about 5 minutes each, as did the combinatory approach at 22 seconds.

6.2.12 Conclusion

This results of this study have clearly proved the hypothesis. When using brute force and caching algorithms the search time was not scalable. However, when apply the techniques of archiving and combined archiving and caching the search times did not increase past a constant time. Thus, the average search time for a Virtual Secretary can be achieved in a scalable manner.

6.3 Case Study 2

The other important criteria when evaluating GIFS and the Virtual Secretary system is that of functionality and feasibility. There are many large scale network file systems already available, however there is not one that provides the automatic organisation, storage and retrieval of files as suggested in GIFS. In order to show that GIFS incorporating a Virtual Secretary could save the user time and effort, a second experimental case study was designed.

6.3.1 Hypothesis

The management of people-file relationships by a Virtual Secretary is more scalable than by human secretaries.

Previous literature [80, 43, 79, 20] has shown that humans have difficulty in remembering details of their computer related tasks. With respect to the access and distribution lists of a particular files, users would not be able to remember who the file should and should not be seen by when the circumstances change regularly and there is a high volume of items to be processed. This case study aims to show that by using a Virtual Secretary, the management of file distributions becomes scalable.

6.3.2 Assumptions

The following assumptions were made for this case study:

- The file access list of a single or dual user file would be simple for both humans and a Virtual Secretary to monitor. A file which was never shared with other users would not present any challenges whatsoever for the Virtual Secretary or its human counterpart. In order to fully test the scalability of this approach, a reasonably complex example was required to provide a substantial challenge.
- The case presented represents a highly collaborative, structured organisation with a typical but complex file-person scenario. File access and distribution patterns will vary widely between different users and different organisations. It would be impossible to create a file lifecycle that represented the access patterns of every different user. Through informal conversations with various members of the department, it was decided that the scenario presented was not completely unusual for the activities of a user involved in a task such as the one described.
- The Virtual Secretary for this study was new/inexperienced with the needs of the user, so the only data used for reasoning is that which is explicitly stated in the case study. In terms of consistent usage, it is highly likely that a Virtual Secretary would already have various recommendations for users based on previous actions. Whilst such recommendations would likely make the Secretary a more useful assistant, it would make the results of the case study harder to analyse, and give the Secretary an unfair advantage in the comparisons with other techniques.

6.3.3 Approach

Previous research [80, 43, 79, 20] has shown that people are prone to forgetting data related to their personal computing tasks. In order to further support the assumptions made, simulated scenario based on everyday user experience was proposed, and a data collection process was undertaken to gather data specifically on person-file relationships. In order to compensate for the lack of data in the knowledge base, three different Virtual Secretary personalities were used in the analysis process.

6.3.4 Managing File Access Lists

The main focus of this second case study was to show the steps normally required by a user when sharing files with other users or subsets of users. During the functional lifetime of a file, the people who have authorship and ownership rights are likely to change considerably as the file evolves from draft through to final copy to archive. Over a long period of time and in large organisations or institutions the membership of various committees and departments is likely to change through outside factors such as promotion, maternity leave or resignations. Also, if no distribution or membership lists are available for a user, they may make mistakes trying to remember the exact membership of a user group. Whilst it is not possible to prevent users from making mistakes or to predict when external events may affect the distribution of files, it is possible to minimise those effects. For example, by informing the user they had missed out a member from a committee document access list or automatically handling committee and group membership.

There are several different levels of effort required when changing the access list of a file. When a user creates a file and wishes to keep it private, no action is required on their part. If a file is accessible by several people and a user wishes to remove access for one or more of them, a small amount of effort is required to remove the specified user(s). However, if a user wishes to include a user or group of users in the access lists for a file, not only must they know who they intend to include, but also the electronic identification of that user. On traditional systems this would involve manually adding each user by some kind of network identifier. In the best case of traditional file systems this would mean using an email address or mailing list. Thus the levels of effort required to change access privileges for a file can be categorised as:

- No effort

    Making no changes to the access list

- Small effort

    Removing users from an access list

    Removing a group from an access list

- Large effort

    Adding a user to an access list

    Adding a group of users to an access list

As more complex actions are undertaken with access lists, the cumulative effort required for each whole action can be calculated by summing the effort required for each of its constituent parts. For example, the action of removing all users bar one from an access list would take small effort (removing the group) plus large effort (re-adding the single user).

The effort required to perform these same actions using GIFS can be classified into similar groups as those without. Confirming a set of choices or not changing the access control list can be categorised as negligible or no effort. Removing users from an access list would require small effort, and adding users or groups would take slightly more effort. However, using the Virtual Secretary to add a user would take considerably less effort than performing the same action without as the Secretary maintains the list of identities of users, so the addition process is simplified. Also, at other times the Secretary may make suggestions to the user (for example, if the user has unintentionally missed someone off the access control list of a committee distributed document) which would make a potentially large effort action (adding a user) into a negligible effort action (confirming or disagreeing with the Secretary's suggestion).

To demonstrate how the Virtual Secretary reduces the effort needed to maintain user access lists, we shall now examine an example scenario in more detail.

6.3.5 Example Case

User C creates a new file (event 1), some notes on the upcoming budget report to be submitted by his department to the university. He spends a couple of days reworking this file, adding and changing its contents (events 2, 3 and 4) before sending it to two other members of staff (User P and User F) who are also responsible for producing the budget report (event 5).

User F informs user C that he is not interested in the budget report this year, as this responsibility has been handed to User H. User C continues to edit the file and sends the next version to User P and User H (event 6). After reading the document, User H makes some changes and sends it back to Users C and P (event 7). User C thinks some of the changes are useful, but others are incorrect so he makes his own changes to the most recent file and sends it back to Users H and P (event 8). User P makes additional changes to the contents of the file and sends it back to Users C and H (event 9). With the additional input from his coworkers, User C continues to work on the file over the course of a weekend, creating 3 more versions and making each available to Users H and P (events 10, 11 and 12).

Users H and P are both happy with the contents of the file, so it is then sent by User C to the whole department (Group D) for read-only access (event 13) via email, although only a fraction of staff members will actually read the file. Only User C now has write access, and following some suggestions, creates a further two versions of the file (event 14 and 15), with the latter being sent only to the head of the budget committee (User HBC). User HBC is satisfied that the file reaches the minimum requirements for consideration by his committee, and so asks User C to send the file on to the rest of the budget committee (event 16).

User C sends the file to the budget committee (Group BC), but unbeknown to him, leaves off a member of the committee (User X) in error. This causes some confusion within the committee as not all members have been able to see the file's contents. When the error is later discovered, User C also sends it to the previously forgotten member (event 17). The next time User C submits a new version of this report to the budget committee (event 18), someone has left their job (User Y) and so is removed from the readership list.

Finally, the budget committee is happy with the report, and so User C can send it to the executive committee (Group EC) for review (event 19). The executive committee have a few more suggestions for User C, so he creates two more versions of the file (event 20 and 21), allowing the executive committee to see both. Once accepted by the executive committee, User C can send the finished report to the heads of department within his faculty (Group FDH) for informational purposes only(event 22). Having completed the budget report for one year, User C makes additional notes surrounding the document and sends it to Users H and P (event 23) in order to help with next year's report.

Table 6.20 shows how the access privileges for the file described in the above scenario change over a period of time, along with the manual actions required to be taken by User C in order to ensure these privileges are correct. Whilst it may be argued that for the same file there would be residual access privileges, this would not be the case when using a traditional file system with no version control, hence the files would be treated as individual.

6.3.6 File Access Data Collection

Over a period of 5 days, 5 subjects were asked to reason on and remember the file access lists for the example scenario. After being informed of each event, they were asked to consider who the file should next be available to for both read and write access. The amount of time each person took to decide upon the access list was recorded along with their answer.

The user study was conducted in the following way:

- Five unpaid subjects volunteered to participate in this study.
- The study was conducted over a period of five days.
- Each day, every subject was interviewed once in the morning and once in the afternoon.
- Each interview comprised of between two and three questions, and lasted approximately two minutes.
- During the interview, the subject was told of an event affecting the file access list for a document, which corresponded to an event seen in Table 6.20.
- Using both the information they had just heard and by recalling the previous access list from memory, the subject was then asked to decide what the new access list for the file should be.

Event	Desired Access Control List		Action Required
	Write	Read	
1	C	C	-
2	C	C	-
3	C	C	-
4	C	C	-
5	C, F, P	C, F, P	Add F, P(Write, Read)
6	C, P, H	C, P, H	Add H, P(Write, Read)
7	C, P, H	C, P, H	Add F, P(Write, Read)
8	C, P, H	C, P, H	Add F, P(Write, Read)
9	C, P, H	C, P, H	Add F, P(Write, Read)
10	C, P, H	C, P, H	Add F, P(Write, Read)
11	C, P, H	C, P, H	Add F, P(Write, Read)
12	C, P, H	C, P, H	Add F, P(Write, Read)
13	C	C, P, H, Group D	Add F, P, Group D(Read)
14	C	C, P, H, Group D	Add F, P, Group D(Read)
15	C	C, HBC	Add HBC(Read)
16	C	C, HBC, Group BC-X	Add HBC, Group BC-X (Read)
17	C	C, HBC, Group BC	Add HBC, Group BC(Read)
18	C	C, HBC, Group BC-Y	Add HBC, Group BC(Read), Remove Y
19	C	C, Group EC	Add Group EC(Read)
20	C	C, Group EC	Add Group EC(Read)
21	C	C, Group EC	Add Group EC(Read)
22	C	C, Group FDH	Add Group FDH(Read)
23	C	C, H, P	Add H, P(Read)

Table 6.20: Manual changes required for the access control list of the file

- The answer each subject gave was recorded, along with the time taken to produce the answer.

The detailed sets of results per subject can be found in the Appendix, Tables A.56–A.60. The average timings can be seen in Table 6.21, along with the number of errors that the user made in each assertion. If one of the subjects were to give read access to a user who should not have been permitted, this would count as one error. If the subject were to omit access for a required user, this would also count as one error.

Event	Average Time (Seconds)	Average Errors
3	3.6	0
4	7.3	0
5	7.02	0.8
6	6.77	0.6
7	26.77	1
8	9.76	1.2
9	8.64	0.8
10	27.52	0.8
11	13.72	0.8
12	6.1	0.8
13	33.36	3.4
14	19.28	3.4
15	26.99	3.6
16	19.72	3.8
17	36.51	4.6
18	31.43	4.6
19	41.18	4.4
20	39.17	5
21	47.44	4.8
22	49.02	5.2
23	26.86	3.4

Table 6.21: Averaged results of users predicting file access lists

The results from this experimental data show that users were prone to forgetting who they were meant to allow file access to, openly admitted to guessing and took longer to provide an answer (incorrect or not) as time and complexity of the problem increased. The subjects were mostly good at remembering who the file should be distributed to when the circumstances did not change greatly (e.g. events 3–12), but were less consistent at remembering and reasoning when several users were involved. Some subjects even produced names for the access list that had not been mentioned as part of the study, whilst another forgot that the "boss" should have

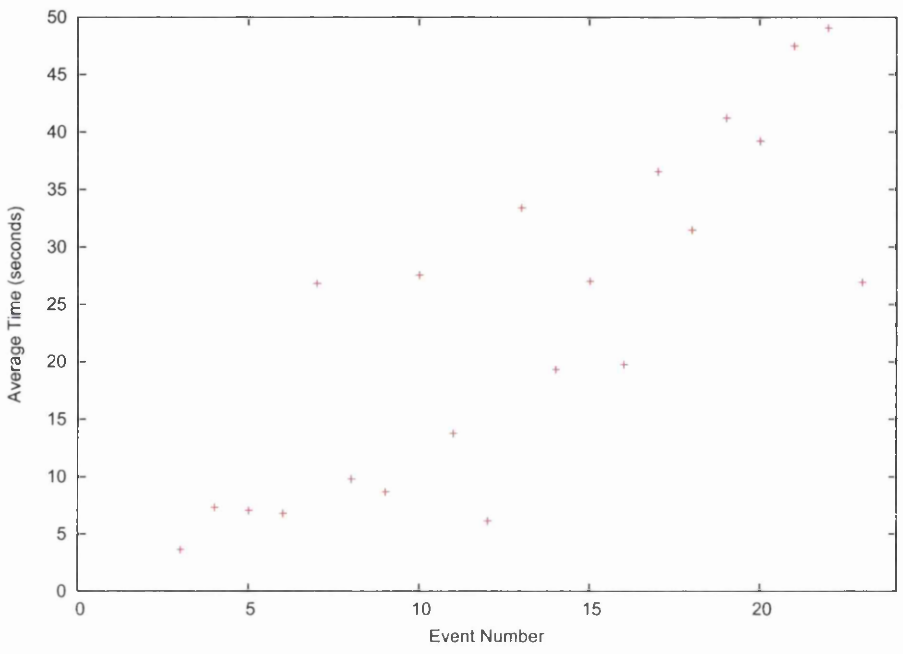


Figure 6.12: Average time taken by a user to calculate access user list for case study 2

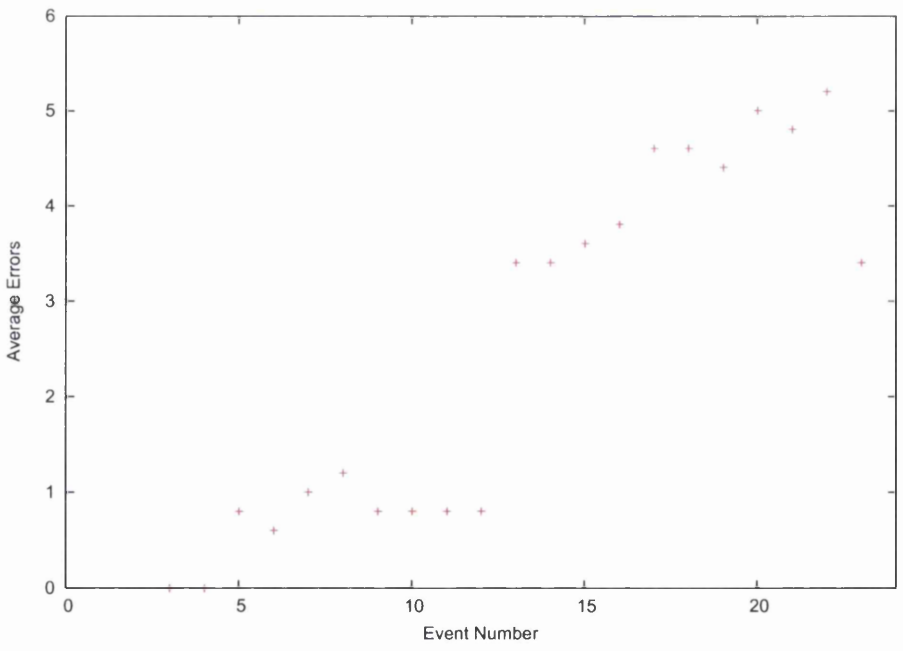


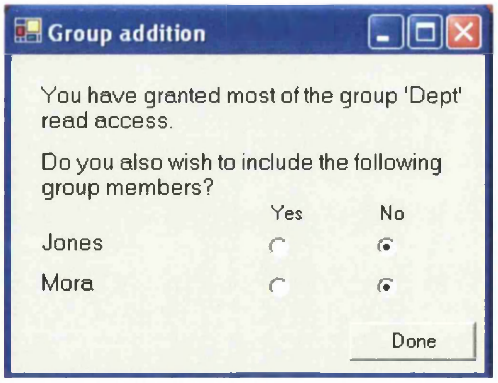
Figure 6.13: Average number of errors by a user to calculate access user list for case study 2

access to their own files.

6.3.7 Virtual Secretary Created Access Lists

In order to evaluate the assistance given to the user via a Virtual Secretary to save time and effort, the scenario in §6.3.5 was used. Three different Secretary personality settings were deployed; default, strict and lenient. When each new version of the file was added (as seen in events 1–23) the suggestions made by the Secretary were noted, along with the actions required by the user to produce the desired access control list.

There are several ways in which the Virtual Secretary can help the user avoid making mistakes as seen in the example case. As well as holding membership lists of defined groups centrally, the Secretary can create groups dynamically if it identifies a set of users who are frequently added to access lists together. When adding a file if the access list intersects with $\geq$75% of the membership of a previously defined group, the Secretary will bring this to the user's attention (seen in Figure 6.14) and suggest that the remaining users are also added. The details of users who can access each file are displayed in a simple way through the optional interface seen in a previous chapter. Should the predictions differ from the desired output, the interface shown in Figure 6.15 is used.



You have granted most of the group 'Dept' read access.

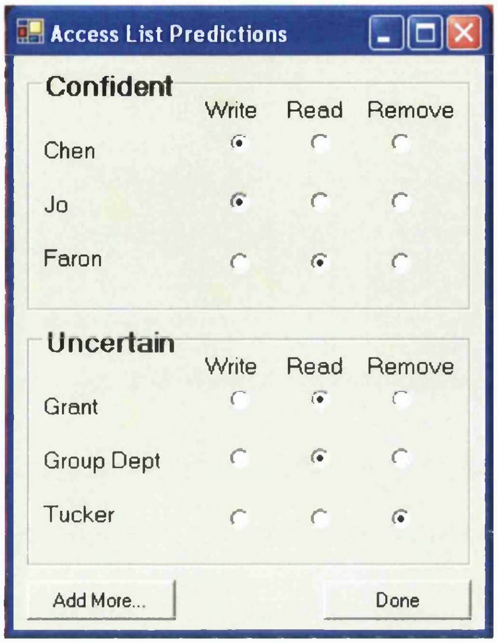
Do you also wish to include the following group members?

	Yes	No
Jones	○	●
Mora	○	●

Figure 6.14: A group membership alert

6.3.7.1 Calculations

In order to produce the access control lists and suggestions for a user, the agents supporting the Virtual Secretary must perform several calculations across the data and knowledge in the knowledge bases. The numerical value of the output denotes a probability that a user should be included or excluded from an access list. The



Confident

	Write	Read	Remove
Chen	●	○	○
Jo	●	○	○
Faron	○	●	○

Uncertain

	Write	Read	Remove
Grant	○	●	○
Group Dept	○	●	○
Tucker	○	○	●

Figure 6.15: The Virtual Secretary prediction interface

overall calculation is made up of several smaller equations, which are described below.

The Virtual Secretary predicts the next access list for a file by using the following equation.

$$\begin{aligned} \text{Prediction}_{\text{final}} &=_{\text{def}} (1-k) * \text{Prediction}_{\text{old}} + k * \text{Prediction}_{\text{new}} \\ & \quad 0 \leq k \leq 1 \end{aligned}$$

k is a constant defined within the personality settings of the Virtual Secretary and denotes how much importance should be attached to the previous results. The smaller *k* is, the more importance is given to the previous results of the calculation, whilst a *k* of value 1 would ignore the results of the previous calculation completely.

The score of Prediction$_{old}$ is located in the knowledge base, whilst the value of Prediction$_{new}$ is calculated by using the following equation:

$$\text{Prediction}_{\text{new}} =_{\text{def}} \frac{\sum w_i c_i}{\sum w_i}$$

w is a set of weightings which vary across a corresponding set of criteria, *c*. For each user two prediction scores are calculated, one for predicting write access, and the other for predicting read access. They are defined slightly differently when studied in detail below, but for the purpose of the main equation above can be treated as the same. The weightings { $w_{1..7}$ } for each score are determined by the personality settings of the Secretary. The calculations { $c_{1..7}$} for each stage of the prediction are defined as follows, where *f* and *f'* denote files, *u* denotes a user and *n* denotes the current time in Unix format.

$$\begin{aligned}
F &=_{\text{def}} \text{the set of all files}\\
U &=_{\text{def}} \text{the set of all users}\\
F_{pv}(f) &=_{\text{def}} \{f' \,:\, f' \text{ is a previous version for } f\}\\
U_{wp}(f) &=_{\text{def}} \{u \,:\, u \text{ has write permission for } f\}\\
U_{rp}(f) &=_{\text{def}} \{u \,:\, u \text{ has read permission for } f\}\\
x \in S &=_{\text{def}} \{x \text{ is a member of set } S\}
\end{aligned}$$

$$\text{cs}(u,f) =_{\text{def}} \frac{|\{f' \in F_{pv}(f) \,:\, u \in U_{wp}(f')\}|}{|F_{pv}(f)|} \tag{6.1}$$

The co-author score, cs(*u, f*) defined in equation 6.1, is calculated by the sum of number of write access permissions a user has had over the previous versions of the file, divided by the total number of versions of the file to give a final score between 0 and 1. Users that have co-authored every previous version of a file will score 1, whilst a user who has never co-authored a previous version will score 0.

$$\text{pa}(u,f) =_{\text{def}} \frac{|\{f' \in F_{pv}(f) \,:\, u \in U_{rp}(f')\}|}{|F_{pv}(f)|} \tag{6.2}$$

The previous access score, pa(*u,f*) defined in equation 6.2, is calculated by the sum of the number of versions of a file that a user *u* had read access for, divided by the total number of versions. This results in a score of between 0 and 1, with a user who has had read access to every previous version scoring the maximum.

$$\begin{aligned}
U_{rcp}(f) &=_{\text{def}} \{u \,:\, u \text{ was correctly predicted to be given read access for } f\}\\
U_{wcp}(f) &=_{\text{def}} \{u \,:\, u \text{ was correctly predicted to be given write access for } f\}\\
U_{ricpn}(f) &=_{\text{def}} \{u \,:\, u \text{ was incorrectly predicted to not be given read access}\\
&\qquad \text{for } f\}\\
U_{ricpg}(f) &=_{\text{def}} \{u \,:\, u \text{ was incorrectly predicted to be given read access for } f\}
\end{aligned}$$

$$\text{crrs}(u,f) =_{\text{def}} \frac{|\{f' \in F_{pv}(f) \,:\, u \in U_{rcp}(f')\}|}{|F_{pv}(f)|} \tag{6.3}$$

$$\text{crws}(u, f) =_{\text{def}} \frac{|\{f' \in F_{pv}(f) \ : \ u \in U_{wcp}(f')\}|}{|F_{pv}(f)|} \tag{6.4}$$

The correct read access score, (crrs(u, f)) and the correct write access score, (crws(u, f)) defined in equations 6.3 and 6.4 respectively, are both calculated by taking the number of correctly made predictions from all previous versions of this file for a given user and dividing it by the number of versions of a file. Users who have been correctly predicted to have read/write access to all previous versions of a file will have a score of 1, whilst users who have had no correct read/write predictions made for this file will score 0.

$$\begin{aligned}\text{wrrs}(u, f) =_{\text{def}} & \frac{|\{f' \in F_{pv}(f) \ : \ u \in U_{ricpg}(f')\}|}{|F_{pv}(f)|} \\ & + \frac{|\{f' \in F_{pv}(f) \ : \ u \in U_{ricpn}(f')\}|}{|F_{pv}(f)|}\end{aligned} \tag{6.5}$$

$$\begin{aligned}U_{wicpn}(f) &=_{\text{def}} \{u \ : \ u \text{ was incorrectly predicted to be given write access for} f\} \\ U_{wicpg}(f) &=_{\text{def}} \{u \ : \ u \text{ was incorrectly predicted to not be given write access for } f\}\end{aligned}$$

$$\begin{aligned}\text{wrws}(u, f) =_{\text{def}} & \frac{|\{f' \in F_{pv}(f) \ : \ u \in U_{wicpn}(f')\}|}{|F_{pv}(f)|} \\ & + \frac{|\{f' \in F_{pv}(f) \ : \ u \in U_{wicpg}(f')\}|}{|F_{pv}(f)|}\end{aligned} \tag{6.6}$$

The definitions for the incorrect read and incorrect write access equations, seen in equations 6.5 and 6.6 are slightly more complex. For each, the score is calculated by the number of times a user was predicted to have access incorrectly, added to the number of times a user was *not* predicted to be given access incorrectly, divided by the total number of file versions. The resulting score falls between the values of 0 and 1, as a user could not logically have a total greater than the number of versions contained in the sets defining both predictions. A user who has always been included in the incorrect predictions will score a 1, and a user who has never had any incorrect predictions will score a 0.

$$\begin{aligned}
\mathrm{t}(f) &=_{\text{def}} \text{time that } f \text{ was accessible} \in \mathbb{N} \\
\mathrm{T}(f,u) &=_{\text{def}} \max\{\mathrm{t}(f') : f' \in F_{pv}(f) \text{ and } u \in U_{wp}(f') \text{ or } u \in U_{rp}(f')\} \\
&\qquad \text{by convention, } \max \emptyset = 0 \\
\tau(F) &=_{\text{def}} \max\{\mathrm{t}(f) \,:\, f \in F\} \\
\mathrm{lf}(F) &=_{\text{def}} \{f \,:\, f \in F \text{ and } \mathrm{t}(f) = \tau(F)\}
\end{aligned}$$

$$\mathrm{las}(u) =_{\text{def}} \begin{cases} 1 & \text{if } u \in U(\mathrm{lf}(F)) \text{ or } u \in V(\mathrm{lf}(F)) \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

$$\mathrm{ols}(u,f,n) =_{\text{def}} \begin{cases} 1 & \text{if } |\mathrm{T}(f,u) - n| > 15778463 \\ 0 & \text{otherwise} \end{cases} \tag{6.8}$$

The last action score, las(u) defined in equation 6.7 has a value of 1 if the user u was included in the last action performed (not necessarily for a version of this file), and 0 if they were not. Similarly, the old score (ols(u,f,n)) defined in equation 6.8 will return a value of 1 if the user u last had access to a version of f over six months (15778463 seconds) ago, or a 0 of they were assigned access to the file more recently.

$$\begin{aligned}
\mathrm{fs}(f) &=_{\text{def}} \text{the set of file strings of} f \\
\mathrm{wws}(u) &=_{\text{def}} \text{the set of strings linked to user } u \text{ for write access} \\
\mathrm{rws}(u) &=_{\text{def}} \text{the set of strings linked to user } u \text{ for read access}
\end{aligned}$$

$$\mathrm{wlw}(f,u) =_{\text{def}} \mathit{fuzzy}(\mathrm{fs}(f), \mathrm{wws}(u)) \sim \frac{|\mathrm{fs}(f) \cap \mathrm{wws}(u)|}{|\mathrm{fs}(f)|} \tag{6.9}$$

$$\mathrm{rlw}(f,u) =_{\text{def}} \mathit{fuzzy}(\mathrm{fs}(f), \mathrm{rws}(u)) \sim \frac{|\mathrm{fs}(f) \cap \mathrm{rws}(u)|}{|\mathrm{fs}(f)|} \tag{6.10}$$

The word link scores for read and write (rlw(f, u) and wlw(f, u), defined in equations 6.9 and 6.10 respectively) utilise the 3rd party fuzzy string matching library. As such, the exact score is difficult to define, although an approximation gives a close enough definition for the purposes of understanding the prediction calculations. The score for each can be approximated to the similarity between the set of words defined in a `wordlink/2` predicate in the knowledge base for a user u, that are also included in the word list for a file f, divided by the number of key words linked to f. Note that in the definition, the term "key words" is different to "keywords" as it also includes strings found in the file name. The score returned is between 1 for a perfect and complete match, to 0 for no match.

