Directory/File Management System (DFMS) Vs Database Management System (DBMS)

Abstract

We present a directory/file management system designed to provide intelligent dynamic file management. To the best of our knowledge it is the first file management system that incorporates structure within the fundamental descriptions of the files. Microsoft's Operating Systems (Windows 7, ex.), for example, use databases within application software tools that allow users the ability to organize certain types of files. We have found no evidence of a file management system that can create logical views for all files found on the hardware.

We tested the architecture using a prototype DFMS built in Visual Basic and executed in a Windows environment, using a resident application program as a temporary extension to their operating system – our current implementation is a parasitic, not native, DFMS. However, nothing precludes one from using this architecture as the basis for the development of a full DFMS implementation, and this would be a cleaner implementation.

Introduction

Some existing operating systems, e.g. Windows, contain pre-built database structures which permit individuals to organize specific files in certain applications [1]. For example, users can organize their music files using pre-constructed categories, such as album, artist, or genre. Our system allows the user to create unlimited, logical views for the entire file system. This is achieved by placing our DFMS over the operating systems' pre-existing File Management system, in a layered architecture. Our system enables users to define meta-data (dynamic tags used to identify sharable content) which, in turn, allow the DFMS to sort pertinent data, such as phone numbers and addresses, into the logical views defined by the user [2]. This sorting is automated whenever the DFMS application program is executing, making updating this information automatic. For example, a user can maintain a resume and update files based on e-mail information without the hassle of cutting and pasting, or opening and saving the associated files. Moreover, this software feature will seamlessly facilitate basic work-flow applications.

Supplementary software would allow information stored on additional electronic devices, such as cell phones, to be automatically uploaded [3]. Tagged information from these devices would be updated in the computer each time the device was connected. This would significantly reduce the user's time spent personally updating files, and still produce the 'hand edited' precision users achieve doing these tasks manually.

Architecture

In recent years individual computers (including PC's) have become more complex. In consequence, the numbers of files needed to run them, as well as the types of files, have grown rapidly [4]. These files include video, audio, photographic, text, binary, and application specific files such as e-mail. All together, they number in the millions. Such large quantities of information require significant time and effort to organize. Presently this organizational task is

being left to the user [1]. Originally, computers had limited numbers of files and subsequently their file systems were flat. As computers began to evolve, hierarchical file systems began to develop [5]. These systems were designed to help refine file organization and searching, and to allow access to larger files through networking. Hierarchical file systems have been the norm for several decades.

In this paper, we outline the architecture for a DFMS - a directory/file management system. It's design is analogous to a DBMS – Data Base Management System [6]. Theoretically it would manage the physical underlying file structure in cooperation with the operating system directly. However, our prototype sits on top of the existing Operating System, demonstrating the proof of concept. Our DFMS allows the user to switch user type, and then displays only the file structure and associated files related to that type, hiding the existence of unrelated files.

In our prototype, we maintain a flat file of full-path-name-qualified files. Our prototype interface allows the user to tag each file with one or more associations (logical file types), independent of the physical path name, forming a group of like files. In order to reduce time spent accessing memory, and to reduce computational complexity, the flat file prototype supporting structure has been hashed. Each file can be placed into any number of groups by the user, as is the case with UNIX type groups [7]. For example, a user may choose to put a given file into a 'Wedding Planning' group, or a 'Classical Music' group, or both. They may also choose not to list the file in any group, making it exist in the physical view only. When a DFMS window is opened by the user, the user selects which group they prefer to work with, but can change groups at anytime. Full file functionality is available at both the physical and logical levels (ex. create new, remove, rename, etc...). The DFMS contains several built-in groups, such as resume, date, and time. This will, for example, allow a user the ability to reduce the amount of files they work with for specific tasks [8]. The user's ability to reconfigure the file structure to adapt to their own personal specifications may also reduce the keystrokes needed to locate and utilize files as the directory hierarchy may often times be significantly less deep.

Current business practices often limit user access to databases in order to protect important data from being accidentally altered or deleted. Users' access levels are determined by the tasks which they are routinely expected to perform. These practices are used to protect business' interests, and are warranted. However, current practices do not always allow for users with diversified assignments. In such cases a user may not be given access to files vital to the completion of a few tasks that they occasionally need to perform that fall outside their average job needs. Our DFMS supports a new business model that solves this existing Software Engineering problem. Switching logical views is equivalent to declaring that you are now performing a different job, and this will more easily allow access to required materials for tasks as needed by the user. Such an approach still provides security measures, but moves the protection mechanisms away from user-defined and towards task-defined. This in turn introduces versatility that is not easily achieved