6.3.7.2 Defining Personalities and Output

The values required to define the default Secretary personality for use in the above calculations can be seen in Listing 6.1.

Listing 6.1: The personality settings for the default Virtual Secretary

```
weighting(a1, 0, coauthor, 0.35).
weighting(a2, 0, previousaccess, 0.25).
weighting(a3, 0, wordlink, 0.2).
weighting(a5, 0, correct, 0.05).
weighting(a6, 0, mistake, 0.05).
weighting(a7, 0, lastaction, 0.05).
weighting(a8, 0, old, 0.05).
threshold(a9, 0, certain, 0.4).
threshold(a10, 0, perhaps, 0.2).
k(0.98).
```

When the calculation agent runs across the knowledge base the scores are output in the predicate `predict/6` which takes the form of:

```
predict(FID, Timestamp, UID, AccessType, PredictType, Score)
```

This predicate contains the scores of each individual user which is then used by the permission agent in order to propose the access list membership to the user via the Virtual Secretary interface. An example set of results created by the calculation agent can be seen in Listing 6.2, with the `FID` shown in condensed format to save space.

Listing 6.2: Example output from the calculation agent

```
predict(11...4854, 1146493474, "C", read, certain, 0.751).
predict(11...4854, 1146493474, "P", read, perhaps, 0.314).
predict(11...4854, 1146493474, "H", read, perhaps, 0.223).
predict(11...4854, 1146493474, "C", write, certain, 0.751).
predict(11...4854, 1146493474, "P", write, perhaps, 0.314).
predict(11...4854, 1146493474, "H", write, perhaps, 0.223).
```

6.3.8 Virtual Secretary Suggestions

6.3.8.1 Default Personality

The first test was run using the default Virtual Secretary personality, and the results displayed in Table 6.22. To begin with (events 1–5) the Secretary correctly predicts that the new document is to be kept private. Once users P and H have each been added twice the Secretary makes an uncertain prediction that they should have read and write access. At event 11, user P scores high enough to be moved into the certain prediction category. When user H also reaches this level (event 13) the access control list is changing dramatically from the previous patterns seen. Users H and P now have to be removed from the list and their scores begin to drop accordingly. When group D is added to the access list its score remains too low for it to become even an uncertain prediction as so many versions previously have been sent to users H and P. For the remainder of the actions (events 15–22) the Secretary does not make any correct predictions until users P and H are added in event 23. The Virtual Secretary helps the user by avoiding the exclusion of user X in event 17 and automatically removing user Y in event 18 and has an overall effort level less than that of manual addition. However, with the default personality settings, the prediction scores do not decrease sufficiently when users are removed from the access lists, or increase sufficiently when they are continually added (after the initial creation).

6.3.8.2 Strict Personality

The second test used a different set of Secretary personality settings. The so-called "strict" personality has higher score thresholds for predictions to be made and places a higher weighting on previous actions. The results of using this Secretary to predict the access control list can be seen in Table 6.23. Similar to the default personality, the strict personality also included users P and H incorrectly in event 13, although by event 19 user P only has an uncertain prediction for read access and user H has been removed from the predictions entirely.

Whilst the incorrect predictions of users P and H in events 13–18 reduce their scores sufficiently to be excluded from further predictions, the scores of other users such as HBC and group BC are not high enough to be included.

6.3.8.3 Lenient Personality

The lenient Virtual Secretary personality had a lower threshold for uncertain predictions but also placed less importance on long-term user scores and more on the previous actions and errors. The aim was to make this personality identify rapidly-changing access patterns promptly and also stop making incorrect predictions faster.

As seen in Table 6.24 this personality had some success at reducing the amount of effort required by the user. Even though users P and H have been included in over half the access lists before event 13, once an erroneous prediction is flagged their scores begin to drop sufficiently to be excluded from addition in three events. Once a user has been added to the access list their score will be high enough for the Secretary to make an uncertain prediction to include them in the next version of the file. The high adaptability of this personality means that is is particularly suited to

Event	Confident		Uncertain		Action Required
	Write	Read	Write	Read	
1	C	C	-	-	-
2	C	C	-	-	-
3	C	C	-	-	-
4	C	C	-	-	-
5	C	C	-	-	Add P, F(Write, Read)
6	C	C	-	-	Add P, H(Write, Read)
7	C	C	P	P	Add H(Write)
8	C	C	P, H	P, H	-
9	C	C	P, H	P, H	-
10	C	C	P, H	P, H	-
11	C, P	C, P	H	H	-
12	C, P	C, P	H	H	-
13	C, P, H	C, P, H	-	-	Remove P, H(Write); Add Group D(Read)
14	C	C	P, H	P, H	Remove P, H(Write); Add Group D(Read)
15	C	C	P, H	P, H	Remove P, H(Write, Read); Add HBC(Read)
16	C	C	P, H	P, H	Remove P, H(Write, Read); Add HBC, Group BC(Read)
17	C	C	P, H	P, H	Remove P, H(Write, Read); Add HBC, Group BC(Read)
18	C	C	P, H	P, H	Remove P, H(Write, Read); Add HBC, Group BC(Read)
19	C	C	P, H	P, H	Remove P, H(Write, Read); Add Group EC(Read)
20	C	C	P, H	P, H	Remove P, H(Write, Read); Add Group EC(Read)
21	C	C	P, H	P, H	Remove P, H(Write, Read); Add Group EC(Read)
22	C	C	P, H	P, H	Remove P, H(Write, Read); Add Group FDH(Read)
23	C	C	P, H	P, H	-

Table 6.22: Changes required for the access control list of the file using the default Virtual Secretary personality

Event	Confident		Uncertain		Action Required
	Write	Read	Write	Read	
1	C	C	-	-	-
2	C	C	-	-	-
3	C	C	-	-	-
4	C	C	-	-	-
5	C	C	-	-	Add P, F(Write, Read)
6	C	C	P, F	P, F	Remove F(RW); Add H(Write, Read)
7	C	C	P, H	P, H	-
8	C	C	P, H	P, H	-
9	C	C	P, H	P, H	-
10	C	C	P, H	P, H	-
11	C	C	P, H	P, H	-
12	C	C	P, H	P, H	-
13	C, P	C, P	H	H	Remove P, H(Write, Read); Add Group D(Read)
14	C	C	P, H	P, H	Remove P, H(Write, Read); Add Group D(Read)
15	C	C	P, H	P, H	Remove P, H(Write, Read); Add HBC(Read)
16	C	C	P, H	P, H	Add HBC, GroupBC(Read); Remove P, H(Write, Read)
17	C	C	P, H	P, H	Add HBC, GroupBC(Read); Remove P, H(Write, Read)
18	C	C	P, H	P, H	Add HBC, GroupBC(Read); Remove P, H(Write, Read)
19	C	C	-	P	Add EC(Read); Remove P(Read)
20	C	C	-	P	Add EC(Read)
21	C	C	-	-	Add EC(Read)
22	C	C	-	-	Add FDH(Read)
23	C	C	-	-	Add P, H(Read)

Table 6.23: Changes required for the access control list of the file using a strict Virtual Secretary personality

situations where the access list changes frequently and many short-term changes are made. It may not suit more long-term, complex access list patterns.

6.3.9 Comparison of Effort

The effort levels required to manipulate the access control lists using the manual method and Virtual Secretary personalities can be seen in Table 6.25. Manually setting the file access privileges would take 35 large effort and 1 small effort actions in total. In addition to these actions, an error was made on the part of the user in event 16 meaning that a user was accidentally excluded. When using any of the Virtual Secretary personalities this exclusion was brought to the user's attention and so the error did not occur.

The default Virtual Secretary personality shows a reduction in the levels of effort required of the user to produce the same access control list. However, in the latter events (13 onwards), more effort is required to maintain the access list than when using the manual technique as this personality places more importance on long-term trends. When the access permissions change in quick succession the errors made do not cause a great enough alteration in the prediction scores of each user or group. The default personality is best suited for use where files exhibit a long-term of slowly-evolving access pattern.

The strict Virtual Secretary personality takes more account of the incorrect predictions that were made previously, but also has high thresholds for prediction scores. This results in the Secretary making fewer incorrect predictions (hence the reduction in the total number of small effort actions), but no overall increase in the number of correct predictions, meaning the total of large effort actions remains the same as when using the default personality.

In comparison, the lenient Virtual Secretary personality has much lower thresholds for making uncertain predictions. This reduces the number of large effort actions required by the user as the Secretary is more likely to correctly suggest a user or group to have access. Conversely, this also increases the likelihood that an incorrect prediction will be made and a small effort action will be needed to remove superfluous users from the access list. This clarifies why the lenient Secretary required more small effort actions than any of the other approaches. Displaying this behaviour means that the lenient personality is particularly suited to situations where access privileges change a great deal frequently.

6.3.10 Results

As expected, when users are left to remember and reason about a file access list over a period of days, they frequently make mistakes and took an increasing amount

Event	Confident		Uncertain		Action Required
	Write	Read	Write	Read	
1	C	C	-	-	-
2	C	C	-	-	-
3	C	C	-	-	-
4	C	C	-	-	-
5	C	C	-	-	Add P, F(Write, Read)
6	C	C	P, F	P, F	Remove F(Write, Read); Add H(Write, Read)
7	C	C	P, H	P, H	-
8	C	C	P, H	P, H	-
9	C	C	P, H	P, H	-
10	C	C	P, H	P, H	-
11	C	C	P, H	P, H	-
12	C	C	P, H	P, H	-
13	C, P	C, P	H	H	Remove P, H(Write); Add Group D(Read)
14	C	C	P, H	P, H, Group D	Remove P, H(Write)
15	C	C	-	P, H, Group D	Remove P, H(Read), Group D(Read); Add HBC(Read))
16	C	C	-	P, H, HBC, Group D	Add GroupBC(Read); Remove P, H, Group D(Read)
17	C	C	-	HBC, Group BC	-
18	C	C	-	HBC, Group BC	-
19	C	C	-	HBC, Group BC	Add EC(Read); Remove HBC(Read), Group BC(Read)
20	C	C	-	HBC, Groups EC, BC	Remove HBC(Read), Group BC(Read)
21	C	C	-	Group EC	-
22	C	C	-	Group EC	-
23	C	C	-	FDH, Group EC	Add P, H(Read); Remove Group FDH, Group EC(Read)

Table 6.24: Changes required for the access control list of the file using a lenient Virtual Secretary personality

Event	Manual	Default VS	Strict VS	Lenient VS
1	-	-	-	
2	-	-	-	
3	-	-	-	
4	-	-	-	
5	2 large	2 large	2 large	2 large
6	2 large	2 large	1 large, 1 small	1 large, 1 small
7	2 large	1 large	-	-
8	2 large	-	-	-
9	2 large	-	-	-
10	2 large	-	-	-
11	2 large	-	-	-
12	2 large	-	-	-
13	3 large	1 large, 2 small	1 large, 2 small	1 large, 2 small
14	3 large	1 large, 2 small	1 large, 2 small	2 small
15	1 large	1 large, 2 small	1 large, 2 small	1 large, 3 small
16	2 large	2 large, 2 small	2 large, 2 small	1 large, 3 small
17	2 large	2 large, 2 small	2 large, 2 small	-
18	2 large, 1 small	2 large, 2 small	2 large, 2 small	-
19	1 large	1 large, 2 small	1 large, 1 small	1 large, 2 small
20	1 large	1 large, 2 small	1 large	2 small
21	1 large	1 large, 2 small	1 large	-
22	1 large	1 large, 2 small	1 large	-
23	2 large	-	2 large	2 large, 2 small
Total	35 large	18 large	18 large	9 large
	1 small	20 small	14 small	16 small

Table 6.25: Effort required to create the desired access control list

of time to think about their answers. As shown in the previous case study, the Virtual Secretary can provide results in a linear time, so the recommendations made to the user do not take longer as the situation becomes more complex. The human counterparts in comparison were not able to answer in linear time. Even when using a computer to assist in the housekeeping of the file access list manually, mistakes can be made and more effort is required. Using a Virtual Secretary reduces the amount of effort needed to maintain the access lists of files and so is far more scalable than its human counterpart when dealing with large amounts of data.

6.3.11 Conclusion

The results of this case study support my hypothesis that a Virtual Secretary is more scalable when managing file-person relationships than human secretaries. As well as reducing the time needed to maintain the file distribution and access lists, the Virtual Secretary was able to help users avoid mistakes. In comparison, humans are more likely to take a long time remembering their past actions, reasoning over what action to take next and then arriving at an incorrect answer. This case study only looked at the effects of maintaining one file specifically. In a real-life scenario users would have to maintain hundreds of access lists for their individual files. Whilst this would be no problem for the Virtual Secretary, it would likely further decrease the performance of the human counterparts. The human management of people-file relationships has proved to be unscalable as many errors were made and the time taken to reach a decision increased, whilst the Virtual Secretary approach was shown to be scalable for administering this task.

6.4 Discussion

This chapter has demonstrated the ways in which a Virtual Secretary system could save the user time and effort. By performing two case studies, the scalability of the knowledge-based approach was evaluated.

In case study 1, a search was performed over synthesised datasets of varying time spans. Whilst a brute force search took up to several hours to complete, by implementing a caching technique these times were dramatically reduced. The addition of an archiving mechanism reduced these times further for datasets over 10 years in size. A consistent search time of around 10 seconds for any sized dataset was achieved by combining the archiving and caching methods. When the tests were repeated over the larger worst-case scenario datasets the same trends were displayed and the combinatory approach still produced the best search time (approximately 20 seconds). These results confirm the scalability of the knowledge-based approach used in the GIFS framework.

Case study 2 examined the scalability of the Virtual Secretary system with respect to maintaining file access lists. By showing the actions required by a user to share a file with various users and groups, a cumulative result effort was computed. Using three different Virtual Secretary personality settings, it was shown that the amount of time and effort required by the user could be reduced. The Virtual Secretary also averted potential mistakes that were made by the user when performing the actions manually or undertaken directly from the memory of users. The three different personality settings each made differing predictions to the user as to the most appropriate access list for a file. The example case involved a frequently-changing access pattern meaning that whilst all personalities offered an improvement over the manual method, the lenient personality yielded the best results. Whilst it was shown that humans were not scalable with respect to maintaining the human-file relationships, the Virtual Secretary was proven to be scalable, returning results in a consistent time and causing fewer errors than the human counterparts.

Both case studies have shown the suitability of deploying a Virtual Secretary system to assist users in the management and organisation of their files. Having now documented in detail the structure and design of the knowledge-based Secretary interface, the underlying file storage mechanisms are explored in the next chapter.

Chapter 7

System Level Development

7.1 Overview

The Virtual Secretary and the knowledge framework seen previously provide the functionality of a personal assistant. Behind this interface lies the file system, network transport and storage mechanisms that combine to produce GIFS. From a system viewpoint there are five constituent parts required to facilitate this global file system framework. The *Virtual Secretary* interface, *knowledge base(s)* and *agents* have already been studied in depth previously, although their location plays an important role in the overall architecture design. The program referred to as the *FileDB* (which originally stood for "file database", the name remained even after the design changed to incorporate more than just a database) acts as a gateway to the file store, administering file requests and user authentication. The *file store* contains the raw file data, however the FileDB must be used in order to locate the data successfully.

7.1.1 Technical Questions

From the discussions in previous chapters, we can clearly see that there is no current file system that can operate like what proposed for GIFS. This inevitably leads to an important technical question: "*is the proposed GIFS framework technically feasible?*"

In addition, there are numerous detailed technical questions, such as:

- What network architecture is the most suitable for supporting the GIFS framework?
- How should the constituent parts of the GIFS framework be distributed across a network architecture?

- How can multiple versions of files be maintained in an efficient manner?
- How can the system keep files secure but still deliver a transparent service?
- How can a key distribution mechanism be designed for use with file encryption?
- How are collaborative files kept both secure and accessible?

7.1.2 Assumptions

The following assumptions are made for the design and implementation of the file system of GIFS:

- Users of GIFS have access to a local or wide-area network.
- Transfers between the user's computer and the network are considered instantaneous. Network speeds are continuing to increase as they become more widely used, and should continue to increase at a rate greater than that at which file sizes are increasing.
- Users will require all of their files to be encrypted. Increased network usage means that users want to keep their files secure in both transit and storage. It is not an unreasonable assumption to make that the best way to keep file secure is through some form of encryption system.
- Users will require access to different versions of a file.
- Users will assign metadata to a file where they feel the automatic assignment was not sufficient. Although §3.3.3 showed that users are unlikely to add metadata to files, the system provides this facility in the event that users deem extra assignment necessary.
- Users are capable of remembering one password for authentication. As most computer or network systems at the current time require users to enter at least one password it would not be unreasonable to expect users to be able to remember a password in order to authenticate themselves with the system.
- Users will want to work collaboratively with other users on some files.
- Users will not want to remember more than one password or security key. The problems of information overload discussed in §3.1 can also be applied to passwords and encryption keys. Having several different passwords to perform various tasks on a computer is both time consuming and frustrating if forgotten. Hence it is desirable that the increase security of a file system does not increase the amount of effort a user has to expend in using that system.

- The network and other networked computers on the system are stable and reliable. As networks and computers improve in speed an efficiency, they are mostly very reliable. However, it is sometimes the case that computers fail or networks are inaccessible due to circumstances outside the user's control. It is assumed for the purposes of this work that the networks and computers mentioned are stable and always accessible, as obviously this is a major requirement for the system.
- Disk sizes can be considered limitless for the purposes of this study. As discussed before in §6.2.4.5, disk sizes are increasing at a rate greater than that of projected file or knowledge base sizes. Therefore, disk sizes are irrelevant to this area of our study.
- Users may wish to partially open files. Sometimes it may be desirable, especially with larger files, that user may want to edit the middle of the file rather than the beginning. In this scenario it is possible in some file systems to save time by opening only the sections of a file as they are required.
- All file management tasks are undertaken through the Virtual Secretary interface. As stated in the previous chapter, it is presumed that the Virtual Secretary interface will replace traditional file system user interfaces and in the future users will not have the specialised knowledge to manipulate the file system manually. Those users who are capable are assumed to know enough to realise that their actions will not be recorded by the GIFS framework and thus no new data will be placed into the knowledge bases.

7.1.3 Research Approaches

There are two basic approaches for the design and implementation of GIFS. The first is to build GIFS on top of an existing file system and add in the required additional components. The second is to design a new architecture, implement a number of key features and show that all the components in the design are feasible.

The features and concepts behind the idea of GIFS are significantly different to previously implemented file systems. Although it would be possible to add in some extra components and enhance a file system to exhibit a GIFS-like behaviour, the extra complexity caused by integration with commercial or unfamiliar code make the approach undesirable. It would also be difficult to insert the extra functionality that GIFS would require to display the full potential of the concept. The focus of this work was to prove the concept of a new file system rather than to present a fully function system by the end (as such an expectation would be well beyond the reach of a single PhD thesis), so the second approach of completely redesigning the architecture of a file system and implementing the components which are required to prove the feasibility of the framework was chosen. Both the design and implementation are discussed in detail in sections 7.2–7.4. Despite choosing this approach of a new concept for file system storage, previous technologies were still used where necessary in order to present a functional framework.

The GIFS framework makes use of many small, previously invented technologies as can be seen in Table 7.1. As they have been used in other systems and developed independently previously, there is no need in this work to prove their feasibility for use.

Technology	Reference
encryption	[253]
block-based file system	[130]
databases	[233]
knowledge bases	[167]
network security	[40]
collaborative work	[217]
versioning	[69]
suggestion agents	[175]
archiving	[223]
tagging	[41]
network file systems	[168]

Table 7.1: Previous technologies

Various encryption and hashing algorithms have been invented over the years [81, 286, 219] and the study of mathematics behind them continues to find weaknesses or propose newer, more secure techniques. For the purposes of this work, it is assumed that the chosen cryptographic functions are secure, even though there may be ways in which they can be broken. The GIFS framework uses encryption to store the user's files in a secure manner, as well as in the processes of user authentication, communication and key distribution.

A block-based file system allows segments (or blocks) of the file to be read without requiring the rest of the file. When saving new versions of files this saves space as only the changed blocks are required to be stored, not the whole file. ZFS [46] uses this method to improve the efficiency of writing to disk.

Network and distributed file systems have been used for many years [21, 58, 246] and the concept of storing files over a network is not new. However, none of these previous technologies have automatically managed the placement of files, instead acting as an extension to the local hierarchical file system. In the GIFS framework, the network is used to store the files but the user does not have knowledge of where on the network the file is stored. Users can however, assign version numbers to files if they wish. The Virtual Secretary automatically increments the version number for each subsequent file and also gives the user the option to increment the version number further to highlight major revisions. This is a relatively simple versioning system, although much work has gone into systems operating over collaborative environments to ensure the safe and effective use of multiple file versions by multiple users [217, 69].

Archiving is a basic operation for file and disk management to ensure efficient and stable file access. Whilst it may have been previously desirable to archive files [223, 222], the increases in disk sizes have reduced the need to save space. The GIFS framework employs archiving techniques to improve search efficiency and to optimise knowledge base and file access.

Databases are a mature technology, but so far there has been no operating system built utilising it closely. The nearest attempt of an operating systems was that of BeOS and BFS [111]. The concept of a knowledge base rather than a database is newer in comparison, but still a proven technique in the fields of data mining and discovery [210]. Knowledge bases are used within the GIFS framework to offer a flexible storage solution to the large amounts of data gathered about files and user actions as well as providing a convenient source of knowledge for the intelligent agents to work across. The agents are not only used to perform searches, but also to provide the user with recommendations or suggestions of actions or settings. Such agent systems have become highly popular since the AI renaissance in the mid 1990s with a wide range of applications employing agent technology [119, 29, 75, 195]. Adding extra data attributes to files (or "tagging") is used both in GIFS and other systems [41] to aid in the analysis, search and retrieval of data. There are many different methods for classifying what data should be used within tags, although this research area is still in its infancy. As such, in the name of extensibility, metadata assigned in the GIFS framework could be either user or system generated.

These technologies are used as the building blocks for the GIFS framework but are obviously not part of the novelty of the system. However, unlike in the GIFS framework, these technologies have not been combined to form an entire system previously, and have not been closely integrated with the structure of the operating system.

Although GIFS could feasibly work on a single machine with no network connection, one of the main focuses of this research is the distribution, transport and automatic storage of files over a network of any size. In order to implement the file system, the network and server architecture was carefully considered.

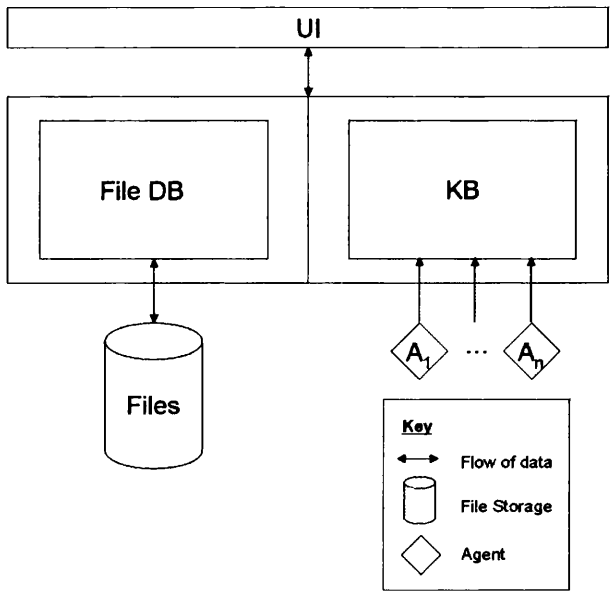


Figure 7.1: Single user with single computer architecture

7.2 System Design

7.2.1 Architectural approaches

When designing the framework and architecture for GIFS, several different options were reviewed, each with their own merits and drawbacks.

The simplest design can be seen in Figure 7.1. In this instance, the file system runs on one single machine and is not part of a network. The knowledge bases and files are both stored in their entirety on the local machine, and the data processing agents therefore also run on this local machine. This design is elegant in its simplicity but fails to address one of the major objectives of this work: network access to files and the scalability of the file storage capacity.

The next design, in Figure 7.2, removes everything except the user interface from the local machine. The files and knowledge bases are all stored on a server, separated from the user's local machine by a LAN or WAN. As the knowledge bases are stored remotely, the data processing agents are also situated there. Taking into account all users connected to the file server collectively, it increases the amount of network traffic needed, as almost all user interaction with the Virtual Secretary will have to access the relevant knowledge bases. The speed of the network would be one of the major bottlenecks in such a system and create problems with latency. The advantages of keeping all the data on the server are the decreased processing load

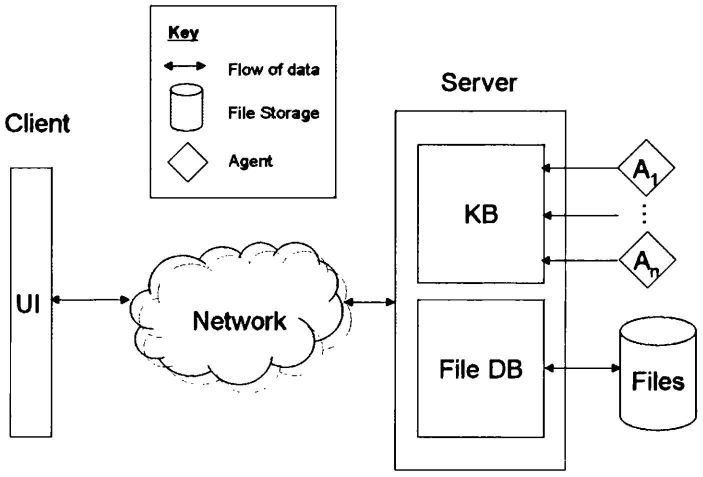


Figure 7.2: Single user with networked computer architecture

on the client machine and increased ease of key distribution for encrypted and shared files. The server will already have all the data available that it requires to generate new keys for each different user to access a file. However it would mean the keys are all stored in one central location, which creates a security risk as well as raising questions over the ownership of the data.

The design seen in Figure 7.3 is a multi-user extension of the system seen in Figure 7.2. This design does not scale well to support a large amount of users as there would be a great number of agent processes working over different data but running on the same machine. This would create an unmanageable workload for a single server and would increase the complexity of storing each user's knowledge bases separately in order to keep their data discrete.

To address the bottleneck created by the server being the repository for all data, the design in Figure 7.4 takes a half and half approach. The files are still stored on the server across a network, but the knowledge bases for each are stored locally. Whilst this has the drawback that the local computer will be running the agent processes and so will have a higher processing load, it means that basic knowledge base queries can be performed without the need for network communications. Although the local system will now have to request data from the server when making new keys for encrypted files and then distributing them to the relevant Virtual Secretaries, the keys can be stored locally giving the user extra reassurance that they still "own" their data.

The architecture in Figure 7.5 is a hybrid design. Whilst there still remains one single server for multiple users, the knowledge bases are now kept on the local ma-

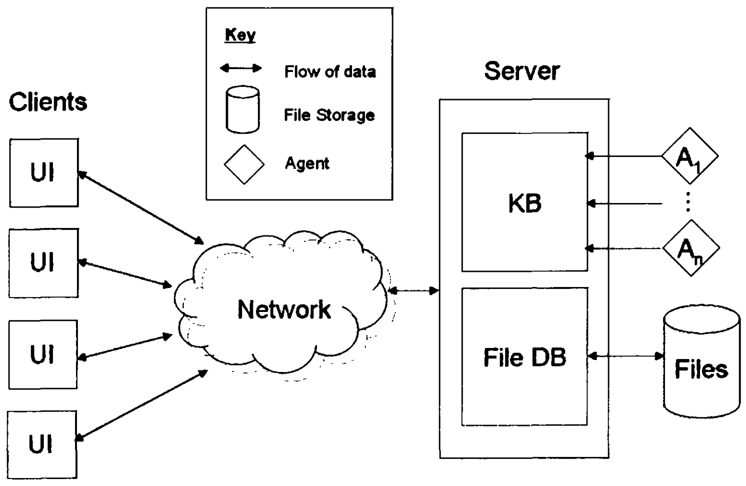


Figure 7.3: Multiple users with single server architecture

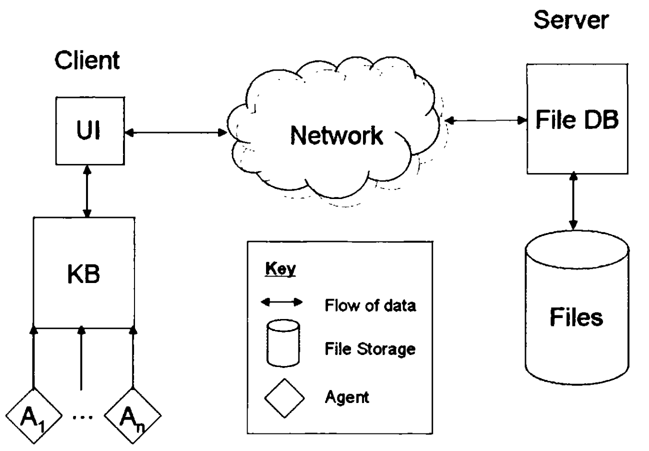


Figure 7.4: Single user with networked computer alternative architecture

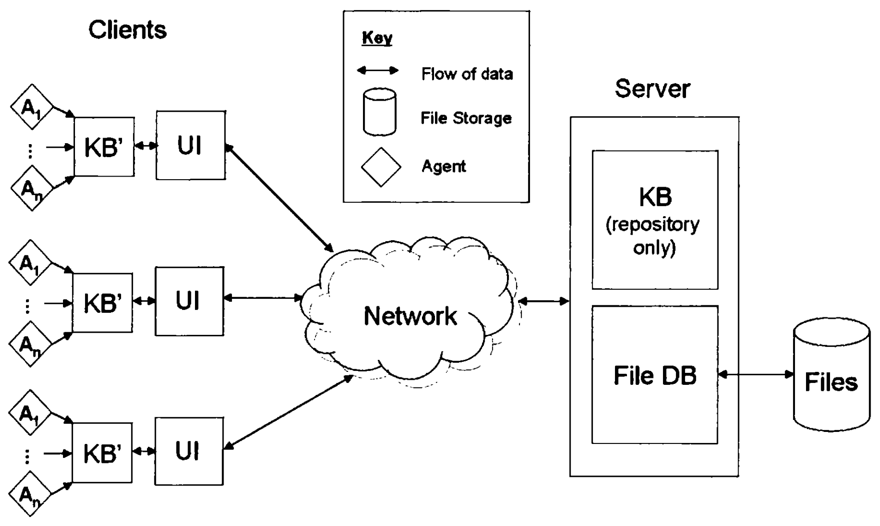


Figure 7.5: Multiple users with single server alternative architecture

chine, meaning that the agent processes are executed there also. The advantages of this approach are two-fold. Firstly the process load on the server is significantly reduced, and secondly the network traffic and bandwidth required is also lessened. A repository for knowledge bases is maintained on the server, providing a opportunity for back-ups as well as retaining certain pieces of data with respects to key management to help with the tasks of key distribution and creation.

In order to help the system scale for a large number of users, Figure 7.6 proposes a system where the server is part of or connected to a cluster of large capacity storage machines. Whilst this helps with the space needs of many users, it does create a bottleneck of a single server gateway through the network and back to the client machine. This design also introduces the need for a more complex file placement and retrieval strategy within the server.

The final design in this section shows one (or more users) connected to a set of servers, each with their own FileDB program and file storage space. As seen in Figure 7.7 this is similar to the concept of a totally distributed file system. The loads on each server can be balanced to share the time and space needed for access by a large amount of users. There is an extra layer of complexity due to the number of servers unless the data is to be duplicated across each of them. If a user connected to one server in one session and stored a file, unless the data was then distributed to the other servers, the user would have to connect to the same server again in order to retrieve that particular file. Distributed systems have long been an active area of research, and remain a very complex type of network storage.

Although a distributed file system has many advantages the extra complexity that would be added in terms of protocol design, balancing mechanisms and so forth

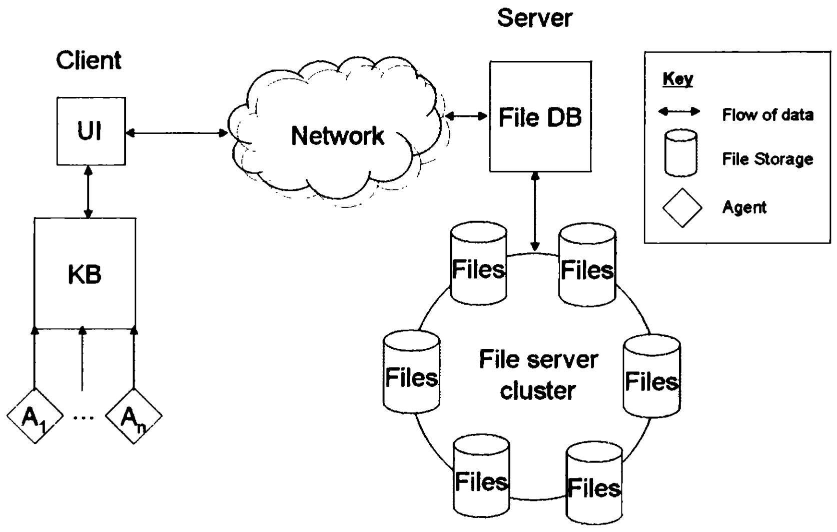


Figure 7.6: Clustered server architecture

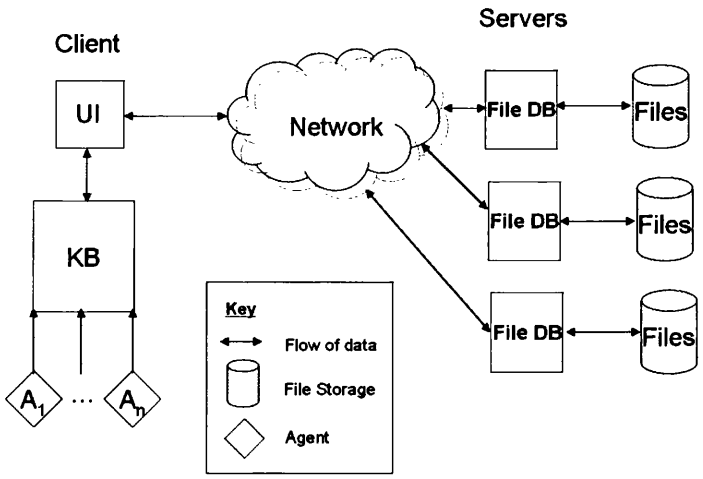


Figure 7.7: Distributed server architecture

make it an unsuitable architecture to use for the purposes of his research. Keeping the knowledge bases on the user's local machine would allow the information and knowledge to be kept separate from other users as well as enabling them to keep "ownership" of their data. By using the design seen in Figure 7.5 the system retains the advantages of local knowledge base placement whilst also simplifying other processes (such as encryption) by keeping a small repository of knowledge on the server. It would be possible to have a file cache on the client, similar to those used in any modern distributed file system or web browsers. However, as this is a proven concept, it was not implemented in order to focus on the core design issues.

7.2.2 File placement

As mentioned in previous chapters, the user has neither the knowledge or control over the physical location (on disk) of their files. This allows the file storage mechanism to be designed for the ease of incorporating other features. In order to provide an elegant solution for the versioning and encryption systems and also minimise the space required for file storage, each file is split into blocks.

7.2.2.1 Allocation Units

Hard disk drives are typically accessed as block devices and this abstraction is maintained by GIFS. Allocation units, or blocks can be a variety of different sizes within file systems dependent on disk architecture, in GIFS they are 4096B. Starting at byte 0, a file is split into sequential 4096 byte chunks with the final block being padded if necessary in order to be of an equal size. The blocks are then stored independently of each other and are reassembled in order if the complete file is needed. When a file is encrypted (the process of which will be demonstrated in §7.2.4.1), each block is encrypted individually.

7.2.2.2 Versioning

Storing a new version of an entire file each time a modification is made is an inefficient use of disk space and network bandwidth. A better solution would be to only store those parts of the file that have changed. By splitting files into blocks as described above an effective storage mechanism for multiple-versioned files can be implemented within the FileDB.

For each version of a file, the FileDB stores the file ID, version number and the date/time of addition. These values will correspond to those in the knowledge base of the Virtual Secretary from which the request originated. In addition, a variable-length binary string (referred to as a bitfield) is also stored. The bitfield will have a length equal to the number of blocks a file has been split into (i.e. the size of the file divided by the block size). Each bit in the bitfield represents whether or not the corresponding block in this version of a file has been modified since the previous version. If there have been changes, a 1 is stored in the appropriate position, if there were no changes a 0 is stored.

When a user requests a certain block of a file (usually the beginning two blocks, but conceivably also specific mid-file blocks), the FileDB looks at the latest entry for that file ID in a table contained within the FileDB. If for the required block the the corresponding position in bitfield is set to 1, then the last version of the file has the most recent data for this block and so the block is retrieved from the file system. If the value is 0, there were no changes to this block in the last revision of the file, so the entry for the previous version is examined from the FileDB. The value of the bit in the appropriate position in this bitfield is checked. This process continues until for all required blocks the last occurrence of a 1 in that position has been identified. The bitfield of the chronologically first entry will always be comprised entirely of 1's, as all the bytes in this file will have changed during creation. If a user requests a specific version of a file, the FileDB looks up the bitfield for that version and checks each of the required blocks. If any of them are 0, it then begins to work backwards through the previous bitfield entries as described above. Using this method, only the FileDB knows of the relationship between all versions of any file and avoids duplicating the entire file on every revision.

This technique is similar to the copy-on-write strategy of ZFS. When the contents of a file is modified, it writes the blocks that have been changed to a different location. The metadata is then updated to show where the new versions of the blocks can be found. If the metadata is never written (due to an error) then the old version of the file will still be recoverable as the original blocks will have not been overwritten. Since the data is not overwritten on an update, snapshots are an O(1) operation in both GIFS and ZFS.

If a user has write permission for a file they are able to upload changed blocks in order to be included in the revision history. To ensure only one person is editing a file at any one time a simple check-in/out flag is used. If two people decide to edit the same file at the same time despite warnings then the two resulting files are forked, and either some external software or human intervention is needed to unite them back into one file again.

7.2.2.3 File Placement

As each file is split into equal-sized blocks, there are now many equal-sized chunks of data to store and retrieve. In GIFS, as the user has no control over the physical location or format of their files, they can be stored in the most convenient and efficient way for the computer to maximise performance. Several different options were considered, including the arrangement of files in a one-directory-per-version basis, a one-directory-per-user basis or an equal distribution algorithm. The first two approaches would have resulted in an unbalanced and inefficient directory structure but would simplify the naming/location process. Placing the file blocks in an equally-distributed directory structure would have improved efficiency but added extra complexity to the naming and location of each data block. As each block was of an equal size, it was decided to store them in a database.

As no inbuilt database currently exists on the Windows platform, an external database was used. The advantages of using a database from a system designer's perspective are that location and retrieval mechanisms for data will be pre-optimised, the storage mechanism will be automatically handled so as to provide the fastest access for all entries, and multiple users and queries can be serviced simultaneously. More simply, a database will provide the most optimal outcome with a greatly reduced implementation effort.

As a database already forms part of the FileDB program, the file store can be easily incorporated there. Although both the FileDB and the file store share the same database, they should still be viewed as conceptually different parts of the architecture, in the same way that the FileDB program is more than just a database.

Each record in the file store represents one block of a file and holds 4 fields required for file storage tasks. The raw data will be held in one field and will obviously have a fixed size of 4096 bytes. The file ID will be held in another field, along with the block number to identify the correct position for the data within the file. The date and time at which this block was added is also stored.

7.2.3 Communication Protocols

In order for the Virtual Secretary to communicate with the server and other Secretaries, a communications protocol was designed. Using an XML schema, several different types of message and response were defined. The extensible nature of XML means that in the future new message types can be added without constraining the older versions.

The XML schema is used not only to define the protocol, but also to check that each message is well-formed and valid. The full XML schema to define all message types can be found in Appendix B, Listing B.2, meanwhile a few sections of the schema are examined below.

Within the schema there are two major types of messages: requests and responses. Listing B.1 in Appendix B gives the definition of the request type, whilst a pictorial interpretation can be seen in Figure 7.8. The request message has a complex type which is composed of other elements and attributes. The choice tag allows one and only one element to to selected. Therefore a request element can contain either: a

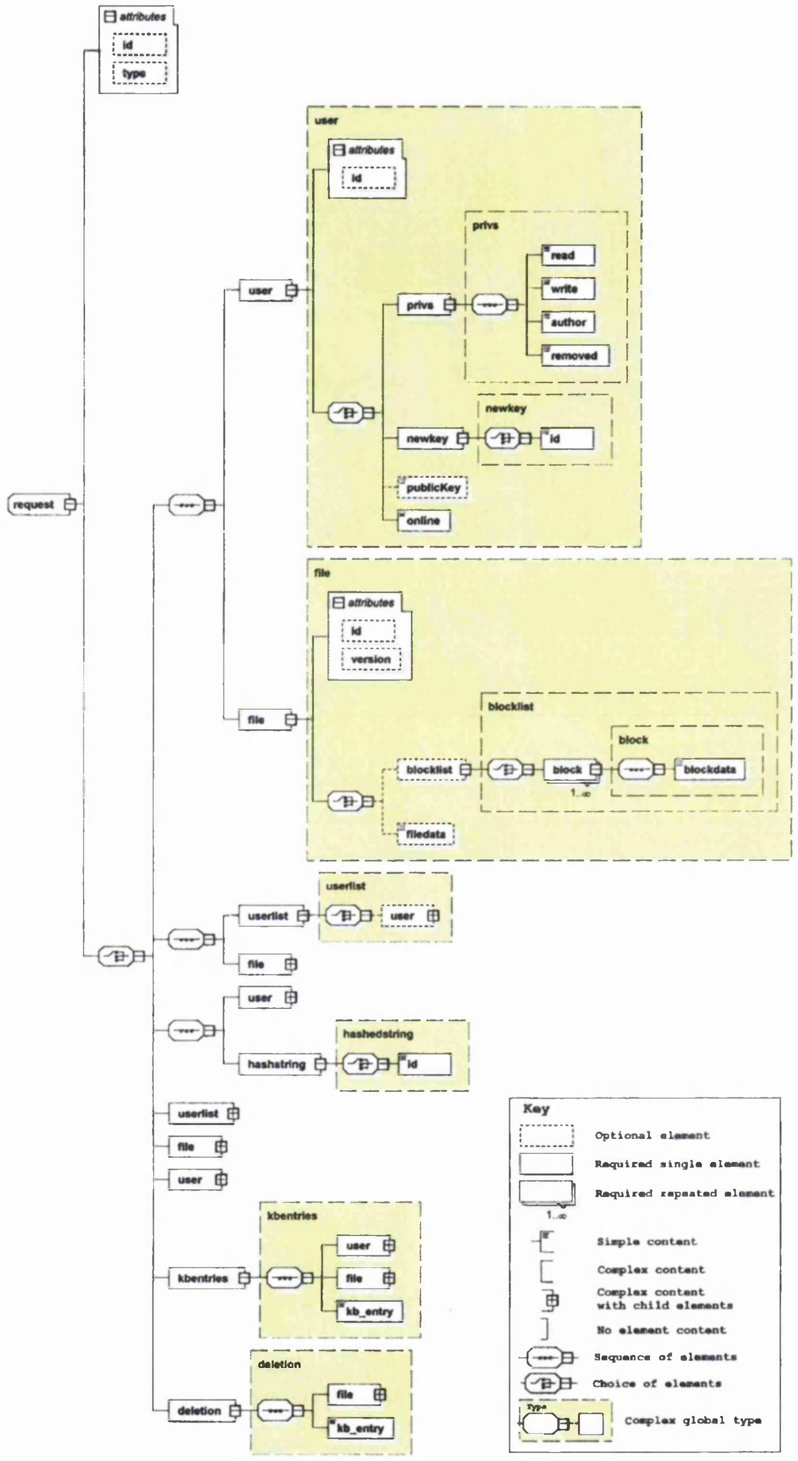


Figure 7.8: The request element from the XML schema

user and file element; a userlist and file element; a user and hashstring element; a userlist element; a file element; a user element; a kbentries element; or a deletion element.

The sequence tag specifies the order in which elements have to occur. Therefore if a request element did contain the first sequence of elements displayed in the choice tag, the user element would come first, followed by the file element.

The attributes of "id" and "type" that come after the choice tags can both be included and refer to the attributes of the request message. The "id" attribute contains the identification number of the type of message as an integer value, and the "type" attribute contains the string representation of the corresponding request type. The list of message ids and request type can be seen in Table 7.2.

ID	Type
1	request_key/return_keys
2	store_file
3	request_file/respond_file
4	acknowledge
5	permissions
6	kb_entries
7	logon
8	authenticate
9	online_status
10	deletion

Table 7.2: Message id and type values

Listing 7.1: Section of XML Schema defining the response message
```
<xsd:element name = "response" type="response"/>
<xsd:complexType name="response">
    <xsd:choice>
        <xsd:element name="userlist" type="userlist"/>
        <xsd:element name="file" type="file"/>
        <xsd:element name="valid" type="xsd:string"/>
        <xsd:element name="logon" type="xsd:string"/>
        <xsd:sequence>
            <xsd:element name="file" type="file"/>
            <xsd:element name="userlist" type="userlist"/>
        </xsd:sequence>
        <xsd:element name="string" type="string"/>
    </xsd:choice>
    <xsd:attribute name= "id" type="xsd:int"/>
        <xsd:attribute name= "type" type="xsd:string"/>
</xsd:complexType>
```

The response element defined in Listing 7.1 shows the structure of a message sent in response to a request message. As before, the choice tag denotes that only one element (or sequence) from inside the list can be selected. So a response element will contain the attributes of "id" and "type" which contain the response type integer and the string definition (e.g. "4" and "acknowledge" respectively), plus any one of the following: a userlist element; a file element; a valid element; a logon element; a name element; or a sequence containing a file elements and a userlist element.

The file element seen in the response and request elements is defined in Listing 7.2. As before, this element has two attributes: "id" and "version". The id attribute denotes the id of the file as set in the knowledge base. The version attribute is an optional integer value denoting the version number of the file and does not have to be included in every message. One other element appears in the file element, either a blocklist element or a filedata element. The full definitions of the filedata and kbentries elements can be found in Appendix B, Listing B.3 and are reasonably self-explanatory after the element definitions already seen. The blocklist element (which is used to transport the raw file data blocks) can be seen in Listing 7.3. A blocklist element can contain only one element type, a "block". The "minOccurs" and "maxOccurs" values denote the range of occurrences acceptable within this element. Logically, a blocklist element must therefore contain at least 1 block element, but does not have an upper size limit.

Listing 7.2: Section of XML Schema defining the file element

```
<xsd:complexType name = "file">
    <xsd:choice>
        <xsd:element name = "blocklist" type = "blocklist"
           minOccurs="0"/>
        <xsd:element name = "filedata" type = "xsd:string"
           minOccurs="0"/>
    </xsd:choice>
    <xsd:attribute name = "id" type="xsd:string"/>
    <xsd:attribute name = "version" type="xsd:int" use="
       optional"/>
</xsd:complexType>
```

Listing 7.3: Section of XML Schema defining the blocklist element

```
<xsd:complexType name = "blocklist">
    <xsd:choice>
        <xsd:element name = "block" type = "block" minOccurs
           ="1" maxOccurs="unbounded"/>
    </xsd:choice>
</xsd:complexType>
```

Examples of communication messages created by using the XML schema can be seen in §7.4.

7.2.4 Encryption and Security

To offer a secure service to users it was decided that each individual block of a file should be encrypted before transmission across the network and storage on the server. However, it was equally important that the encryption mechanism should be transparent to the user. Within GIFS, several different types of encryption and hashing algorithms are used.

7.2.4.1 File Encryption

The versioning system, based on splitting files into allocation units, was designed not only to decrease the amount of space needed to store newer version of files, but also in order to allow each block of a file to be encrypted separately.

Upon the creation of a file, the Virtual Secretary of the originating author produces a very long symmetric AES key. AES is a standardised block cipher [81], the key for which is produced by the random number generator on the client machine. This key is then used to encrypt every block of the file. Each user has a pair of keys (public and private) for use in an asymmetric encryption algorithm. The newly-created AES key is encrypted by the user's public key which is stored on the server. This encrypted key is kept within the user's knowledge base.

If the file is to be shared with other users, they will each require an encrypted AES key in order to decrypt the file. The Virtual Secretary requests the public keys of each user from the FileDB and uses each one to create an encrypted version of the AES key. These keys are then either sent back to the FileDB for distribution or sent directly to the relevant Secretaries.

The FileDB also stores the user's private key which is required to decrypt all their AES keys. Storing the public and private key pairs in the same location would not be secure, so the private key is stored in an encrypted format. The user's password is hashed using a known algorithm which creates the symmetric key required to encrypt the private key. The choice of hash algorithm is not important provided that it is different to the one used during the user authentication process (See §7.2.4.2) as the outcome of that operation is stored on the server. The whole encryption process for two users can be seen in Figure 7.9. The advantage of this encryption system is that it is transparent to the user and they are only required to remember one password for the whole system.

If the user wishes to change their password, their private key must be decrypted using the old password hash and then re-encrypted using a hash of the new password. The AES keys for each file do not need to be recalculated unless the user suspects that their password has been compromised. In this event, the password, private and public keys must be changed and then used to create replacement encrypted AES keys for each file. The AES key itself would stay the same so access to the files for other legitimate users would be unaffected.

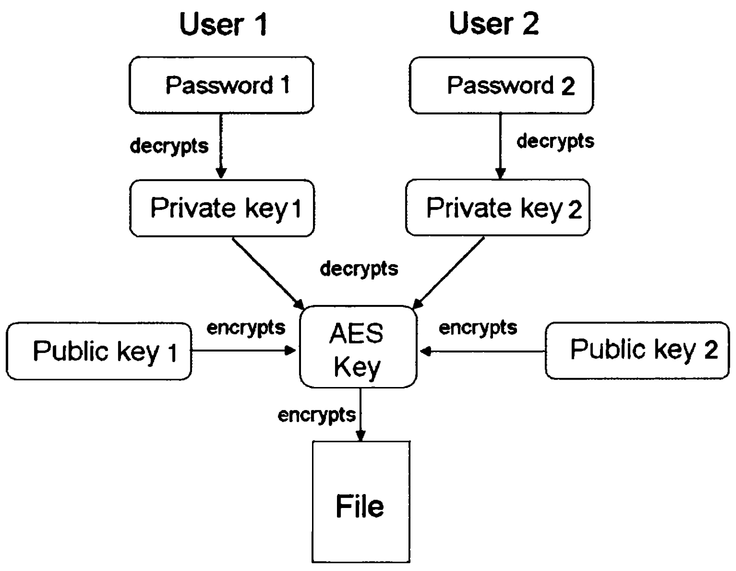


Figure 7.9: File encryption methods within GIFS

7.2.4.2 User Authentication

Upon first use of the Virtual Secretary, the user's password is hashed using a known algorithm and sent to the FileDB for storage on the server. When the user starts their Virtual Secretary it sends a logon message to the server. The server replies with a randomly generated string. The user is prompted to enter their password which is hashed using a chosen algorithm and concatenated with the random string sent by the server. The resultant string is then hashed using the same algorithm again, and sent by the Virtual Secretary to the server. The server will have performed the same operations as the Virtual Secretary (but using the hashed version of the password it has stored) and so the two strings should be equal. If the two values match, the user is authenticated. As the password is stored on the server in a hashed form it is important that two different hashing algorithms are used for user authentication and file encryption purposes.

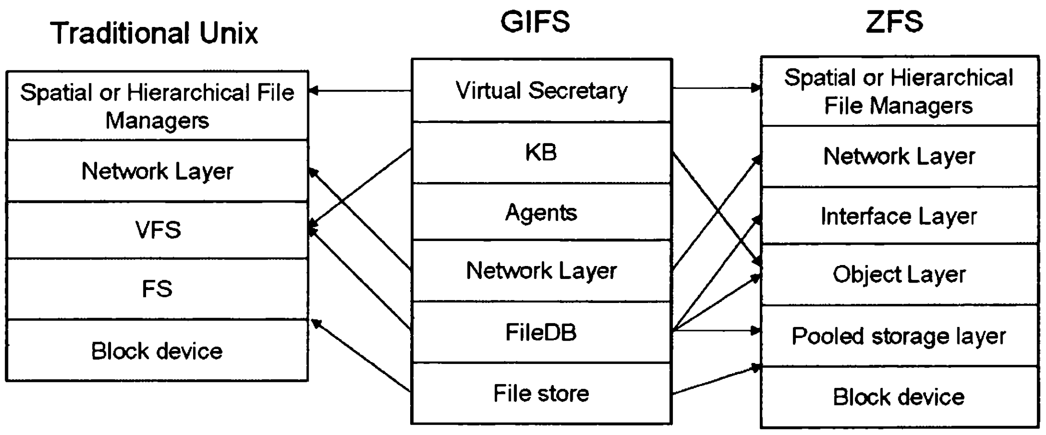


Figure 7.10: Comparison of layers in a Unix file system, GIFS and ZFS

7.3 System Level Implementation

Figure 7.10 shows roughly how the layers in the GIFS architecture correspond with those in a traditional Unix file system and Sun's ZFS. In both cases the Virtual Secretary replaces the traditional hierarchical file managers. The knowledge base holds all the metadata on the files, along with the extra data, information and knowledge gathered and deduced by the agents. As well as processing data, the agents provide the bridge to the network layer. This layer is directly comparable to the network layer of both traditional Unix file systems (such as CIFS in AFS) and ZFS (NFS). The FileDB (discussed in more detail in 7.3.1) controls access to the file data, user authentication and knowledge dissemination. The file store contains the raw file data.

7.3.1 File Database

The FileDB on the server has several purposes. It stores the files, controls access and permissions, manages user authentication, content delivery and knowledge dissemination. It is implemented in C# on top of a database (Microsoft SQLServer), however the name database does not adequately encompass the range of services the FileDB offers but is used as a generalisation in this instance. Databases are currently seen as an additional program within an operating system, but ideally for GIFS it would be included in the lower level code of the operating system, improving performance.

The interface program on the server that forms part of what is classed as the FileDB parses the communications from the Virtual Secretary clients, requests data from the database and returns appropriate answers back to the Virtual Secretaries. The requests for data within the FileDB are all relatively simple, as the majority of heavy data-processing occurs on the client in the knowledge base, which has a much richer collection of information. Some information is duplicated between the knowledge bases and the FileDB, however this is for extra security checking and version management only and the server does not provide any knowledge itself. Simple queries are passed to the database by the FileDB wrapper program, where the more complex operations are performed.

One of the advantages of using a pre-packaged database is the efficiency of searches. SQLServer automatically balances the B-Tree structures of each table it holds data on, meaning that search times are kept to a minimum. For this reason, the segments or blocks of a file are actually kept within the database itself as it would be considerably less efficient in this implementation to keep them in an alternative position on disk without access to the native file system or operating system source code.

There are 5 tables within the file database, each of which is shown along with a brief explanation of the fields in Tables 7.3, 7.4, 7.5, 7.6 and 7.7.

Field name	**Description**
FID	The file ID
dateTime	The date and time that this version was created
bitField	An array of bits that show which blocks have changed since the last version
permsChanged	A true/false flag to show if someone's permissions have been changed since this version and the new blocks require a new AES key
version	A numerical indication of version

Table 7.3: The version table

Field name	**Description**
UID	The user ID
FID	The file ID
permissions	Can be A for authoring/ownership, R for read only, W for write, or REM for removed
dateTime	The date and time these permissions were given/changed

Table 7.4: The permissions table

7.4 File Lifecycle

There follows a closer look at the operations of the FileDB and the server. First of all, the basic variables and concepts are introduced to give a better grasp of the more technical procedures in the example.

Field name	Description
UID	The user ID
password	The user's password, hashed using MD5
publickey	The user's public key

Table 7.5: The users table

Field name	Description
FID	The file ID
UID	The user who is currently editing this file
dateTime	The date and time this file was checked out
flag	A boolean value showing whether or not this file has been returned
version	A numerical value showing which version of the file was checked out
duplicate	A boolean value denoting if this file was checked out >1 times without return

Table 7.6: The checkout table

Field name	Description
FID	The file ID
blockno	The number (position) of this block
dateTime	The date and time this file was added
blockdata	The binary data for this block

Table 7.7: The files table

7.4.1 Variables

In this example the real-life values of variables have been replaced by text that is not only easier to read, but will help the reader follow the logical path of the processes occurring. The definitions of variables and naming conventions set out in Chapter 5 are used within the system as specified but may have been substituted by other non-compliant values for the purposes of the example.

There are 5 users of the system, named userA, userB, userC, userD, and userE. These values will replace the user/Virtual Secretary IDs (UID) which would usually be a 'meaningless' string of characters. The file under scrutiny in this example is known to the users as "Budget.xls", although this is stored along with the rest of the metadata in the knowledge base on the client's machine and so is not needed here. The FileDB refers to a file by the file ID (FID), which matches the FID value set for this file in the knowledge base. The users are unaware that their files have IDs and even less the knowledge of the value of the ID string. Within this example the FID is "000000000001" and any variable or value followed directly by the character "#" is in an encrypted format and must be decrypted before it can be processed.

All the metadata for this file is placed in the knowledge base on the client machine. Some of this data is also duplicated in the FileDB to assist in versioning and access control. These are fields such as the file size/block total (needed to facilitate the block mechanism), the file creation time, and access privileges for each user. The different blocks of each file are stored within the FileDB. Hence there are no traditional network paths. Each file that is sent to a user is created on-the-fly, using the required blocks for the specified version.

7.4.2 File Example

Using the variables set out above, the lifecycle of a file as it passes through the system can be observed. This includes examples of communications between the Virtual Secretary and the server, the internal workings of the FileDB and the processes required to facilitate the file storage of GIFS.

7.4.2.1 User Authentication

The Virtual Secretary starts a session with the server by sending a logon request message which contains an identifying user ID as seen in Listing 7.4.

Listing 7.4: Logon request to server

```
<request id="7" type="logon">
   <user id="userA" />
</request>
```

The server acknowledges this request (shown in Listing 7.5) by generating a random string and sending it back to the Virtual Secretary.

Listing 7.5: Logon reply to the Virtual Secretary

```
<response id = "7" type="logon">
    <string id="arandomstring" />
</response>
```

The Virtual Secretary hashes the user's password using MD5 and concatenates it with the random string, and then performs an MD5 hash on the resultant string. Listing 7.6 shows the messages used to send the string to the server.

Listing 7.6: The request message containing the hashed password

```
<request id="8" type="authenticate">
    <user id="userA" />
    <hashstring id="veryhashedupstring" />
</request>
```

The FileDB looks up userA's hashed password in the user's table (Table 7.8), concatenates it with the random string it sent previously, hashes it using the same hash function as the Virtual Secretary (e.g. MD5) and then compares this string with the one it just received. If the two strings are the same, the logon is successful and the

UID	password	publickey
userA	passworda#	pkA

Table 7.8: User A's entry in the user's table

server sends an acknowledgment message back to the Secretary, or an error message if the password was incorrect. Both messages can be seen in Listing 7.7.

Listing 7.7: Logon acknowledgment message

```
<response id = "4" type="acknowledge">
    <logon>true</logon>
</response>
or
<response id = "4" type ="acknowledge">
    <logon>false</logon>
</response>
```

7.4.2.2 File Creation

UserA creates a file called "Budget.xls" and instructs their Secretary to file it away. The file is split into allocation units (or blocks) of 4096B in size. The Virtual Secretary generates an AES key which is used to encrypt each allocation unit. The AES key is encrypted with userA's public key and stored locally. The Virtual Secretary then sends the file to the server, using the message seen in Listing 7.8.

Listing 7.8: Sending a file to the server

```
<request id = "2" type="store_file">
    <file id="000000000001">
        <blocklist>
            <block id="1">
                <blockdata>encryptedblockdata1</blockdata>
            </block>
            <block id="2">
                <blockdata>encryptedblockdata2</blockdata>
            </block>
            ...
            <block id="16">
                <blockdata>encryptedblockdata16</blockdata>
            </block>
        </blocklist>
    </file>
</request>
```

The FileDB parses this message and extracts the necessary information from it. A new entry is inserted into the version table, shown in Table 7.9.

FID	dateTime	bitField	permsChanged	version
000000000001	1234567	1111111111111111	False	1

Table 7.9: The new entry into the version table

Once this process has been completed successfully, an acknowledgment message is sent back to the Virtual Secretary. The file is now stored securely on the server, but cannot be accessed as the permissions have yet to be set.

7.4.2.3 Granting Access Permission

UserA is the original author and owner of the file and thus has full access permissions. They also wish to share the file with userB, userC, userD and userE and additionally allow userB to modify it. There are several smaller steps needed to achieve this, the first of which is seen in Listing 7.9, the request for public keys from the server. Note that userA is omitted from this step, as their Virtual Secretary already knows the public key from within their knowledge base. Once the FileDB has processed this message, it queries the database for the public keys of the specified users, and finds the entries shown in Table 7.10. The FileDB then returns the keys to the Virtual Secretary in the format shown in Listing 7.10.

Listing 7.9: An XML message requesting public keys

```
<request id="1" type = "request_key">
    <userlist>
        <user id = "userB">
            <publicKey />
        </user>
        <user id = "userC">
            <publicKey />
        </user>
        <user id = "userD">
            <publicKey />
        </user>
        <user id = "userE">
            <publicKey />
        </user>
    </userlist>
</request>
```

UID	password	publickey
userB	passwordb#	pkB
userC	passwordc#	pkC
userD	passwordd#	pkD
userE	passworde#	pkE

Table 7.10: Entries for users B,C,D and E in the users table

Listing 7.10: Returning the public keys to the Virtual Secretary

```
<response id="1" type = "return_keys">
    <userlist>
        <user id ="userB">
            <publicKey>pkB</publicKey>
        </user>
        <user = id "userC" >
            <publicKey>pkC</publicKey>
        </user>
        <user id ="userD">
            <publicKey>pkD</publicKey>
        </user>
        <user id = "userE">
            <publicKey>pkE</publicKey>
        </user>
    </userlist>
</response>
```

Once it has received the necessary public keys, the Virtual Secretary uses each one to encrypt the AES key that was used to encrypt the file. There are now 5 encrypted keys for this file, each one only decrypted by the respective users' private key. The Virtual Secretary builds a message to denote the permissions for the file. A section of this message can be seen in Listing 7.11, whilst the full message can be found in Appendix B in Listing B.3.

The FileDB inserts this new data into the permissions table of the database, shown in Table 7.11.

Listing 7.11: Partial permissions message

```
<response id="5" type = "permissions">
    <file id ="000000000001" />
    <userlist>
        <user id="userA">
                <privs>
                    <read>true</read>
                    <write>true</write>
                    <author>true</author>
                    <removed>false</removed>
                </privs>
            </user>
        <user id="userB">
                <privs>
                    <read>true</read>
                    <write>true</write>
                    <author>false</author>
                    <removed>false</removed>
                </privs>
        </user>
        ...
    </userlist>
</response>
```

UID	FID	permissions	dateTime
userA	000000000001	WRA	1234567
userB	000000000001	WR	1234567
userC	000000000001	R	1234567
userD	000000000001	R	1234567
userE	000000000001	R	1234567

Table 7.11: New data inserted into the permissions table

Once this process is complete, the FileDB sends back a valid acknowledgment message as in Listing 7.12.

Listing 7.12: An XML acknowledgment message

```
<response id = "4" type="acknowledge">
    <valid>true</valid>
</response>
```

In the case of a new file, the above process for altering permissions is performed at the same time the file is created, but permissions can also be modified in the same way at any subsequent point in the future.

7.4.2.4 Sharing Knowledge

When a new file is available to multiple users their Virtual Secretaries will be unable to find it unless the appropriate knowledge is added to each users' knowledge base. In order to distribute the new knowledge and keys, the Virtual Secretary sends a message (an example of which is shown in Listing 7.13) to the server requesting the online status of the other Secretaries. The empty online tags will be filled in by the FileDB with the IP addresses of the online Secretaries, or left blank otherwise.

Listing 7.13: An XML online status request

```
<request id="9" type = "online_status">
    <userlist>
        <user id ="userB">
            <online/>
        </user>
        <user = id "userC" >
            <online/>
        </user>
        <user id ="userD">
            <online/>
        </user>
        <user id = "userE">
            <online/>
        </user>
    </userlist>
</request>
```

The FileDB will check to see which Secretaries in the user list are currently online. Those that are will receive a message containing the knowledge to be added to the knowledge bases seen in Listing 7.14. The messages for the Secretaries that are not online are kept on the server until the next time the Virtual Secretary connects. To reduce the load on the server it is possible for Secretaries that are online to communicate directly, This is referred to as "gossip", whilst the messages for offline Secretaries sent to the server as described above.

Listing 7.14: An XML knowledge distribution message

```
<request id="6" type="kb_entries">
    <kb_entries>
        <user id="userB">
            <newkey id="encryptedkeyB">
            </newkey>
        </user>
        <file id="000000000001" version="0"/>
        <kbentry>KB-entry-string</kbentry>
    </kb_entries>
</request>
```

7.4.2.5 Informing Virtual Secretaries

If a file is to be viewed by multiple users, their Virtual Secretaries must be informed of the required key for decryption, not only facilitate the addition of the newly created knowledge to enter into the knowledge bases. As with the sharing of knowledge, the keys can be sent to the Secretaries via the server or directly. The key data is sent in the same message as the new knowledge base entries.

7.4.2.6 Editing a File

UserB wishes to modify the file, so begins by requesting the most recent version from the server (Listing 7.15).

Listing 7.15: An XML message requesting a file from the server

```
<request id="3" type="request_file">
    <user id="userB" />
    <file id="000000000001" version="0" />
</request>
```

The FileDB checks what permissions userB has for this file, shown in Table 7.12. It should not be possible for a user to request a file that they do not have access to as their knowledge bases will not contain the required information. However, performing this extra check on the server helps prevent unauthorised access from external sources.

UID	FID	permissions	dateTime
userB	000000000001	WR	1234567

Table 7.12: User B's entry in the permissions table

As userB has write permission an entry is inserted into the checkout table to show the file is being edited as in Table 7.13.

FID	UID	dateTime	flag	version	duplicate
000000000001	userB	7654321	True	1	False

Table 7.13: The new entry into the checkout table

The FileDB then looks at bitfield for this file, shown in Table 7.14, in order to start assembling the correct version of the file to be sent back to userB. As all the positions are in this bitfield are '1', the most recent version of each is required to assemble the complete file (Listing 7.16). It would be desirable to save network traffic if only the blocks of the file requested (e.g. the first two for the opening of a file or other specific mid-file blocks) were sent to the Virtual Secretary, however at this time the whole file is needed in order for the client computer to be able to open the file. The storage and transport mechanisms have been designed so that the introduction of a virtual file system on the client will not result in major changes to the schema, FileDB or file store.

FID	dateTime	bitField	permsChanged	version
000000000001	1234567	111111111111111	False	1

Table 7.14: The most recent entry in the version table

Listing 7.16: An XML message sending a file to a VS

```
<response id="3" type="respond_file">
    <file id ="000000000001">
        <filedata>allthefiledatablocks</filedata>
    </file>
</response>
```

The Virtual Secretary client uses userB's password to decrypt their private key, which in return is used to decrypt the AES key from their knowledge base. The AES key is used to decrypt each of the blocks in turn. Once the user has finished modifying the file, the Virtual Secretary compares each new block with the old ones. Only if they have changed are they sent back to the server. In this case, userB has only made changes to blocks 3 & 4, so they are the only blocks that need re-encrypting for sending back to the server. Blocks 3 & 4 are encrypted with the AES key that was used for decryption and the file is then sent back to the server as seen previously, along with the updated knowledge base entries.

The FileDB checks that userB has write access for this file, and that it is not currently 'checked out' by anyone else with write access. The version table is updated with

FID	dateTime	bitField	permsChanged	version
000000000001	1234567	1111111111111111	False	1
000000000001	8765432	0011000000000000	False	2

Table 7.15: The contents of the version table after a new version has been added

the FID and the new date/time and bitfield, the result of which is shown in Table 7.15.

The file is checked back in by editing the entry in the checkout table. Once the file is checked back in, an acknowledgment message is sent back to the Virtual Secretary. The FileDB also distributes new version knowledge received from the Virtual Secretary to each user who has access to the file.

7.4.2.7 Reading the Latest Version of a File

After many versions of the same file have been created, the correct assembly of the file on the server becomes more important. All the entries for the file in this example can be seen in Table 7.16.

FID	dateTime	bitField	permsChanged	version
000000000001	1234567	1111111111111111	False	1
000000000001	8765432	0011000000000000	False	2
000000000001	9876543	0010010000000000	False	3

Table 7.16: The entries for this file in the version table

UserA has requested the most recent version of this file, so the FileDB queries the database for the most recent version in the version table. Blocks 3 & 6 were modified, so version 3 of block 6 & 3 are needed. Then the previous entry is checked, which shows that blocks 3 & 4 were modified. However, we already have a more recent version of block 3, so from this version only block 4 is needed. The FileDB checks the next previous bitfield, where all the other blocks (1,2,5,7-16) were modified so are required. The file and block string is constructed and sent to the Virtual Secretary as seen in a previous example.

7.4.2.8 Modifying Access Permissions

The permissions of a file can be modified at any time by the file's owner. UserA decides to revoke userB's access permissions for the file. It would be superfluous to take away access to a file the user has already seen, as they may have an extra copy of the file stored locally. Instead, the versions created after this time are made unavailable to the user, the Virtual Secretary informs the server of these changes by sending the message seen in Listing 7.17.

Listing 7.17: Changing file permissions

```
<response id="5" type = "permissions">
    <file id ="000000000001" />
    <userlist>
        <user id="userB">
                <privs>
                    <read>false</read>
                    <write>false</write>
                    <author>false</author>
                    <removed>true</removed>
                </privs>
        </user>
    </userlist>
</response>
```

The FileDB updates userB's permissions for this file in the database, the result of which can be seen in Table 7.17.

UID	FID	permissions	dateTime
userB	000000000001	WR	1234567
userB	000000000001	REM	99876543

Table 7.17: The updated entries for User B in the permissions table

The FileDB sets a flag in the version table next to the latest version of this file to show that the next version of the file will require a new AES key (Table 7.18), and then sends an acknowledgment message back to the Virtual Secretary that originated the request.

FID	dateTime	bitField	permsChanged	version
0000000000001	1234567	111111111111111	False	1
0000000000001	8765432	001100000000000	False	2
0000000000001	9876543	001001000000000	True	3

Table 7.18: The most recent version is flagged in the version table

When userA next checks out the file for editing the Virtual Secretary is informed that a new AES key must be generated to encrypt the edited blocks. In the same way as when the file was originally set, new encrypted AES keys are created for the remaining users with access and distributed accordingly, along with the additional knowledge base entries generated for a new version of a file. The database is updated by new entries in the permissions table (Table 7.19) and the version table (Table 7.20).

UID	FID	permissions	dateTime
userA	0000000000001	WRA	1234567
userB	0000000000001	WR	1234567
userC	0000000000001	R	1234567
userD	0000000000001	R	1234567
userE	0000000000001	R	1234567
userB	0000000000001	REM	9876543
userA	0000000000001	WRA	19234569
userC	0000000000001	R	19234569
userD	0000000000001	R	119234569
userE	0000000000001	R	19234569

Table 7.19: The permissions table after User B has had their permissions changed

FID	dateTime	bitField	permsChanged	version
0000000000001	1234567	111111111111111	False	1
0000000000001	8765432	001100000000000	False	2
0000000000001	9876543	001001000000000	True	3
0000000000001	9876543	110000000000000	False	4

Table 7.20: The version table after the permission alteration

The next time userB tries to access the file, they can only get the versions made prior to when their permissions were removed. The FileDB checks the permissions table within the database, which shows they were revoked at time stamp 99876543, and then identifies the latest version of the file that was created before: the third version in this example (see Table 7.21).

FID	dateTime	bitField	permsChanged	version
0000000000001	9876543	001001000000000	True	3

Table 7.21: The last entry made in the version table before userB's permissions changed

The database provides the appropriate blocks of the file by working backwards through the bitfields as described in previous examples, then the required blocks are sent. If another user wishes to read the latest version of the file, their Virtual Secretary must now use at least 2 keys to decrypt all the blocks. For this reason, currently when a user has permissions for a file removed the next version of the file will have to be completely re-encrypted giving a bitfield full of 1's. This is a less elegant solution than that of using multiple keys for one file, but the least complex one since all the blocks are currently sent to the user per version (as the client machine cannot open file segments). The communications protocol, FileDB program and internal database have been designed with the intention of using a block-based file system on the client, so no further implementation work would be needed to enable a multiple-key mechanism once a virtual file system was in place on the client.

7.4.2.9 Deleting a File

When a user instructs their Virtual Secretary to delete a file, the file information is still held on the server and in the file store. However, it will no longer be accessible unless the author instructs their Secretary to undelete it. Listing 7.18 shows the message which would be sent to the FileDB if userA wished to delete the file seen throughout this example.

Listing 7.18: A file deletion message

```
<request id="10" type="deletion">
    <deletion>
        <file id="000000000001" version="2" />
        <kbentry>KB-deleted-entry-string</kbentry>
    </deletion>
</request>
```

The KB-deleted-entry-string contains the knowledge that would need to be inserted into the knowledge bases of other users who previously had access to this file. The FileDB does not retain any knowledge of which files have been deleted as the knowledge bases and Virtual Secretary will ensure that the file is no longer requested. If when a file is deleted it is still checked out by a user with write access, should they alter the file and give it back to their Secretary the files will be effectively undeleted as new data will be produced.

7.5 Summary

This chapter has illustrated the features found in the system level architecture of GIFS and answered the technical questions set out in §7.1.1. Several alternative designs were analysed for suitability and the communications protocol was specified. The different layers found in the GIFS model were explained and contrasted to the layers of other file systems. The mechanisms required to facilitate the allocation units, versioning system and encryption techniques were also examined.

The examples seen in the second half of the chapter show a subsection of the communications between the Virtual Secretary and the server and also those communications between Virtual Secretaries. It has been demonstrated how the FileDB processes the data contained in the communications messages, returning query results from the database, assembling files to send to Virtual Secretaries or managing the distribution of knowledge and encryption keys. The file store which is controlled by the FileDB holds multiple versions of the files without unnecessary space wastage. Used in coordination with the knowledge-based framework and the Virtual Secretary interface these mechanisms can be used to provide transparent and secure file storage whilst removing all associated burdens from the user.

Chapter 8

Conclusion

8.1 Achievements and Evaluation

Throughout the course of this thesis, an alternative paradigm to the design, implementation and deployment of current file systems has been presented, and the technology contributing to its constituent parts examined.

This work has achieved its objectives as set out in Chapter 1, including the following. We have:

- reviewed the surrounding literature and made a critical analysis of current file system technologies, highlighting the increasing gap between user needs and operating system technologies;
- proposed the concept of a global file system which offers increased functionality over existing file systems, catering for a wide-range of user needs including document placement and organisation, version control, support for collaborative environments, secure storage, and powerful and intelligent search functions through an adaptive interface;
- presented the design of a knowledge-based framework for the global file system and implemented the separate parts to fulfill the highlighted concepts, including the structure of the knowledge bases, the user interface, integration with the operating system, network communication protocols and the file storage system;
- evaluated the system through simulation in order to demonstrate its scalability in various tasks over a long period of service.

8.1.1 Review and Analysis of Current Technologies

Although all the required technologies for a global file system have coexisted for a considerable time, they had never been combined to provide the framework necessary to provide users with functionality that current file systems lack. Chapter 2 presented an in-depth review of file system development over a spectrum of operating systems, and §2.2 gave a critical analysis of several file system architectures. The problems currently faced by file system design and implementation were discussed in §2.3. As the Internet and more generally computer networks have grown to surround much of our everyday interactions with computers and file systems, a brief history of its design and development was documented. In particular, we considered the technologies used within the Internet which are displaying increasing similarity to features provided traditionally by file systems in §2.4.2–§2.4.6. It was shown that as the Internet continues to grow in terms of both popularity and commercial success it is already beginning to integrate into local file systems and searching mechanisms.

To provide a broad understanding of the remaining constituent technologies needed to create a global file system framework, Chapter 3 provided a thorough background survey of several artificial intelligence related topics. The concept of knowledge bases and data mining were introduced along with a review of the field of intelligent agents in §3.2. This background information enabled an evaluation of artificial intelligence applications, in particular those offering services similar to a personal assistant. A review of personal search assistants and those which create personalised interfaces followed. Section 3.3 described the concepts of personal information management, groupware and computer supported cooperative work.

Through the literature studies in these two main respects, we found that there is a huge gap between current operating system technology and the needs of users for handling files in everyday life. In particular,

- current systems do not assist users with the organisation of their files [43, 314, 45];
- current systems are not able to identify and store relationships between similar files [91, 124];
- adding encryption or security mechanisms to files is time-consuming and cumbersome, especially in a collaborative environment [163, 80];
- current systems lack rich knowledge of file metadata and access patterns [79, 305];
- the folder paradigm found in hierarchical systems is no longer sufficient for meeting the organisational and storage needs of users [23, 10, 108, 158, 174, 180, 268].

8.1.2 Concept of a Global File System

The second objective was met in Chapters 4, 5 and 7, where the concept of a global file system was introduced and examined in detail. As many features of the file system are transparent to the user, a system overview was presented in §4.4.1 with the focus on the expected behavior of the system from that perspective. The technical concepts regarding the development of a suitable framework were shown in Chapters 5 and 7, with Chapter 5 concentrating on the knowledge-based aspects such as data mining and agent operation, whilst Chapter 7 looked in detail at the system level design including network structure, communications protocols and encryption. The work found in these two chapters in particular provides the evidence needed to support the first major thesis objective.

The concept we proposed is novel and ambitious, because it differs from the existing operating system and file system technologies by providing a transparent and secure facility for the automatic storage of files. It contrasts with other notions found in the literature, such as conventional digital personal assistants (§3.3.2) of personal information management (§3.3) by removing from the user the burden or knowledge of how and where their files are physically stored. It meets the user's requirements (as examined in Chapter 3) for secure and transparent file storage in a collaborative environment including accomplished searching mechanisms.

GIFS is technologically feasible with the ability to develop its intelligence in an evolutionary manner. The extensible nature of the agents and knowledge framework mean that further processing and knowledge gathering techniques can be incorporated without the need for a complete system or protocol redesign. The design is scalable because the knowledge framework and the architecture of the system on the client and server machines (discussed in §7.2) were constructed to be flexible enough to be implemented on any particular machine, operating system architecture or disk configuration.

8.1.3 Knowledge-Based Framework Design

Chapters 5 and 7 showcased the design and implementation options behind the global file system and Virtual Secretary and satisfied the major thesis objective. The knowledge-based design was examined in Chapter 5, including an in depth explanation of data acquisition from user actions as well as the knowledge base structure and design principles. This implicitly establishes how the amount of knowledge currently gathered and analysed by operating systems is insufficient to support the actions of a global file system, as well as demonstrating how the user is not required to expend any additional input in the training of their Secretary.

Each of Chapters 2,3 and 4 presented examples of commercial software which uses a knowledge-based or data mining/search approach such as Stuff I've Seen [90], iTunes [294], Google Desktop Search [1] and Spotlight [5]. These commercial products strengthen the approach used in this thesis, whilst other sources [79, 20, 305] affirm that file and operating systems are both moving toward a data/knowledge-based architecture.

The agent technology required to extract meaningful information from the data gathered in the knowledge bases was then explained in §5.5, with several different techniques for agent deployment being evaluated. Chapter 5 then provided a more detailed analysis of several agents deployed in the system, demonstrating how the raw data gathered through the system could be processed and analysed to yield useful information.

The penultimate chapter of this thesis provided a discussion of several alternative architectures for the server-side architecture and overall framework. Through a detailed example of the lifecycle of a file, Section 7.4 demonstrated the system level technologies required to support the framework which were presented previously in Chapters 4 and 6 and explained the parts of the framework that are transparent to the user through normal operation. Chapter 7 also included a practical explanation of the versioning, encryption and security mechanisms that are used to facilitate a transparent service, which demonstrates how such a service was technologically possible.

8.1.4 Evaluating System Scalability

The final objective was met in Chapter 6, where we demonstrated the effectiveness of a global file system and Virtual Secretary in assisting the user with the problems of information overload and document management in a secure and collaborative environment. Two case studies were presented to analyse the system performance under different conditions.

The first study in §6.2 concentrated on the growth of the knowledge within the system, and provided experimental results for validating the scalability of a knowledge-based file system over long term usage. As the required test data was not available, §6.2.4.6 documents how we created large amounts of simulation data in order to properly scrutinize the performance, as well as analysing historic data in §6.2.4.3. Several different techniques were used in order to demonstrate the improvements in operational times a Virtual Secretary could provide. It was shown that by deploying intelligent agents to analyse data and produce further information and knowledge in the background, search times for large datasets could be significantly reduced. Without these agents and improved search mechanisms, the search times were shown to be non-scalable, which would outnumber any advantages of deploying the system. The combination of search techniques were proved to follow a constant time, meaning that the operational times were scalable and thus gathering large amounts of information would not cause problems with system performance.

The second case study focused on the scalability of the Virtual Secretary whilst assisting the user in everyday file management tasks. It was shown that the Virtual Secretary helped users by decreasing user-generated errors in file distribution and access lists and significantly reducing the amount of time users would need to spend on file administration tasks. The suggestions made by the Virtual Secretary in this case study show that in comparison to traditional file systems, GIFS conclusively reduces the effort required to manage a user's files. In comparison to a human secretary, this study demonstrated that the Virtual Secretary was scalable when reasoning over file access control lists, both in terms of the time taken and the number of errors generated.

8.2 Future Work

The file system mechanism in GIFS was written for optimal storage when multiple versions of file are created. However, part of the elegance of this solution is lost to the extra network traffic created where the file system on the client machine (e.g. the part responsible for displaying the file) could not display only parts of the file. To make full use of the bandwidth optimisations afforded by the file system a virtual file system able to handle and display partial files could be implemented on the client.

The agents included so far within GIFS assist the Virtual Secretary with a variety of tasks. By the addition of more agents, not only could increasingly complex trends be identified within the user-created data to provide a more accurate service, but a more inclusive solution could be offered. A natural step would be to include emails, calendar entries and so on as files and gather data about them in a similar way to which other files are handled.

The opportunity to gather real data from user interactions with the system would be invaluable when analysing the performance of GIFS. User studies take a large amount of time to organise, but would provide useful, naturally-produced data which over a long period of time would become important for system benchmarking. In addition to a user study of file management behaviours using GIFS, it would also be beneficial to analyse further how well the Virtual Secretary performed when asked to calculate the access control lists of a file or the next actions of a user. The lack of currently available data on file system access patterns or file usage statistics also highlights an area of study that could be used to further this work.

The client-server network mechanism has long been acknowledged as giving poor performance under large loads and causing bottlenecks. It would be desirable to distribute the file data over a large number of different network locations without losing reliability or accessibility. Due to the extra information which is processed by the FileDB and kept along with file data, a more sophisticated dissemination mechanism would be required. Although distributed file systems are already a popular area of research, extra attention in this instance should be paid to the security and ownership issues of distributing files and knowledge bases. Another important area to consider when adding in a distributed file system to the framework would be the duplication and availability of files when one or more nodes of the network storage system became unavailable.

As all communications and data transferred over the network are encrypted in some way, the major security weakness of the system lies in the internal format of the knowledge bases on the client machine. Ideally, the knowledge bases should incorporate an encryption mechanism without compromising system performance, requiring the agents which process the data to be able to decrypt it. The design and implementation of a supporting encrypted knowledge framework would prove an interesting topic of research.

Appendix A

Full Results for Case Study 1

A.1 Results for Brute Force, Best-Case Scenario Data

Dataset	1	2	3	4	5	avg	SD
1	48.156	47.422	47.141	47.266	47.859	47.569	0.380
2	48.047	47.063	47.469	47.828	46.969	47.475	0.418
3	47.281	48.531	47.188	47.766	47.469	47.647	0.484
4	46.703	48.094	48.281	47.969	47.813	47.772	0.556
5	47.938	48.688	47.781	47.938	48.094	48.088	0.316

Table A.1: Brute force results for data over 3 months (seconds)

Dataset	1	2	3	4	5	avg	SD
1	1:26.2	1:26.4	1:27.2	1:28.3	1:27.1	1:27.0	0.744
2	1:29.1	1:26.5	1:26.0	1:28.1	1:28.3	1:27.6	1.163
3	1:27.5	1:27.3	1:27.3	1:27.4	1:27.3	1:27.4	0.055
4	1:25.2	1:25.1	1:24.5	1:26.9	1:26.3	1:25.6	0.861
5	1:27.3	1:27.8	1:28.3	1:28.2	1:28.8	1:28.1	0.494

Table A.2: Brute force results for data over 6 months (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:07.6	2:07.9	2:05.6	2:09.6	2:08.9	2:07.9	1.360
2	2:08.8	2:08.2	2:07.0	2:06.5	2:06.8	2:07.5	0.880
3	2:10.0	2:08.1	2:08.1	2:07.4	2:06.8	2:08.1	1.085
4	2:08.9	2:08.2	2:07.4	2:06.8	2:08.7	2:08.0	0.802
5	2:08.2	2:04.3	2:05.7	2:07.8	2:05.6	2:06.3	1.464

Table A.3: Brute force results for data over 9 months (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:45.0	2:49.0	2:46.2	2:45.9	2:47.5	2:46.7	1.390
2	2:45.8	2:46.6	2:47.7	2:46.7	2:51.2	2:47.6	1.876
3	2:46.1	2:45.5	2:46.7	2:48.2	2:47.6	2:46.8	0.988
4	2:49.2	2:47.8	2:46.7	2:48.0	2:46.0	2:47.5	1.117
5	2:46.4	2:48.2	2:48.6	2:46.8	2:44.8	2:47.0	1.346

Table A.4: Brute force results for data over 1 year (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	13:37	13:48	13:38	13:44	13:40	13:41	4.211
2	13:33	13:34	13:50	13:55	13:43	13:43	8.340
3	13:34	13:44	13:39	13:34	13:44	13:38	4.361
4	13:36	13:33	13:49	13:51	13:43	13:42	6.890
5	13:54	13:47	13:58	13:51	13:44	13:50	4.996

Table A.5: Brute force results for data over 5 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	27:46	28:04	27:45	27:50	27:45	27:50	7.284
2	27:45	27:34	27:38	27:43	27:49	27:41	5.205
3	27:52	27:45	28:13	27:57	27:40	27:53	11.447
4	28:06	27:53	27:30	28:06	28:19	27:58	16.595
5	28:27	28:01	28:10	28:09	28:10	28:11	8.426

Table A.6: Brute force results for data over 10 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	58:18	58:44	58:28	58:04	57:56	58:17	16.925
2	57:29	57:54	58:34	57:18	57:51	57:49	26.013
3	57:55	58:08	57:16	57:12	57:46	57:39	21.902
4	57:15	57:36	57:35	57:40	58:43	57:45	29.980
5	58:33	57:42	57:51	58:44	58:28	58:15	24.392

Table A.7: Brute force results for data over 20 years (minutes:seconds)

Dataset	**1**	**2**	**3**	**4**	**5**	**avg**	**SD**
1	1:38:51	1:37:50	1:40:20	1:40:17	1:43:50	1:40:13	121
2	1:37:04	1:39:25	1:40:02	2:09:03	1:55:30	1:48:12	737
3	1:34:13	1:34:26	1:32:44	1:31:43	1:32:16	1:33:04	64
4	1:44:10	1:42:26	1:37:26	1:38:47	1:37:49	1:40:07	161
5	1:32:39	1:34:36	1:57:51	1:57:35	1:35:03	1:43:32	695

Table A.8: Brute force results for data over 30 years (hours:minutes:seconds)

Dataset	**1**	**2**	**3**	**4**	**5**	**avg**	**SD**
1	2:47:38	2:20:58	2:42:46	2:44:14	2:43:22	2:39:47	573
2	2:39:38	2:44:59	2:57:05	3:14:33	2:41:46	2:51:36	777
3	2:50:48	2:30:26	3:22:53	3:23:55	2:59:24	3:01:29	1212
4	3:16:13	3:53:45	3:50:08	3:57:34	3:33:47	3:42:17	921
5	3:03:41	3:07:56	3:07:18	3:03:27	3:35:50	3:11:38	734

Table A.9: Brute force results for data over 40 years (hours:minutes:seconds)

Dataset	**1**	**2**	**3**	**4**	**5**	**avg**	**SD**
1	4:11:04	4:12:59	3:59:01	4:44:41	3:53:24	4:12:13	1068
2	3:57:14	3:36:40	4:03:30	4:54:02	3:23:44	3:59:01	1857
3	3:46:07	4:57:14	4:50:39	4:44:15	5:47:04	4:49:03	2311
4	3:51:55	5:01:37	4:49:06	4:35:56	5:47:40	4:49:14	2249
5	4:58:35	3:55:22	4:54:36	5:13:34	4:48:37	4:46:08	1601

Table A.10: Brute force results for data over 50 years (hours:minutes:seconds)

A.2 Results for Cached, Best-Case Scenario Data

Dataset	1	2	3	4	5	avg	SD
1	09.5	09.3	09.4	09.1	09.5	09.4	0.149
2	09.8	09.0	09.5	09.9	09.8	09.6	0.328
3	09.5	09.8	09.4	09.2	09.2	09.4	0.222
4	09.3	09.5	09.7	09.8	09.5	09.6	0.174
5	09.1	09.5	09.5	09.6	09.1	09.4	0.215

Table A.11: Caching method results for data over 1 month (seconds)

Dataset	1	2	3	4	5	avg	SD
1	10.1	10.1	10.2	10.0	10.2	10.1	0.074
2	10.0	10.2	10.1	10.3	10.0	10.1	0.116
3	10.1	10.1	10.2	10.0	10.1	10.1	0.063
4	09.7	10.0	10.4	10.2	10.2	10.1	0.236
5	10.0	10.1	10.2	10.2	10.1	10.1	0.074

Table A.12: Caching results for data over 3 months (seconds)

Dataset	1	2	3	4	5	avg	SD
1	10.6	11.1	09.9	11.0	10.2	10.6	0.458
2	10.7	11.4	11.4	11.3	10.2	11.0	0.477
3	11.2	11.2	10.1	10.7	10.7	10.8	0.406
4	10.7	10.5	11.3	11.0	10.7	10.8	0.280
5	10.9	11.2	11.0	10.4	10.8	10.9	0.265

Table A.13: Caching results for data over 6 months (seconds)

Dataset	1	2	3	4	5	avg	SD
1	12.3	12.4	12.4	12.5	11.9	12.3	0.209
2	12.4	12.7	12.1	11.9	12.0	12.2	0.292
3	11.9	12.0	12.3	11.9	12.5	12.1	0.240
4	12.2	12.4	12.5	13.0	12.5	12.5	0.263
5	16.6	16.3	16.6	17.3	17.1	16.8	0.365

Table A.14: Caching results for data over 9 months (seconds)

Dataset	1	2	3	4	5	avg	SD
1	14.2	13.6	13.6	14.0	14.1	13.9	0.252
2	14.8	14.9	14.4	15.4	14.2	14.7	0.417
3	14.5	14.2	14.3	14.1	15.0	14.4	0.318
4	14.5	14.7	13.7	14.1	13.9	14.2	0.370
5	13.8	14.3	14.1	14.3	14.2	14.1	0.185

Table A.15: Caching results for data over 1 year (seconds)

Dataset	1	2	3	4	5	avg	SD
1	51.4	51.4	55.3	53.6	54.5	53.2	1.595
2	53.7	52.8	53.5	52.8	54.2	53.4	0.540
3	54.8	54.7	55.0	53.7	54.4	54.5	0.453
4	54.7	54.9	54.9	55.4	54.6	54.9	0.275
5	53.8	54.7	54.3	55.7	52.8	54.3	0.960

Table A.16: Caching results for data over 5 years (seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:36.7	2:36.7	2:35.8	2:36.7	2:36.4	2:36.5	0.349
2	2:35.8	2:35.2	2:35.1	2:35.5	2:47.5	2:37.8	4.846
3	2:55.7	2:54.8	2:54.4	2:54.4	2:54.3	2:54.7	0.519
4	2:55.4	2:53.6	2:54.6	2:54.1	2:54.1	2:54.4	0.608
5	2:54.8	2:54.7	2:42.9	2:38.3	2:38.9	2:45.9	7.381

Table A.17: Caching results for data over 10 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	8:44.2	9:23.1	9:45.1	9:14.2	8:44.7	9:10.3	23.352
2	8:44.5	8:43.4	8:44.5	8:43.4	8:56.5	8:46.5	5.044
3	9:41.9	9:37.2	8:42.3	8:42.4	8:42.4	9:05.2	28.053
4	8:40.5	9:04.4	9:39.7	9:23.1	8:56.4	9:08.8	20.655
5	8:46.4	8:46.9	8:46.7	8:47.6	8:47.2	8:47.0	0.412

Table A.18: Caching results for data over 20 years(minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	20:03.3	20:03.2	20:02.0	20:01.7	20:02.2	20:02.5	0.649
2	21:52.4	21:33.1	21:21.2	20:55.7	20:02.7	21:09.0	37.909
3	20:14.4	20:17.2	20:16.0	20:15.2	20:15.0	20:15.6	0.966
4	20:51.6	21:21.3	20:00.3	19:59.0	20:00.1	20:26.5	33.978
5	20:04.6	20:03.3	20:04.3	20:03.5	20:03.7	20:03.9	0.491

Table A.19: Caching results for data over 30 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	36:18.1	36:16.6	36:17.3	36:16.8	36:17.3	36:17.2	0.519
2	36:21.7	36:17.6	36:16.5	36:17.2	36:18.3	36:18.3	1.816
3	34:20.7	34:19.1	34:19.9	34:19.7	34:21.8	34:20.2	0.932
4	36:11.6	36:10.7	36:25.0	36:11.6	36:10.3	36:13.8	5.603
5	36:14.5	36:12.8	36:13.1	36:11.0	36:11.5	36:12.6	1.238

Table A.20: Caching results for data over 40 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	50:29.8	50:26.0	50:24.8	50:25.4	50:25.3	50:26.3	1.810
2	50:30.5	50:28.7	50:30.0	50:31.5	50:31.7	50:30.5	1.088
3	50:32.6	50:27.8	50:26.1	50:28.4	50:27.8	50:28.5	2.170
4	50:12.6	50:22.7	50:07.4	50:07.4	50:08.4	50:11.7	5.825
5	50:31.0	50:28.8	50:28.4	52:46.6	52:36.8	51:22.3	64.894

Table A.21: Caching results for data over 50 years (minutes:seconds)

A.3 Results for Archived, Best-Case Scenario Data

Dataset	1	2	3	4	5	avg	SD
1	21.453	21.672	21.391	21.172	21.172	21.372	0.188
2	21.219	21.578	21.281	21.078	21.469	21.325	0.178
3	21.313	21.188	21.359	21.281	21.375	21.303	0.066
4	20.984	21.609	21.156	21.688	21.609	21.409	0.283
5	21.516	21.375	21.422	21.406	21.563	21.456	0.071

Table A.22: Archived results for data over 1 month (seconds)

Dataset	1	2	3	4	5	avg	SD
1	47.688	47.656	46.984	47.375	47.031	47.347	0.298
2	47.328	47.438	47.156	47.500	47.609	47.406	0.154
3	47.156	47.016	47.266	47.281	47.156	47.175	0.095
4	47.359	47.578	47.406	47.734	46.875	47.390	0.289
5	47.406	47.078	46.984	46.969	47.500	47.187	0.222

Table A.23: Archived results for data over 3 months (seconds)

Dataset	1	2	3	4	5	avg	SD
1	1:26.09	1:26.93	1:26.32	1:26.90	1:26.57	1:26.56	0.326
2	1:26.75	1:26.32	1:26.37	1:26.39	1:27.00	1:26.56	0.263
3	1:26.67	1:26.65	1:26.82	1:27.95	1:27.23	1:27.06	0.488
4	1:25.20	1:25.65	1:26.18	1:26.01	1:26.15	1:25.84	0.371
5	1:26.70	1:26.25	1:26.34	1:29.28	1:26.31	1:26.97	1.162

Table A.24: Archived results for data over 6 months (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:06.17	2:05.93	2:06.40	2:06.82	2:05.81	2:06.23	0.360
2	2:03.68	2:03.31	2:03.60	2:03.35	2:03.54	2:03.50	0.144
3	2:03.07	2:04.23	2:03.79	2:03.48	2:03.46	2:03.61	0.385
4	2:02.54	2:02.73	2:02.37	2:03.15	2:02.93	2:02.75	0.276
5	2:01.35	2:04.39	2:02.01	2:02.45	2:01.92	2:02.42	1.041

Table A.25: Archived results for data over 9 months (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:05.76	2:05.12	2:05.87	2:05.32	2:05.10	2:05.44	0.321
2	1:57.65	1:56.76	1:57.71	1:57.84	1:56.42	1:57.28	0.574
3	1:58.23	1:58.59	1:57.18	1:57.46	1:57.10	1:57.71	0.591
4	1:54.71	1:54.81	1:56.26	1:55.29	1:56.07	1:55.43	0.636
5	1:57.43	1:56.28	1:56.43	1:56.43	1:56.34	1:56.58	0.429

Table A.26: Archived results for data over 1 year (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	3:04.93	3:05.12	3:07.89	3:05.93	3:05.23	3:05.82	1.086
2	3:06.76	3:06.54	3:05.67	3:06.04	3:06.54	3:06.31	0.399
3	2:47.40	2:46.70	2:46.65	2:46.03	2:46.31	2:46.62	0.461
4	2:49.79	2:48.75	2:46.89	2:47.82	2:47.81	2:48.21	0.985
5	2:27.17	2:26.28	2:26.68	2:26.68	2:26.96	2:26.76	0.301

Table A.27: Archived results for data over 5 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:30.26	2:32.31	2:30.18	2:30.76	2:29.43	2:30.59	0.958
2	2:37.21	2:39.10	2:37.84	2:37.06	2:38.14	2:37.87	0.732
3	2:31.23	2:30.10	2:31.29	2:29.98	2:32.98	2:31.12	1.079
4	2:47.15	2:47.14	2:47.64	2:46.96	2:48.70	2:47.52	0.631
5	3:14.09	3:15.03	3:14.31	3:16.79	3:14.09	3:14.86	1.025

Table A.28: Archived results for data over 10 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:47.31	2:47.03	2:45.12	2:47.28	2:47.15	2:46.78	0.834
2	3:16.23	3:18.25	3:17.56	3:17.89	3:16.82	3:17.35	0.730
3	3:01.93	3:01.18	3:00.43	3:02.32	2:59.15	3:01.01	1.130
4	2:56.81	2:55.28	2:54.93	2:55.00	2:54.43	2:55.29	0.806
5	2:37.26	2:37.48	2:37.17	2:37.15	2:36.62	2:37.14	0.283

Table A.29: Archived results for data over 20 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:24.98	2:23.84	2:24.96	2:24.17	2:26.90	2:24.97	1.063
2	2:43.18	2:43.82	2:45.25	2:43.17	2:43.57	2:43.80	0.764
3	2:36.92	2:35.53	2:36.75	2:36.26	2:36.39	2:36.37	0.482
4	3:08.60	3:07.59	3:08.03	3:07.95	3:06.93	3:07.82	0.550
5	2:27.32	2:27.06	2:25.78	2:26.98	2:27.40	2:26.91	0.587

Table A.30: Archived results for data over 30 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	3:10.59	3:10.65	3:10.59	3:12.32	3:10.68	3:10.97	0.678
2	2:51.06	2:51.64	2:51.57	2:51.07	2:51.12	2:51.29	0.256
3	2:43.84	2:55.90	2:44.50	2:42.98	2:43.26	2:46.10	4.930
4	2:53.85	2:54.90	2:53.62	2:54.79	2:53.20	2:54.07	0.666
5	2:51.64	2:53.04	2:51.03	2:51.90	2:51.03	2:51.73	0.741

Table A.31: Archived results for data over 40 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	2:32.28	2:32.84	2:34.28	2:33.18	2:32.53	2:33.02	0.697
2	2:37.39	2:37.12	2:38.93	2:37.90	2:37.31	2:37.73	0.654
3	2:51.43	2:52.57	2:51.03	2:51.06	2:52.01	2:51.62	0.594
4	2:16.79	2:16.71	2:16.00	2:17.81	2:18.07	2:17.08	0.762
5	2:35.26	2:35.43	2:35.10	2:36.40	2:35.21	2:35.48	0.471

Table A.32: Archived results for data over 50 years (minutes:seconds)

A.4 Results for Combined, Best-Case Scenario Data

Dataset	1	2	3	4	5	avg	SD
1	9.4	10.0	9.1	9.5	9.5	9.5	0.289
2	9.5	9.4	9.5	9.4	9.6	9.5	0.074
3	9.2	9.3	9.7	9.0	9.5	9.3	0.241
4	9.0	9.6	9.7	9.6	9.4	9.5	0.249
5	9.4	9.5	9.6	9.5	9.8	9.6	0.135

Table A.33: Combined results for data over 1 month(seconds)

Dataset	1	2	3	4	5	avg	SD
1	10.2	10.6	10.2	10.2	10.8	10.4	0.252
2	10.6	10.6	10.7	10.2	10.2	10.5	0.215
3	10.6	09.9	10.2	10.2	10.2	10.2	0.222
4	10.6	10.2	10.2	10.2	10.2	10.3	0.160
5	10.5	10.4	10.5	09.8	09.9	10.2	0.305

Table A.34: Combined results for data over 3 months (seconds)

Dataset	1	2	3	4	5	avg	SD
1	10.3	10.9	10.5	10.8	10.2	10.5	0.272
2	10.9	10.1	10.9	10.6	10.7	10.6	0.293
3	11.2	10.3	10.5	10.8	11.3	10.8	0.386
4	10.6	11.5	10.0	11.3	11.5	11.0	0.591
5	10.3	10.7	10.0	11.3	10.8	10.6	0.444

Table A.35: Combined results for data over 6 months (seconds)

Dataset	1	2	3	4	5	avg	SD
1	12.6	12.3	12.5	12.9	12.5	12.6	0.195
2	12.0	12.4	12.5	11.8	12.5	12.2	0.287
3	12.3	12.7	12.9	11.9	12.5	12.5	0.344
4	11.9	12.0	12.5	11.9	12.3	12.1	0.240
5	15.8	16.0	15.9	15.9	16.0	15.9	0.074

Table A.36: Combined results for data over 9 months (seconds)

Dataset	1	2	3	4	5	avg	SD
1	13.2	12.9	13.2	13.0	13.7	13.2	0.275
2	13.5	13.0	12.7	12.2	12.4	12.8	0.458
3	13.0	13.4	13.1	13.2	13.5	13.2	0.185
4	12.8	13.0	13.1	13.0	13.0	13.0	0.097
5	13.0	13.0	13.1	12.5	13.5	13.0	0.318

Table A.37: Combined results for data over 1 year(seconds)

Dataset	1	2	3	4	5	avg	SD
1	15.5	14.6	14.6	15.2	14.9	15.0	0.349
2	14.6	14.3	14.7	15.1	14.3	14.6	0.296
3	15.3	15.2	14.3	15.0	15.4	15.0	0.392
4	15.4	14.7	15.1	16.2	15.1	15.3	0.501
5	14.2	15.1	14.3	14.3	13.7	14.3	0.448

Table A.38: Combined results for data over 5 years (seconds)

Dataset	1	2	3	4	5	avg	SD
1	15.0	14.7	15.3	15.0	15.4	15.1	0.248
2	15.3	14.8	14.9	15.6	15.6	15.2	0.338
3	15.8	15.3	15.5	15.4	15.5	15.5	0.167
4	15.6	15.0	14.6	14.9	15.5	15.1	0.376
5	16.2	15.5	15.0	15.3	15.7	15.5	0.402

Table A.39: Combined results for data over 10 years (seconds)

Dataset	1	2	3	4	5	avg	SD
1	15.2	15.4	16.0	15.3	15.5	15.5	0.278
2	16.8	16.6	16.7	17.0	16.5	16.7	0.172
3	15.8	15.4	15.4	15.4	15.0	15.4	0.252
4	16.0	15.0	16.0	15.4	15.3	15.5	0.397
5	15.4	14.8	14.9	15.5	15.3	15.2	0.278

Table A.40: Combined results for data over 20 years (seconds)

Dataset	1	2	3	4	5	avg	SD
1	15.1	14.9	15.3	14.7	15.0	15.0	0.200
2	14.7	15.2	15.5	15.7	15.6	15.3	0.361
3	15.4	14.5	14.7	14.9	15.4	15.0	0.365
4	16.1	15.6	16.5	15.7	16.1	16.0	0.322
5	15.4	16.2	16.0	16.2	15.8	15.9	0.299

Table A.41: Combined results for data over 30 years (seconds)

Dataset	1	2	3	4	5	avg	SD
1	15.6	16.0	16.4	15.8	16.6	16.1	0.370
2	15.6	15.2	15.7	16.1	14.9	15.5	0.414
3	15.6	14.9	15.4	15.5	15.3	15.3	0.241
4	15.6	15.4	15.3	16.2	15.3	15.6	0.338
5	15.7	15.4	15.3	15.5	15.5	15.5	0.132

Table A.42: Combined results for data over 40 years (seconds)

Dataset	1	2	3	4	5	avg	SD
1	15.7	15.4	15.5	14.7	16.0	15.5	0.431
2	15.8	16.1	14.9	16.0	14.7	15.5	0.583
3	15.2	15.4	15.3	16.6	14.9	15.5	0.584
4	15.1	14.4	15.2	14.7	15.0	14.9	0.292
5	15.5	15.5	16.0	16.1	14.7	15.6	0.496

Table A.43: Combined results for data over 50 years (seconds)

A.5 Results for Brute Force, Worst-Case Scenario Data

Dataset	1	2	3	4	5	avg	SD
1	15:07.7	15:02.7	14:57.6	15:11.6	15:21.3	15:08.2	9.03
2	14:58.3	14:58.7	15:09.3	15:21.5	15:16.3	15:08.8	10.37
3	14:54.1	15:06.8	14:58.5	15:06.5	15:07.3	15:02.6	6.00
4	15:05.8	14:59.2	14:58.6	14:58.8	15:01.5	15:00.8	3.04
5	15:04.5	15:00.3	15:30.0	15:00.4	15:22.5	15:11.6	13.79

Table A.44: Brute force results for worst-case data over 1 year (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	33:02.5	33:03.7	32:44.1	32:36.8	32:47.6	32:50.9	11.77
2	33:04.5	32:55.2	32:57.1	33:08.4	32:50.3	32:59.1	7.29
3	32:45.8	33:06.8	32:41.6	32:50.5	32:56.9	32:52.3	9.89
4	32:52.8	32:43.2	32:58.2	32:57.4	32:54.1	32:53.2	5.99
5	32:49.8	32:45.5	32:47.0	32:52.7	32:51.8	32:49.4	3.07

Table A.45: Brute force results for worst-case data over 5 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	1:38:42	1:33:39	1:37:39	1:29:01	1:27:58	1:33:24	292.45
2	1:25:08	1:24:58	1:27:48	1:27:46	1:26:58	1:26:32	83.18
3	1:42:45	1:41:27	1:32:50	1:26:08	1:25:30	1:33:44	490.09
4	1:26:21	1:29:15	1:27:53	1:25:50	1:29:00	1:27:40	92.16
5	1:27:21	1:26:53	1:28:19	1:27:16	1:27:13	1:27:24	32.56

Table A.46: Brute force results for worst-case data over 10 years (h:mm:ss)

A.6 Results for Cached, Worst-Case Scenario Data

Dataset	1	2	3	4	5	avg	SD
1	1:19.3	1:16.6	1:16.6	1:16.4	1:16.0	1:17.0	1.32
2	1:17.9	1:17.6	1:16.6	1:16.6	1:16.6	1:17.1	0.64
3	1:16.1	1:16.5	1:16.2	1:16.0	1:16.7	1:16.3	0.29
4	1:15.9	1:10.6	1:11.3	1:09.6	1:10.2	1:11.5	2.53
5	1:10.6	1:10.1	1:10.2	1:09.9	1:10.7	1:10.3	0.34

Table A.47: Cached results for worst-case data over 1 year (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	3:47.3	3:46.2	3:47.0	3:46.7	3:46.0	3:46.7	0.54
2	3:47.2	3:50.7	4:11.7	4:12.7	4:12.4	4:02.9	12.84
3	4:10.8	4:03.5	3:45.5	3:46.6	3:46.7	3:54.6	11.74
4	3:48.1	3:47.8	3:45.9	3:46.2	3:47.1	3:47.0	0.96
5	3:46.4	3:48.1	3:47.2	3:47.5	3:47.0	3:47.3	0.63

Table A.48: Cached results for worst-case data over 5 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	18:48.6	18:49.0	18:48.8	18:49.9	18:49.0	18:49.1	0.50
2	18:49.3	18:50.3	18:50.4	18:50.3	18:49.9	18:50.1	0.46
3	18:49.4	18:50.2	18:48.9	18:49.1	18:49.3	18:49.4	0.5
4	18:49.7	18:48.6	18:49.3	18:48.9	18:47.9	18:48.9	0.69
5	18:49.7	18:49.7	18:50.3	18:49.6	18:49.2	18:49.7	0.39

Table A.49: Cached results for worst-case data over 10 years (minutes:seconds)

A.7 Results for Archived Worst-Case Scenario Data

Dataset	1	2	3	4	5	avg	SD
1	5:43.4	5:41.9	5:40.5	5:45.2	5:42.8	5:42.8	1.75
2	4:54.8	4:55.6	4:55.6	4:56.5	4:54.0	4:55.3	0.94
3	5:40.8	5:40.4	5:36.1	5:36.5	5:38.9	5:38.6	2.17
4	3:55.5	3:56.9	3:58.8	3:58.8	3:58.2	3:57.6	1.43
5	4:00.7	4:03.7	4:04.6	4:00.4	4:00.4	4:02.0	2.03

Table A.50: Archived results for worst-case data over 1 year (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	4:18.2	4:11.3	4:11.3	4:12.8	4:10.7	4:12.9	3.08
2	4:30.5	4:33.7	4:30.5	4:29.8	4:29.7	4:30.8	1.64
3	5:56.3	5:51.5	5:51.3	5:50.2	5:50.7	5:52.0	2.46
4	7:27.2	7:25.3	7:27.4	7:30.0	7:25.4	7:27.0	1.91
5	6:17.0	6:18.3	6:23.8	6:25.4	6:15.6	6:20.0	4.33

Table A.51: Archived results for worst-case data over 5 years (minutes:seconds)

Dataset	1	2	3	4	5	avg	SD
1	6:19.2	6:21.6	6:20.0	6:19.8	6:17.6	6:19.6	1.44
2	4:55.6	4:57.5	4:55.3	5:06.0	4:56.2	4:58.1	4.49
3	4:14.5	4:15.2	4:14.9	4:16.5	4:15.2	4:15.3	0.75
4	5:59.8	5:54.3	5:55.1	5:53.5	5:53.4	5:55.2	2.65
5	4:28.6	4:29.4	4:29.9	4:31.4	4:29.4	4:29.7	1.04

Table A.52: Archived results for worst-case data over 10 years (minutes:seconds)

A.8 Results for Combined, Worst-Case Scenario Data

Dataset	1	2	3	4	5	avg	SD
1	22.328	23.141	22.828	22.531	22.406	22.647	0.273
2	21.578	21.844	21.203	21.641	21.406	21.534	0.198
3	22.672	22.094	23.156	22.406	21.969	22.459	0.388
4	17.422	17.156	18.25	17.797	18.906	17.906	0.566
5	18.188	18.469	17.609	18.266	18.484	18.203	0.290

Table A.53: Combined results for worst-case data over 1 year (seconds)

Dataset	1	2	3	4	5	avg	SD
1	18.719	19.469	19.578	19.578	19.906	19.45	0.359
2	19.297	20.656	19.563	20.125	20.016	19.931	0.429
3	23.563	22.953	23.078	22.391	23.281	23.053	0.356
4	29.125	28.203	28.203	28.094	28.422	28.409	0.340
5	24.094	23.625	24.703	24.016	24.281	24.144	0.321

Table A.54: Combined results for worst-case data over 5 years (seconds)

Dataset	1	2	3	4	5	avg	SD
1	25.813	25.25	25.797	26	25.359	25.644	0.263
2	21.797	21.109	21.219	20.781	21.313	21.244	0.301
3	20.719	20.563	20.328	19.906	19.906	20.284	0.304
4	23.641	24.594	24.156	23.719	24.703	24.163	0.397
5	20.563	20.781	20.453	20.234	20.563	20.519	0.162

Table A.55: Combined results for worst-case data over 10 years (seconds)

A.9 Full Results for Data Collection Study

Event	Read Access	Write Access	Time (Seconds)
3	C	C	0.78
4	C	C	20.53
5	CPF	C	10.95
6	CH	C	8.16
7	CHP	C	66.56
8	CHP	C	20.67
9	CHP	C	15.72
10	CP	CP	88.33
11	CP	CP	52.29
12	CP	CP	18.36
13	CD	CHP	98.18
14	C	C	42.43
15	[HBC]	-	35.86
16	[BC]	C	13.18
17	[BC]+X C	C	37.64
18	[BC]-Y C	C	38.58
19	[EC]	C	30.07
20	-	C	36.69
21	[EC]	C	51.30
22	[FDH]	C	45.89
23	C	C	57.37

Table A.56: Full results of Subject A predicting file access lists

Event	Read Access	Write Access	Time (Seconds)
3	C	C	0.61
4	C	C	5.45
5	CPF	C	11.52
6	CHP	CHP	10.08
7	CH	CH	18.28
8	CH	CH	0.95
9	CHP	CHP	19.70
10	CHP	CHP	28.7
11	CHP	CHP	0.96
12	CHP	CHP	0.75
13	D	CHP	12.37
14	D	CHP	11.92
15	[BC]D	CH	31.54
16	[BC]D	HC	41.13
17	[BC]+X D	CH	29.62
18	[BC]+X-Y	CH	13.44
19	[EC] [BC] HPD	CH	49.68
20	[EC] [BC] CHPD	HP	49.97
21	[BC]HPD	CHP	93.6
22	[FDH][BC] [EC]CHPD	CHP	23.10
23	C	C	22.95

Table A.57: Full results of Subject B predicting file access lists

Event	Read Access	Write Access	Time (Seconds)
3	C	C	12.15
4	C	C	0.80
5	CPF	CPF	0.91
6	CHP	CHP	7.46
7	CHP	CHP	12.33
8	CHP	CHP	6.15
9	CHP	CHP	1.37
10	CHP	CHP	4.94
11	CHP	CHP	2.52
12	CHP	CHP	0.86
13	CHPD	CHP	7.78
14	CHPD	CHP	11.65
15	[HBC]D	CHPD[HBC]	27.54
16	[BC][HBC] CHPD	[BC][HBC] CHP	22.72
17	[BC]+X CHPD [HBC]	CHP [BC]+X [HBC]	84.51
18	CHP [HBC] [BC]-Y+X D	CHP[HBC] [BC]-Y+X	63.20
19	[BC]+X [EC] CHD	[BC]+X [EC] H	67.87
20	[EC] [BC]+X CD	[EC] [BC]+X	47.45
21	[EC][BC]CFXD	[EC][BC]CFX	38.88
22	[FDH][BC] [EC]CFDX	[BC][EC] CFX	48.69
23	C	C	7.02

Table A.58: Full results of Subject C predicting file access lists

Event	Read Access	Write Access	Time (Seconds)
3	C	C	0.89
4	C	C	6.24
5	CPF	CPF	5.62
6	CHP	CHP	2.63
7	CHP	CHP	20.63
8	CHP	CHP	4.34
9	CHP	CHP	4.31
10	CHP	CHP	7.62
11	CHP	CHP	6.79
12	CHP	CHP	5.13
13	CD	C	38.70
14	CD	C	22.43
15	[HBC]C	C	25.74
16	[BC]C	C	8.21
17	[BC]+X C	C	11.68
18	[BC]+X-Y C	C	18.52
19	[EC] [BC]-Y+X C	C	40.8
20	[EC] [BC]+X-Y C	C	46.45
21	[EC][BC]-Y CX	C	29.54
22	[FDH][EC] [EC][MC]CX	C	104.65
23	[BC]-Y+X C	C	31.49

Table A.59: Full results of Subject D predicting file access lists

Event	Read Access	Write Access	Time (Seconds)
3	C	C	3.01
4	C	C	4.11
5	CPF	CPF	6.28
6	CHP	CHP	6.74
7	FP	FP	17.04
8	FP	FP	17.64
9	FP	FP	3.82
10	FP	FP	8.01
11	FP	FP	6.02
12	FP	FP	4.09
13	HPD	HP	10.48
14	HP	HP	8.36
15	[HBC]C	CHP	14.26
16	[BC][HBC]	CHP	13.36
17	HPXD	HPX	19.10
18	HPXD	HPX	23.43
19	[EC]	HP	17.46
20	HP	HP	15.31
21	[BC]HPD	HP	23.90
22	[FDH][EC]	HP	22.88
23	HP	HP	15.49

Table A.60: Full results of Subject E predicting file access lists

Appendix B

Communication Protocol Examples

Listing B.1: Section of XML Schema defining the request element

```
<xsd:element name="request" type="request"/>
<xsd:complexType name="request">
    <xsd:choice>
        <xsd:sequence>
            <xsd:element name="user" type="user"/>
            <xsd:element name="file" type="file"/>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:element name="userlist" type="userlist"/>
            <xsd:element name="file" type="file"/>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:element name="user" type="user"/>
            <xsd:element name="hashstring" type="
               hashedstring"/>
        </xsd:sequence>
        <xsd:element name="userlist" type="userlist"/>
    <xsd:element name="file" type="file"/>
        <xsd:element name="user" type="user"/>
        <xsd:element name="kbentries" type="kbentries"/>
        <xsd:element name="deletion" type="deletion"/>
    </xsd:choice>
        <xsd:attribute name= "id" type="xsd:int"/>
        <xsd:attribute name= "type" type="xsd:string"/>
</xsd:complexType>
```

Listing B.2: XML Schema defining the communications protocol

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="request" type="request"/>
<xsd:element name = "response" type="response"/>

<xsd:complexType name="request">
    <xsd:choice>
        <xsd:sequence>
            <xsd:element name="user" type="user"/>
            <xsd:element name="file" type="file"/>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:element name="userlist" type="userlist"/>
            <xsd:element name="file" type="file"/>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:element name="user" type="user"/>
            <xsd:element name="hashstring" type="
               hashedstring"/>
        </xsd:sequence>
        <xsd:element name="userlist" type="userlist"/>
    <xsd:element name="file" type="file"/>
        <xsd:element name="user" type="user"/>
        <xsd:element name="kbentries" type="kbentries"/>
        <xsd:element name="deletion" type="deletion"/>
    </xsd:choice>
        <xsd:attribute name= "id" type="xsd:int"/>
        <xsd:attribute name= "type" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name = "userlist">
    <xsd:choice>
        <xsd:element name = "user" type = "user" minOccurs="
           0"/>
        </xsd:choice>
</xsd:complexType>

<xsd:complexType name = "user">
    <xsd:choice>
        <xsd:element name = "privs" type="privs"/>
        <xsd:element name = "newkey" type="newkey"/>
        <xsd:element name="publicKey" type="xsd:string"
           minOccurs="0"/>
        <xsd:element name = "online" type="xsd:string"/>
    </xsd:choice>
```

```
        <xsd:attribute name= "id" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name = "privs">
    <xsd:sequence>
        <xsd:element name ="read" type = "xsd:string" />
            <xsd:element name ="write" type = "xsd:string" /
               >
            <xsd:element name ="author" type = "xsd:string"
               />
            <xsd:element name ="removed" type = "xsd:string"
                />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name ="newkey">
        <xsd:choice>
        <xsd:element name = "id" type="xsd:string" />
        </xsd:choice>
</xsd:complexType>

<xsd:complexType name ="hashedstring">
        <xsd:choice>
            <xsd:element name = "id" type="xsd:string" />
        </xsd:choice>
</xsd:complexType>

<xsd:complexType name = "kbentries">
        <xsd:sequence>
            <xsd:element name = "user" type = "user" />
            <xsd:element name = "file" type = "file" />
            <xsd:attribute name = "kb_entry" type = "
               xsd:string"/>
        </xsd:sequence>
</xsd:complexType>


<xsd:complexType name = "deletion">
        <xsd:sequence>
            <xsd:element name = "file" type = "file" />
            <xsd:attribute name = "kb_entry" type = "
               xsd:string"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name = "file">
```

```
    <xsd:choice>
        <xsd:element name = "blocklist" type = "blocklist"
           minOccurs="0"/>
        <xsd:element name = "filedata" type = "xsd:string"
           minOccurs="0"/>
    </xsd:choice>
    <xsd:attribute name = "id" type="xsd:string"/>
    <xsd:attribute name = "version" type="xsd:int" use="
       optional"/>
</xsd:complexType>

<xsd:complexType name = "blocklist">
    <xsd:choice>
        <xsd:element name = "block" type = "block" minOccurs
           ="1" maxOccurs="unbounded"/>
    </xsd:choice>
</xsd:complexType>

<xsd:complexType name = "block">
    <xsd:sequence>
        <xsd:element name="blockdata" type="xsd:string"
           minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="response">
    <xsd:choice>
        <xsd:element name="userlist" type="userlist"/>
        <xsd:element name="file" type="file"/>
        <xsd:element name="valid" type="xsd:string"/>
        <xsd:element name="logon" type="xsd:string"/>
        <xsd:sequence>
            <xsd:element name="file" type="file"/>
            <xsd:element name="userlist" type="userlist"/>
        </xsd:sequence>
        <xsd:element name="string" type="string"/>
    </xsd:choice>
    <xsd:attribute name= "id" type="xsd:int"/>
        <xsd:attribute name= "type" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="string">
    <xsd:attribute name= "id" type="xsd:int"/>
</xsd:complexType>

</xsd:schema>
```

Listing B.3: Example permissions message

```
<response id="5" type = "permissions">
    <file id ="000000000001" />
    <userlist>
        <user id="userA">
                <privs>
                    <read>true</read>
                    <write>true</write>
                    <author>true</author>
                    <removed>false</removed>
                </privs>
        </user>
        <user id="userB">
                <privs>
                    <read>true</read>
                    <write>true</write>
                    <author>false</author>
                    <removed>false</removed>
                </privs>
            </user>
        <user id="userC">
                <privs>
                    <read>true</read>
                    <write>false</write>
                    <author>false</author>
                    <removed>false</removed>
                </privs>
        </user>
        <user id="userD">
                <privs>
                    <read>true</read>
                    <write>false</write>
                    <author>false</author>
                    <removed>false</removed>
                </privs>
        </user>
    </userlist>
</response>
```

Bibliography

[1] Google desktop search. `http://desktop.google.com/features.html`.

[2] Lynx source distribution directory. `http://lynx.isc.org/`.

[3] RDF Vocabulary Description Language 1.0: RDF Schema. `http://www.w3.org/TR/rdf-schema/`, 2004.

[4] Flickr. `http://www.flickr.com/`, 2005.

[5] Spotlight. `http://www.apple.com/macosx/features/spotlight/`, 2005.

[6] Internet world usage stats. `http://www.internetworldstats.com/`, 2006.

[7] Reiser 4 Design Principles: Dancing Tree. `http://www.namesys.com/v4/v4.html#dancing_tree`, 2006.

[8] J. Abbate. *Inventing the Internet.* MIT Press, 1999.

[9] G. D. Abowd and E. D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, 7(1):29–58, 2000.

[10] D. Abrams, R. Baecker, and M. Chignell. Information archiving with bookmarks: Personal web space construction and organization. In *Proceedings CHI 1998*, pages 41–48, 1998.

[11] L. A. Adamic and B. A. Huberman. The web's hidden order. *Communications of the ACM*, 44(9):55–59, Sept. 2001.

[12] M. Ames and M. Naaman. Why we tag: Motivations for annotation in mobile and online media. In *CHI '07: Proceedings of the SIGCHI conference on human factors in computing systems*, pages 971–980, New York, NY, USA, 2007. ACM Press.

[13] N. Anderson. Tim Berners-Lee on Web 2.0: "Nobody even knows what it means". http://arstechnica.com/news.ars/post/20060901-7650.html, Sept. 2006.

[14] E. Andre and T. Rist. From adaptive hypertext to personalized web companions. *Communications of the ACM*, 45(5):43–46, May 2002.

[15] M. Andreessen and E. Bina. NCSA Mosaic: A Global Hypermedia System. *Internet Research: Electronic Networking Applications and Policy*, 4(1):7–17, 1994.

[16] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Albert. The Internet Gopher Protocol (a distributed document search and retrieval protocol), 1993.

[17] Apple Computer, Inc. HFS specification from developer.apple.com. http://developer.apple.com/documentation/mac/Files/Files-99.html, July 1996.

[18] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a distributed file system. In *SOSP '91: Proceedings of the thirteenth ACM symposium on Operating systems principles*, pages 198–212, New York, NY, USA, 1991. ACM Press.

[19] O. Bälter. Strategies for organizing email messages. In *Proceedings of HCI 1997*, pages 21–38, 1997.

[20] X. Bao, J. L. Herlocker, and T. G. Dietterich. Fewer clicks and less frustration: Reducing the cost of reaching the right folder. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 178–185, New York, NY, USA, 2006. ACM Press.

[21] A. Barak, D. Malki, and R. Wheeler. AFS, BFS, CFS... or Distributed File Systems for UNIX. In *European UNIX Users Group Conference Proceedings*, pages 461–472. EUUG, Sept. 1986.

[22] D. Barreau. Context as a factor in personal information management systems. *Journal of the American Society for Information Science*, 46(5):327–339, 1995.

[23] D. Barreau and B. A. Nardi. Finding and reminding: File organization from the desktop. *SIGCHI Bull.*, 27(3):39–43, 1995.

[24] M. Bauer, D. Dengler, and G. Paul. Instructible information agents for web mining. In *IUI 2000, New Orleans, LA, USA*, pages 21–28. ACM, 2000.

[25] M. Bauer, D. Dengler, G. Paul, and M. Meyer. Programming by demonstration for information agents. *Communications of the ACM*, 43(3):98–103, 2000.

[26] N. J. Belkin. Intelligent information retrieval: Whose intelligence? In *Proceedings des 5. Internationalen Sypmosiums far Informationswissenschafi (ISI '96)*, 1996.

[27] N. J. Belkin. Helping people find what they don't know. *Communications of the ACM*, 43(8):58–61, Aug. 2000.

[28] J. G. Bellika, G. Hartvigsen, and R. A. Widding. Using user models in software agents: The virtual secretary. *ICMAS98*, pages 391–392, 1998.

[29] J. G. Bellika, G. Hartvigsen, and R. A. Widding. The virtual library secretary: A user model-based software agent. *Personal Technologies*, (2):162–187, 1998.

[30] V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Innovation in extremis: Evolving an application for the critical work of email and information management. In *Proceedings of DIS'02*, pages 181–192, 2002.

[31] V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Taking email to task: The design and evaluation of a task management centered email tool. In *Proceedings of CHI'03*, pages 345–352, 2003.

[32] J. M. Bennett, M. A. Bauer, and D. Kinchlea. Characteristics of files in NFS environments. In *SIGSMALL '91: Proceedings of the 1991 ACM SIGSMALL/PC symposium on Small systems*, pages 33–40, New York, NY, USA, 1991. ACM Press.

[33] H. Berghel. Cyberspace 2000: Dealing with information overload. *Communications of the ACM*, 40(2):19–24, Feb. 1997.

[34] O. Bergman, R. Beyth-Marom, and R. Nachmias. The user subjective approach to personal information management systems. *Journal for the American Society for Information Science*, 9(54):872–878, 2003.

[35] O. Bergman, R. Boardman, J. Gwizdka, and W. Jones. Personal information management. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1598–1599, New York, NY, USA, 2004. ACM Press.

[36] J. E. P. Bernardo A. Huberman, Peter L. T. Pirolli and R. M. Lukose. Strong Regularities in World Wide Web Surfing. *Science*, 280:95–97, Apr. 1998.

[37] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World Wide Web. *Communications of the ACM*, 37(8):76–82, Aug. 1994.

[38] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American Website: http://www.scientificamerican.com, May 2001.

[39] C.-G. Bian, W. Cao, and G. Hartvigsen. ViSe2 - An Agent-Based Expert Consulting System with Efficient Cooperation. *International Journal of Advanced Computational Intelligence*, 2(3):104–110, 1998.

[40] M. Blaze. A Cryptographic File System for UNIX. In *Proceedings of the First ACM Conference on Communications and Computing Security, Fairfax, VA*, Nov. 1993.

[41] S. Bloehdorn, O. Görlitz, S. Schenk, and M. Völkel. TagFS - Tag Semantics for Hierarchical File Systems. In *Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06), Graz, Austria, September 6-8, 2006*, September 2006.

[42] R. Boardman. *Improving Tool Support for Personal Information Management.* PhD thesis, Imperial College London, Sept. 2004.

[43] R. Boardman and M. A. Sasse. "Stuff Goes into the Computer and Doesn't Come Out": A Cross-tool Study of Personal Information Management. In *CHI 2004, Vienna, Austria*, pages 583–590, Apr. 2004.

[44] R. Boardman, R. Spence, and M. A. Sasse. Too Many Hierarchies? The Daily Struggle for Control of the Workspace. In *Proceedings of HCI International 03*, volume 1, pages 616–620, 2003.

[45] O. Bondarenko and R. Janssen. Documents at hand: Learning from paper to improve digital technologies. In *CHI 2005, Portland, Oregon, USA*, pages 121–130, Apr. 2005.

[46] J. Bonwick. ZFS: The Last Word in Filesystems, A Personal Blog. `http://blogs.sun.com/roller/page/bonwick?entry=zfs_the_last_word_in`, Oct. 2005.

[47] S. Brecher. HFS File Structure Explained. *MacTech*, 1(12).

[48] D. Bridges. Inside the High Performance File System. *Significant Bits magazine*, 1996.

[49] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 265–276, New York, NY, USA, 1997. ACM Press.

[50] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[51] M. Brinkmeier. Pagerank revisited. *ACM Trans. Inter. Tech.*, 6(3):282–301, 2006.

[52] C. H. Brooks and N. Montanez. Improved annotation of the blogosphere via autotagging and hierarchical clustering. In *WWW '06: Proceedings of*

the 15th international conference on World Wide Web, pages 625–632, New York, NY, USA, 2006. ACM Press.

[53] F. Buchholz. The structure of the Reiser file system. `http://homes.cerias.purdue.edu/~florian/reiser/reiserfs.php`, Jan. 2006.

[54] M. Buffa and F. Gandon. SweetWiki: Semantic Web enabled technologies in Wiki. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 69–78, New York, NY, USA, 2006. ACM Press.

[55] P. Buneman, S. Khanna, and W. C. Tan. Data Provenance: Some Basic Issues. In *FSTTCS*, volume 1974 of *Lecture Notes in Computer Science*, pages 87–93. Springer, 2000.

[56] V. Bush. As we may think. *The Atlantic Monthly*, 1(176):101–108, July 1945.

[57] B. Callaghan, B. Pawlowski, and P. Staubach. RFC 1813: NFS version 3 protocol specification, June 1995. See also RFC1094 [269]. Status: INFORMATIONAL.

[58] R. Campbell. *Managing AFS: The Andrew File System.* Prentice Hall, 1998.

[59] S. K. Card, J. D. MacKinlay, and B. Schneiderman. *Readings in Information Visualisation.* Morgan Kaufmann, 1999.

[60] V. G. Cerf, Y. K. Dalal, and C. A. Sunshine. RFC 675: Specification of Internet Transmission Control Program, Dec. 1974. Status: UNKNOWN. Not online.

[61] V. G. Cerf and R. E. Kahn. A protocol for packet network interconnection. *IEEE Transactions on Communcation Technology*, 22(5):627–641, May 1974.

[62] D. Chaffey. *Groupware, Workflow and Intranets: Re-engineering the Enterprise with Collaborative Software.* Digital Press, 1998.

[63] M. Chen and J. Gooch. Global Intelligent File Tele-System (GIFTS): The next generation of World Wide Web. In *The UK First Workshop on Grand Challenges for Computing Research*, 2002.

[64] M. Chen and J. Gooch. Virtual Secretary: A Knowledge-based User Interface for File Management. In *The 8th World Multiconference on Systematics, Cybernetics and Informatics*, 2004.

[65] A. Chin and M. Chignell. A social hypertext model for finding community in blogs. In *HYPERTEXT '06: Proceedings of the seventeenth conference on hypertext and hypermedia*, pages 11–22, New York, NY, USA, 2006. ACM Press.

[66] J. M. Chirico. Seagate outlines the future of storage. `http://www.hardwarezone.com/articles/cat.php?id=30`, Jan. 2006.

[67] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[68] P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. pages 197–204, 1998.

[69] B. Collins-Sussman. The subversion project: building a better CVS. *Linux J.*, 2002(94):3, 2002.

[70] A. Cooper. *About Face 2.0: The Essentials of User Interface Design.* John-Wiley & Sons, Inc., 2003.

[71] B. Coppin. *Artificial Intelligence Illuminated.* Jones and Bartlett Publishers International, 2004.

[72] P. Corbett, S. J. Baylor, and D. G. Feitelson. The Vesta Parallel File System. IBM Research Report RC 18337, Yorktown Hts, NY, Sept. 1992.

[73] P. F. Corbett and D. G. Feitelson. The Vesta parallel file system. *ACM Trans. Comput. Syst.*, 14(3):225–264, 1996.

[74] G. Coulouris. Say hello to the reactive computer. Technical Report 1976, Queen Mary College, Department of Computer Science, Apr. 1976.

[75] I. Crabtree, S. J. Soltysiak, and M. P. Thint. Adaptive personal agents. *Personal Technologies*, (2):141–151, 1998.

[76] H. Custer. *Inside the Windows NT File System.* Microsoft Press, first edition, 1994.

[77] E. Cutrell and S. T. Dumais. Exploring personal information. *Communications of the ACM*, 49(4):50–51, Apr. 2006.

[78] E. Cutrell, S. T. Dumais, and J. Teevan. Searching to eliminate personal information management. *Communications of the ACM*, 49(1):58–64, Jan. 2006.

[79] E. Cutrell, D. Robbins, S. Dumais, and R. Sarin. Fast, flexible filtering with Phlat. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 261–270. ACM Press, 2006.

[80] M. Czerwinski and E. Horvitz. An investigation of memory for daily computing events. In *Proceedings of HCI 2002*, pages 230–245, 2002.

[81] J. Daemen and V. Rijmen. AES Proposal: Rijndael, Sept. 1999.

[82] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. C. A. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, 4(5):63–74, 2000.

[83] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification, December 1998.

[84] A. R. Dennis, J. Alan R. Heminger, J. J. R. Nunamaker, and D. R. Vogel. Bringing automated support to large groups: The Burr-Brown experience. *Inf. Manage.*, 18(3):111–121, 1990.

[85] M. Devarakonda. Impact of application scale and diversity on file systems. In *SIGOPS European Workshop '94*, pages 123–124, Apr. 1994.

[86] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the Semantic Web. *The VLDB Journal*, 12(4):303–319, 2003.

[87] J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. *SIGMETRICS Perform. Eval. Rev.*, 27(1):59–70, 1999.

[88] P. Dourish, W. K. Edwards, A. LaMarca, J. Lamping, K. Petersen, M. Salisbury, D. Terry, and J. Thornton. Extending document management systems with user-specific active properties. *ACM Transactions on Information Systems*, 2(18):140–170, 2000.

[89] P. Dourish, W. K. Edwards, A. LaMarca, and M. Salisbury. Presto: An experimental architecture for fluid interactive document spaces. *ACM Transactions on Computer-Human Interaction*, 2(6):133–161, 1999.

[90] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff I've Seen: A system for personal information retrieval and re-use. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 72–79, New York, NY, USA, 2003. ACM Press.

[91] S. Dumais, E. Cutrell, R. Sarin, and E. Horvitz. Implicit queries (IQ) for contextualized search. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 594–594, New York, NY, USA, 2004. ACM Press.

[92] R. Duncan. Design goals and implementation of the new High Performance File System. *Microsoft Systems Journal*, 4(5):1–13, Sept. 2003.

[93] P. Dyson. *Dictionary of Networking*. Sybex Inc.,U.S., 1999.

[94] A. Einstein. (1879 - 1955).

[95] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: Some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

[96] C. A. Ellis and G. J. Nutt. Office information systems and computer science. *ACM Computing Surveys (CSUR)*, 12(1):27–60, 1980.

[97] C. S. Ellis and R. A. Floyd. The ROE File System. In *Proceedings of the 3rd Symposium on Reliability in Distributed Software and Database Systems*. IEEE, Oct. 1983.

[98] A. Emtage and P. Deutsch. Archie - An Electronic Directory Service for the Internet. In *Winter Usenix Conference Proceedings 1992*, pages 93–110, 1992.

[99] O. Etzioni. The World Wide Web: Quagmire or Gold Mine? *Communications of the ACM*, 39(11):65–68, Nov. 1996.

[100] S. Farrell, V. Buchmann, C. S. Campbell, and P. P. Maglio. Information programming for personal user interfaces. In *IUI '02, San Francisco, California, USA*, pages 190–191. ACM, Jan. 2002.

[101] U. Fayyad and R. Uthurusamy. Data mining and knowledge discovery in databases. *Communications of the ACM*, 39(11):24–26, Nov. 1996.

[102] L. Feng, E. Chang, and T. Dillon. A semantic network-based design methodology for XML documents. *ACM Trans. Inf. Syst.*, 20(4):390–421, 2002.

[103] D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, chapter 1. The MIT Press, 2003.

[104] S. Fertig, E. Freeman, and D. Gelernter. Lifestreams: An alternative to the desktop metaphor. In *CHI '96: Conference companion on Human factors in computing systems*, pages 410–411, New York, NY, USA, 1996. ACM Press.

[105] L. Findlater and J. McGrenere. A comparison of static, adaptive and adaptable menus. In *CHI 2004, Vienna, Austria*, pages 89–96, Apr. 2004.

[106] T. Finin, L. Kagal, and D. Olmedilla. Report on the Models of Trust for the Web workshop (MTW'06). *SIGMOD Rec.*, 35(4):54–56, 2006.

[107] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, 13(3):213–228, 1992.

[108] G. W. Furnas and S. Jul. Workshop on navigation in electronic worlds. In *CHI '97: CHI '97 extended abstracts on Human factors in computing systems*, pages 230–230, New York, NY, USA, 1997. ACM Press.

[109] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. MyLifeBits: Fulfilling the Memex Vision. In *Proceedings Multimedia '02, Juan-les-Pins, France*, pages 235–238. ACM, Dec. 2002.

[110] C. Ghaoui. *Encyclopedia of Human Computer Interaction.* IGI Publishing, 2006.

[111] D. Giampolo. *Practical File System Design with the Be File System.* Morgan Kaufmann, 1998.

[112] D. Gifford, P. Jouvelot, M. Sheldon, and J. O'Toole. Semantic file systems. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles (Pacific Grove, CA).* ACM, 1991.

[113] D. S. Gill, S. Zhou, and H. S. Sandhu. A case study of file system workload in a large-scale distributed environment. *SIGMETRICS Perform. Eval. Rev.*, 22(1):276–277, 1994.

[114] J. Goecks and J. Shavlik. Learning users' interests by unobtrusively observing their normal behaviour. In *IUI 2000, New Orleans, LA, USA*, pages 129–132. ACM, 2000.

[115] J. Gooch. *The JoFS.* Undergraduate dissertation, University of Wales, Swansea, May 2002.

[116] B. Gopal and U. Manber. Integrating content-based access mechanisms with hierarchical file systems. In *Proceedings of the 3rd ACM Symposium on Operating Systems Principles (New Orleans, LA).* ACM, Feb. 1999.

[117] P. Graham. Web 2.0. `http://www.paulgraham.com/web20.html`, Nov. 2005.

[118] B. C. Grau. A possible simplification of the Semantic Web architecture. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 704–713, New York, NY, USA, 2004. ACM Press.

[119] R. Green and S. Pant. Multiagent Data Collection in Lycos. *Communications of the ACM*, 42(3):70, Mar. 1999.

[120] S. Grimaldi. The WinFS Files: Divide et Impera. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwinfs/html/winfs20051206.asp`, Dec. 2005.

[121] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, and L. Moreau. An Architecture for Provenance Systems. Technical report, Electronics and Computer Science, University of Southampton, oct 2006.

[122] J. Grudin. Computer-supported cooperative work: Its history and participation. *IEEE Computer*, 5(27):19–26, 1994.

[123] J. Grudin. Groupware and social dynamics: Eight challenges for developers. *Communications of the ACM*, 37(1):92–105, 1994.

[124] R. Guha, R. McCool, and E. Miller. Semantic search. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 700–709, New York, NY, USA, 2003. ACM Press.

[125] V. N. Guivada, V. V. Raghavan, W. I. Grosky, and R. Kasanagottu. Text databases and information retrieval. *IEEE Internet Computing*, Sept. 1997.

[126] S. Haag, M. Cummings, and D. J. McCubbrey. *Management Information Systems for the Information Age.* McGraw-Hill/Irwin, 2006.

[127] T. Hammond, T. Hannay, B. Lund, and J. Scott. Social Bookmarking Tools: A General Review. *D-Lib Magazine*, 11(4), Apr. 2005.

[128] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining.* MIT Press, Cambridge, MA, 2001.

[129] F. Heart, A. McKenzie, J. McQuillian, and D. Walden. ARPANET Completion Report, Jan. 1978. Bolt, Beranek and Newman, Burlington, MA.

[130] V. Henson, J. Bonwick, and M. Ahrens. Existential QoS for Storage. In *Proceedings of the First Workshop on Algorithms and Architectures for Self-Managing Systems*, June 2003.

[131] C. K. Hess and R. H. Campbell. An application of a context-aware file system. *Personal Ubiquitous Computing*, (7):339–352, Nov. 2003.

[132] S. Hiltz and M. Turoff. *The Network Nation: Human Communication Via Computer.* Addison-Wesley Publishing, 1981.

[133] L. L. Hinchey and D. L. Mills. Magnetic properties of superlattices formed from ferromagnetic and antiferromagnetic materials. *Physical Review B*, 33(5):3329, Mar. 1986.

[134] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumière Project: Bayesian User Modeling for Inferring the Goal and Needs of Software Users. In *Proceedings of the fourteenth Conference in Uncertainty in Artificial Intelligence, Madison, WI*, pages 256–265. Morgan Kaufmann Publishers, July 1998.

[135] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transaction on Computer Systems*, 6(1):51–81, Feb. 1988.

[136] M. A. Hoyle and C. Lueg. Open sesame!: A look at personal assistants. In *PAAM '97, London, UK*, pages 51–60, Apr. 1997.

[137] Internet Society (ISOC). A Brief History of the Internet Version 3.31, 4th August 2000. http://www.isoc.org/internet/history/brief.html (13th February 2002).

[138] T. Isakowitz, M. Bieber, and F. Vitali. Web information systems. *Communications of the ACM*, 41(7):78–80, July 1998.

[139] Ivan Herman. W3C Semantic Web Activity. `http://www.w3.org/2001/sw/`, 2005.

[140] B. J. Jansen. Using temporal patterns of interactions to design effective automated searching assistance. *Communications of the ACM*, 49(4):72–74, Apr. 2006.

[141] S. R. Jones and P. J. Thomas. Empirical assessment of individuals' 'personal information management systems'. *Behaviour and Information Technology*, 16(3):158–160, 1997.

[142] W. Jones. Personal information management. *Annual Review of Information Science and Technology*, 9, 2007.

[143] W. Jones, H. Bruce, A. Foxley, and C. F. Munat. The universal labeller: Plan the project and let your information follow. In *Proceedings of ASIST 2005*, Nov. 2005.

[144] W. Jones, S. Dumais, and H. Bruce. Once found, what next? A Study of 'keeping' behaviours in the personal use of web information. In *Proceedings of ASIST 2002*, pages 391–402, 2002.

[145] W. Jones, A. J. Phuwanartnurak, R. Gill, and H. Bruce. Don't Take My Folders Away! Organizing Personal Information to Get Things Done. In *CHI 2005, Portland, Oregon, USA*, pages 1505–1508, Apr. 2005.

[146] W. P. Jones and T. Dumais. The spatial metaphor for user interfaces: Experimental tests of reference by location versus name. *ACM Transactions on Office Information Systems*, 4(1):42–63, Jan. 1986.

[147] K. Nellis. Experts: Information onslaught bad for your health. `http://www.cnn.com/TECH/9704/15/info.overload/index.html`, 1997.

[148] V. Kaptelinin. UMEA: Translating interaction histories into project contexts. In *Proceedings of SIGCHI 2003*, pages 353–360, 2003.

[149] H. A. Kautz, B. Selman, and M. Coen. Bottom-up design of software agents. *Communications of the ACM*, 7(37):143–146, 1994.

[150] H. A. Kautz, B. Selman, M. Coen, and S. Ketchpal. An experiment in the design of software agents. pages 43–48, 1994.

[151] A. Kay. *Computer Software*, volume 251. 1984.

[152] J. Kaye, J. Vertesi, S. Avery, A. Dafoe, S. David, L. Onaga, I. Rosero, and T. Pinch. How do people manage their digital photographs? In *Proceedings CHI 2006*, pages 275–284, Apr. 2006.

[153] D. Kelly. Evaluating personal information management behaviours and tools. *Communications of the ACM*, 49(1):84–86, Feb. 2006.

[154] S. Khoshafian and M. Buckiewicz. *Introduction to Groupware, Workflow and Workgroup Computing*. John Wiley & Sons Inc, 1995.

[155] A. Kidd. The marks are on the knowledge worker. In *CHI '94: Conference companion on Human factors in computing systems*, page 212, New York, NY, USA, 1994. ACM Press.

[156] D. Kirk, A. Sellen, C. Rother, and K. Wood. Understanding photowork. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 761–770, New York, NY, USA, 2006. ACM Press.

[157] W. Knight. 'Info-mania' dents iq more than marijuana, Apr. 2005. New Scientist Online.

[158] A. J. Ko, H. Aung, and B. A. Myers. 'Eliciting design requirements for maintenance-oriented IDEs: A detailed study of corrective and perfective maintenance tasks. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 126–135, 2005.

[159] H. F. Korth and A. Silberschatz. Database research faces the information explosion. *Communications of the ACM*, 40(2):139–142, Feb. 1997.

[160] R. Krait, T. Mukhopadhyay, J. Szczypula, S. Kiesler, and W. Scherlis. Communication and information: Alternative uses of the internet in households. In *CHI 98, Los Angeles, CA*, pages 368–375, Apr. 1998.

[161] G.-S. Kuo and J.-P. Lin. Design Concepts for an Intelligent Internet. *Communications of the ACM*, 41(11):93–98, Nov. 1998.

[162] M. LaMonica. Google's secret of success? Dealing with failure. `http://news.zdnet.com/2100-9588_22-5596811.html`, Mar. 2005.

[163] M. Lansdale. The psychology of personal information management. *Applied ergonomics*, 1(19):55–66, 1988.

[164] P. J. Leach, P. H. Levine, J. A. Hamilton, and B. L. Stumpf. The file system of an integrated local network. In *CSC '85: Proceedings of the 1985 ACM thirteenth annual conference on Computer Science*, pages 309–324, New York, NY, USA, 1985. ACM Press.

[165] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. S. Wolff. The Past and Future history of the Internet. *Communications of the ACM*, 40(2):102–108, Feb. 1997.

[166] D. B. Lenat. CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38(11):32–38, Nov. 1995.

[167] H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. MIT Press, 2001.

[168] E. Levy and A. Silberschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys (CSUR)*, 22(4):321–374, Dec. 1990.

[169] D. Li, Z. Wang, and R. R. Muntz. "Got COCA?" A new perspective in building electronic meeting systems. In *WACC '99: Proceedings of the international joint conference on Work activities coordination and collaboration*, pages 89–98, New York, NY, USA, 1999. ACM Press.

[170] S. Lilley, G. Lightfoot, and P. Amaral. *Representing Organization: Knowledge, Management, and the Information Age*, chapter 8. Oxford University Press, 2004.

[171] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *Internet Computing*, 7(1):76–80, Jan. 2003.

[172] J. Liu, C. K. Wong, and K. K. Hui. An adaptive user interface based on personalized learning. *IEEE Intelligent Systems*, pages 52–57, Mar. 2003.

[173] Lucent Technologies. Fossil manual page. `http://plan9.bell-labs.com/magic/man2html/4/fossil`, 2006.

[174] P. Lyman and H. Varian. How much information? *The Journal of Electronic Publishing*, 6(2), 2000.

[175] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, July 1994.

[176] P. Maes and A. Wexelblat. Interface agents. In *CHI 96*, pages 369–370, Apr. 1996.

[177] T. W. Malone. How do people organise their desks? implications for the design of office information systems. *ACM Transactions on Office Information Systems*, 1(1):99–112, Jan. 1983.

[178] G. Marchionini. *Information Seeking in Electronic Environments*. Cambridge University Press, 1995.

[179] C. Marlow, M. Naaman, D. Boyd, and M. Davis. HT06, tagging paper, taxonomy, Flickr, academic article, to read. In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and Hypermedia*, pages 31–40, New York, NY, USA, 2006. ACM Press.

[180] G. Marsden and D. E. Cairns. Improving the usability of the hierarchical file system. In *Proceedings of SAICSIT 2003*, pages 122–129, Jan. 2003.

[181] P. Martin and P. W. Eklund. Knowledge Retrieval and the World Wide Web. *IEEE Intelligent Systems*, pages 18–25, May 2000.

[182] J. Matthews, D. Roselli, A. Costello, R. Wang, and T. Anderson. Improving the performance of log-structured file systems with adaptive methods. In *Proceedings of the Sixteenth ACM SOSP*, Oct. 1997.

[183] M. Maybury. Intelligent user interfaces: An introduction. In *IUI 99, Redonodo Beach, CA, USA*, pages 3–4. ACM, 1999.

[184] M. Mayer. *The telephone and the uses of time*, pages 225–245. MIT Press, Cambridge, Massachusetts, 1977.

[185] S. McCarthy, M. Leis, and S. Byan. Larger disk blocks or not. In *Proceedings of the USENIX FAST Conference, Monteray, CA*, Jan. 2002.

[186] J. McGrenere, R. M. Baecker, and K. S. Booth. An evaluation of a multiple interface design solution for bloated software. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 164–170. ACM Press, 2002.

[187] D. P. McKay, T. W. Finin, and A. O'Hare. The intelligent database interface: Integrating AI and database systems. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 677–684, Boston, MA, USA, 29– 3 1990. AAAI Press.

[188] M. K. McKusick. Running "fsck" in the background. In *Proceedings of the BSDCon 2002*, pages 55–64, 2002.

[189] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *Local Filesystems, The Design and Implementation of the 4.4BSD Operating System.* Addison Wesley, 1996.

[190] M. K. McKusick and G. R. Ganger. Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast File System. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, pages 1–18, 1999.

[191] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, Aug. 1984.

[192] Microsoft Corporation. How NTFS Works. `http://technet2.microsoft.com/WindowsServer/en/Library/8cc5891d-bf8e-4164-862d-dac5418c59481033.mspx?mfr=true`, Mar. 2003.

[193] Microsoft Corporation. Limitations of the FAT32 File System: 184006. `http://support.microsoft.com/kb/184006/en-us`, Dec. 2004.

[194] D. E. Millard and M. Ross. Web 2.0: Hypertext by any other name? In *HYPERTEXT '06: Proceedings of the seventeenth conference on hypertext and hypermedia*, pages 27–30, New York, NY, USA, 2006. ACM Press.

[195] M. Minsky and D. Riecken. A conversation with Marvin Minsky about agents. *Communications of the ACM*, 37(7):22–25, July 1994.

[196] P. V. Mockapetris. RFC 882: Domain names: Concepts and facilities, Nov. 1983. Obsoleted by RFC1034, RFC1035. Updated by RFC0973. Status: UNKNOWN.

[197] P. V. Mockapetris. RFC 883: Domain names: Implementation specification, Nov. 1983. Obsoleted by RFC1034, RFC1035. Updated by RFC0973. Status: UNKNOWN.

[198] M. Montebello. Information overload–an IR problem? *Spire*, 00:00–65, 1998.

[199] J. H. Morris, M. Satyanarayanan, M. A. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith. Andrew: A distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, Mar. 1986.

[200] S. J. Mullender and A. S. Tanenbaum. Immediate files. *Software - Practice and Experience*, 14(4):365–368, 1984.

[201] M. D. Mulvenna, S. S. Anand, and A. G. Buchner. Personalization on the net using web mining. *Communications of the ACM*, 43(8):122–125, Aug. 2000.

[202] N. Negroponte. *The Architecture Machine*. The MIT Press, Jan. 1973.

[203] M. N. Nelson, Y. A. Khalidi, and P. W. Madany. The Spring File System. Sun Microsystems Laboratories Inc. Technical Report SMLI TR-93-10, Feb. 1993.

[204] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the Sprite Network File System. *ACM Transaction on Computer Systems*, 6(1):134–154, Feb. 1988.

[205] D. Ng'ambi. Dynamic "intelligent handler" of frequently asked questions. In *IUI '02, San Francisco, California, USA*, pages 190–191. ACM, Jan. 2002.

[206] J. Nielsen. *Hypertext and Hypermedia*. Academic Press, 1990.

[207] J. Nielsen. The death of file systems. `http://www.useit.com/papers/filedeath.html`, Feb. 1996.

[208] J. Nielsen. Curmudgeon: IM, not IP (information pollution). *Queue*, 1(8):76–75, 2003.

[209] P. Norton. *Peter Norton's New Inside the PC*, page 428. Sams Publishing, first edition, 2002.

[210] C. O'Dell and C. J. G. Jr. *If Only We Knew What We Know: The transfer of internal knowledge and best practice*, chapter 1. Free Press, 1998.

[211] D. E. O'Leary. The Internet, Intranets, and the AI Renaissance. *IEEE Computer*, pages 71–78, Jan. 1997.

[212] C. H. olscher and G. Strube. Web Search Behaviour of Internet Experts and Newbies. In *9th International WWW Conference, Amsterdam, NL*, 2000.

[213] A. Orlowski. Windows on a database sliced and diced by BeOS vets. `http://www.theregister.co.uk/2002/03/29/windows_on_a_database_sliced/`, Mar. 2002.

[214] K. Panton, C. Matuszek, D. Lenat, D. Schneider, M. Witbrock, N. Siegel, and B. Shepard. Common Sense Reasoning From CYC to Intelligent Assistant. In I. Y. Cai and J. Abascal, editors, *FSTTCS*, volume 3864 of *Lecture Notes in Artificial Intelligence*, pages 1–31. Springer, 2006.

[215] P. F. Patel-Schneider and D. Fensel. Layering the Semantic Web: Problems and Directions. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 16–29, London, UK, 2002. Springer-Verlag.

[216] D. Pinelle and C. Gutwin. Groupware walkthrough: Adding context to groupware usability evaluation. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 455–462, New York, NY, USA, 2002. ACM Press.

[217] D. Pinelle, C. Gutwin, and S. Greenberg. Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration. *ACM Trans. Comput.-Hum. Interact.*, 10(4):281–311, 2003.

[218] J. Pitkow, H. Schutze, T. Cass, R. Cooley, D. Turnbull, A. Edmonds, E. Adar, and T. Breuel. Personalized search. *Communications of the ACM*, 45(9):50–55, Sept. 2002.

[219] B. Preneel, R. Govaerts, and J. Vandewalle. Differential cryptanalysis of hash functions based on block ciphers. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, pages 183–188, New York, NY, USA, 1993. ACM Press.

[220] K. Preslan, A. Barry, J. Brassow, G. Erickson, E. Nygaard, C. Sabol, S. Soltis, D. Teigland, and M. O'Keefe. A 64-bit, shared disk file system for Linux. In *16th IEEE Symposium on Mass Storage Systems, San Diego, CA*, pages 22–41, Mar. 1999.

[221] D. Quan, K. Bakshi, D. Huynh, and D. R. Karger. User interfaces for supporting multiple categorization. In *Proceedings of Interact 2003*, pages 228–235, 2003.

[222] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *First USENIX Conference on File and Storage Technologies, Monterey, California*, 2002.

[223] S. Quinlan, J. McKie, and R. Cox. Fossil, an archival file server. Lucent Technologies Bell Labs, Unpublished memorandum, Sept. 2003.

[224] R. Rada, C. Cargill, and J. Klensin. Consensus and the web. *Commun. ACM*, 41(7):17–22, 1998.

[225] R. Rao, S. K. Card, W. Johnson, L. Klotz, and R. H. Trigg. Protofoil: Storing and finding the information worker's paper documents in an electronic file cabinet. In *CHI 94, Boston, USA*, pages 180–185, Apr. 1994.

[226] Red Hat, Inc. Red Hat Global File System. `http://www.redhat.com/software/rha/gfs/`, 2004.

[227] G. L. Rein, D. L. McCue, and J. A. Slein. A case for document management functions on the web. *Communications of the ACM*, 40(9):81–89, Sept. 1997.

[228] H. Reiser. The naming system venture. Jan. 2001.

[229] J. Rekimonto. Time-machine computing: A time-centric approach for the information environment. In *Proceedings UIST '99, Asheville, NC*, pages 45–54. ACM Press, Nov. 1999.

[230] R. E. Rice and G. Love. Electronic emotion: Socioemotional content in a computer-mediated communication network. *Communication Research*, 14(1):85–108, 1987.

[231] D. Riehle. How and why Wikipedia works: An interview with Angela Beesley, Elisabeth Bauer, and Kizu Naoko. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 3–8, New York, NY, USA, 2006. ACM Press.

[232] R. Rigby. Warning: Interruption overload. `http://www.ft.com/cms/s/d0f71fb6-3243-11db-ab06-0000779e2340,dwp_uuid=4e612cca-6707-11da-a650-0000779e2340,print=yes.html`, Aug. 2006. The Financial Times Limited.

[233] L. C. Rivero, J. H. Doorn, and V. Ferraggine. *Encyclopedia of Database Technologies and Applications*. IGI Publishing, 2005.

[234] T. Rizzo. WinFS 101: Introducing the New Windows File System. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwinfs/html/winfs03112004.asp`, Mar. 2004.

[235] T. Rizzo and S. Grimaldi. An Introduction to "WinFS" OPath. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwinfs/html/winfs10182004.asp`, Oct. 2004.

[236] D. Robbins. Common threads: Advanced filesystem implementor's guide, part 9: Introducing xfs. `http://www-128.ibm.com/developerworks/linux/library/l-fs9.html`.

[237] L. Roberts and T. Merrill. Toward a cooperative network of time-shared computers. In *Proceedings of the Fall AFIPS Conference*, Oct. 1966.

[238] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich. Data mountain: Using spacial memory for document management. In *Proceedings UIST '98, San Francisco, CA*, pages 153–162. ACM Press, 1998.

[239] K. Rodden and K. Wood. How do people manage their digital photographs? In *Proceedings CHI 2003*, pages 409–416, 2003.

[240] D. Rose, R. Mander, T. Oren, D. Poncéleon, B. Salomon, and Y. Y. Won. Content awareness in a file system interface: implementing the "pile" metaphor for organizing information. In *Proceedings of SIGIR '93*, pages 260–269, 1993.

[241] M. Rosenblum and J. K. Ousterhout. The LFS Storage Manager. In *Proceedings of the 1990 Summer Usenix*, pages 315–324, June 1990.

[242] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured filesystem. *ACM Transactions on Computer Systems*, 10(1):26–52, Feb. 1992.

[243] T. Russell. Cloudalicious: Folksonomy over time. In *JCDL '06: Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, pages 364–364, New York, NY, USA, 2006. ACM Press.

[244] M. Russinovich. NT Internals: Inside Win2K NTFS, Part 1. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnw2kmag00/html/NTFSPart1.asp`, 2002.

[245] J. D. Ruvini and J. M. Gabriel. Do Users Tolerate Errors From Their Assistant? Experiments with an E-mail Classifier. In *IUI '02, San Francisco, California, USA*, pages 216–217. ACM, Jan. 2002.

[246] R. Sandberg, D. Golgberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun network filesystem. pages 379–390, 1988.

[247] M. Saptharishi, C. Diehl, K. Bhat, J. Dolan, and P. Khosla. Cyberscout: Distributed agents for autonomous reconnaissance and surveillance. In *Mechatronics and Machine Vision 2000*, pages 93–100, September 2000.

[248] M. Satyanarayanan. A study of file sizes and functional lifetimes. In *SOSP '81: Proceedings of the eighth ACM symposium on Operating systems principles*, pages 96–108, New York, NY, USA, 1981. ACM Press.

[249] M. Satyanarayanan, J. H. Howard, D. N. Nichols, R. N. Sidebotham, A. Z. Spector, and M. J. West. The ITC Distributed File System: Principles and Design. In *Proceedings of the 10th ACM Syposium on Operating Systems Principles*, pages 35–50, Dec. 1985.

[250] M. Satyanarayanan and M. Spasojevic. AFS and the web: Competitors or Collaborators? In *SIGOPS European Workshop '96*, pages 89–94, 1996.

[251] D. Schaffer and S. Greenberg. Sifting through hierarchical information. In *Proceedings CHI 1993*, pages 173–174, 1993.

[252] J. Schmitz and J. Fulk. Organizational colleagues, media richness, and electronic mail: A test of the social influence model of technology use. *Communication Research*, 18(4):487–523, 1991.

[253] B. Schneider. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* Wiley, Oct. 1995.

[254] M. C. Schraefel, M. Wilson, A. Russell, and D. A. Smith. MSPACE: Improving Information Access to Multimedia Domains with MultiModal Exploratory Search. *Communications of the ACM*, 49(4):47–49, Apr. 2006.

[255] M. Seltzer, K. Bostic, M. McKusick, and C. Staelin. The Design and Implementation of the 4.4 BSD Log-structured File System. In *Proceedings of the 1993 Winter Usenix*, Jan. 1993.

[256] M. Seltzer, K. A. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan. File system logging versus clustering: A performance comparison. In *Proceedings of the 1995 USENIX Conference*, pages 249–264, Jan. 1995.

[257] S. Sen, S. K. Lam, A. M. Rashid, D. Cosley, D. Frankowski, J. Osterhouse, F. M. Harper, and J. Riedl. tagging, communities, vocabulary, evolution. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 181–190, New York, NY, USA, 2006. ACM Press.

[258] S. Shearin and H. Lieberman. Inteligent profilling by example. In *IUI 2001, Santa Fe, New Mexico, USA*, pages 145–151. ACM, Jan. 2001.

[259] D. Shenk. *Data Smog: Surviving the Information Glut.* HarperCollins Publishers, New York, NY, USA, 1997.

[260] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. RFC 3530: NFS version 4 protocol specification, Apr. 2003.

[261] Silicon Graphics, Inc. XFS: A high-performance journaling filesystem. `http://oss.sgi.com/projects/xfs/`.

[262] I. Smith. Historical notes about the cost of hard drive storage space. `http://www.littletechshoppe.com/ns1625/winchest.html`, Apr. 2004.

[263] K. A. Smith and M. I. Seltzer. File layout and file system performance. Harvard University technical report TR-35-94, 1994.

[264] K. A. Smith and M. I. Seltzer. File system aging: Increasing the relevance of file system benchmarks. In *SIGMETRICS '97: Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 203–213, New York, NY, USA, 1997. ACM Press.

[265] D. A. Solomon. *Inside Windows NT, 2nd edition.* Microsoft Press, Apr. 1998.

[266] M. Spasojevic and M. Satyanarayanan. An empirical study of a wide-area distributed file system. *ACM Trans. Comput. Syst.*, 14(2):200–222, 1996.

[267] T. Standage. *The Victorian Internet.* Walker and Company, 1998.

[268] K. Sullivan. The Windows 95 user interface: A case study in usability engineering. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 473–480, New York, NY, USA, 1996. ACM Press.

[269] Sun Microsystems, Inc. RFC 1094: NFS: Network File System Protocol specification, Mar. 1989. See also RFC1813 [57]. Status: INFORMATIONAL.

[270] L. Svobodova. File servers for network-based distributed systems. *ACM Comput. Surv.*, 16(4):353–398, 1984.

[271] W. R. Swartout. Future directions in knowledge-based systems. *ACM Computing Surveys (CSUR)*, 28(4):13, 1996.

[272] A. Swartz. RFC 3870: Application/RDF+XML Media Type Registration, Sept. 2004.

[273] Symantec Corporation. Veritas storage foundation. `http://www.symantec.com/Products/enterprise?c=prodinfo&refId=203`.

[274] T. Berners-Lee. Semantic Web road map. `http://www.w3c.org/DesignIssues/Semantic.html`, 1998.

[275] T O'Reilly. Web 2.0 Compact Definition. `http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html`, Oct. 2005.

[276] A. S. Tanenbaum. *Modern Operating Systems*, chapter 13, pages 549–573. Prentice Hall, first edition, 1992.

[277] A. S. Tanenbaum. *Modern Operating Systems*, chapter 6, pages 445–448. Prentice Hall, second edition, 2001.

[278] A. S. Tanenbaum. *Modern Operating Systems*, chapter 10, pages 732–744. Prentice Hall, second edition, 2001.

[279] A. S. Tanenbaum. *Modern Operating Systems*, chapter 8, page 564. Prentice Hall, second edition, 2001.

[280] A. S. Tanenbaum. *Modern Operating Systems*, chapter 6, pages 435–438. Prentice Hall, second edition, 2001.

[281] D. Taniar and J. W. Rahayu. *Web Semantics & Ontology*, chapter 1. IGI Publishing, 2006.

[282] R. B. Thompson and B. F. Thompson. *PC Hardware in a Nutshell*, page 506. O'Reilly, third edition, 2003.

[283] W. F. Tichy and Z. Ruan. A replicated, distributed file system. In *EW 2: Proceedings of the 2nd workshop on making distributed systems work*, pages 1–6, New York, NY, USA, 1986. ACM Press.

[284] H. Tirri. Search in Vain: Challenges for Internet Search. *Computer*, 36(1):115–116, Jan. 2003.

[285] A. Toffler. *Future Shock*. Bantam, 1970.

[286] J. D. Touch. Performance analysis of MD5. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 77–86, New York, NY, USA, 1995. ACM Press.

[287] W. Treese. Web 2.0: Is it really different? *netWorker*, 10(2):15–17, 2006.

[288] M. Uschold. Where are the semantics in the Semantic Web? *AI Mag.*, 24(3):25–36, 2003.

[289] U. Vahalia. *UNIX Internals: The New Frontiers, 1st edition*. Prentice Hall, Oct. 1995.

[290] R. van Zwol and A. Peter M. G. The webspace method: On the integration of database technology with multimedia retrieval. In *Proceedings CIKM 2000, McLean, VA*, pages 438–445. ACM, Feb. 2000.

[291] VERITAS software coorporation. VxFS commands. `http://eval.veritas.com/downloads/van/fs_quickref.pdf`, 2002.

[292] O. Vermeulen. CP/M internals. `http://www.dcast.vbox.co.uk/cpm.html`, Jan. 2003.

[293] W. Vogels. File system usage in Windows NT 4.0. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 93–109, New York, NY, USA, 1999. ACM Press.

[294] A. Voida, R. E. Grinter, N. Ducheneaut, W. K. Edwards, and M. W. Newman. Listening in: Practices surrounding itunes music sharing. In *Proceedings of CHI'05*, pages 191–201, 2005.

[295] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI conference on human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM Press.

[296] A. D. Vries. *Content and Multimedia Database Management Systems*. PhD thesis, University of Twente, Enschede, The Netherlands, Dec. 1999.

[297] P. Waddington. Dying for information: An investigation of information overload in the UK and world-wide, 1996. Reuters Business Information, London.

[298] D. Waltz and S. Kasif. On reasoning from data. *ACM Computing Surveys (CSUR)*, 27(3):356–359, 1995.

[299] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 10. The MIT Press, 1999.

[300] B. Welch and J. K. Ousterhout. Prefix tables: A simple mechanism for locating files in a distributed system. In *Proceedings of the 6th Conference on Distributed Computer Systems*, pages 184–189. IEEE, Oct. 1986.

[301] D. S. Weld, C. Anderson, P. Domingos, O. Etzioni, K. Gajos, T. Lau, and S. Wolfman. Automatically personalizing user interfaces. In *Proceedings of the 18th Joint International Conference on Artificial Intelligence, Acapulco, Mexico*, Aug. 2003.

[302] L. Wenyin, Z. Chen, F. Lin, H. Zhang, and W.-Y. Ma. Ubiquitous media agents: a framework for managing personally accumulated multimedia files. *Multimedia Systems*, (9):144–156, 2003.

[303] M. Wertheim. *The Pearly Gates of Cyberspace*, pages 221–225. Virago Press, 1999.

[304] E. J. Whitehead, Jr. and Y. Y. Goland. WebDAV: A network protocol for remote collaborative authoring on the web. In *Proc. of the Sixth European Conf. on Computer Supported Cooperative Work (ECSCW'99), Copenhagen, Denmark, September 12-16, 1999*, pages 291–310.

[305] S. Whittaker and C. Sidner. Email overload: Exploring personal information management of email. In *Proceedings SIGCHI 1996*, pages 276–283, 1996.

[306] S. Whittaker, L. Terveen, and B. A. Nardi. Let's stop pushing the envelope and start addressing it: A reference task agenda for HCI. In *Human Computer Interaction 15*, pages 75–106, Sept. 2000.

[307] S. Wildermuth. A Developer's Perspective on WinFS: Part 1. `http://msdn.microsoft.com/library/default.asp?url=`

`/library/en-us/dnwinfsta/html/winfsdevpersp.asp`, Mar. 2004.

[308] S. Wildermuth. A Developer's Perspective on WinFS: Part 2. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwinfsta/html/winfsdevpersppart2.asp`, July 2004.

[309] P. Wilson. *Computer supported cooperative work: An introduction.* Kluwer Academic Publishers, 1991.

[310] P. Wilson. Computer Supported Cooperative Work (CSCW): Origins, concepts and research initiatives. *Computer Network and ISDN Systems*, 23(1):91–95, 1991.

[311] B. Winston. *Media Technology and Society, A History: From the telegraph to the Internet*, pages 321–336. Routledge, 2000.

[312] D. Wolber, M. Kepe, and I. Ranitovic. Exposing document context in the personal web. In *IUI '02, San Francisco, California, USA*, pages 190–191. ACM, Jan. 2002.

[313] H. Wu, M. Zubair, and K. Maly. Harvesting social knowledge from folksonomies. In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 111–114, New York, NY, USA, 2006. ACM Press.

[314] R. Wurman. *Information Anxiety.* Doubleday, New York, NY, 1989.

[315] H. Yan and T. Selker. Context-aware office assistant. In *IUI 2000, New Orleans, LA, USA*, pages 276–279. ACM, 2000.

[316] B. Yu and M. P. Singh. An agent-based approach to knowledge management. In *Proceedings CIKM 2002, McLean, Virginia, USA*, pages 642–644. ACM, Nov. 2002.

[317] J. L. Zhao and V. H. Resh. Internet publishing and transformation of knowledge processes. *Communications of the ACM*, 44(12):103–109, Dec. 2001.

List of Figures

List of Tables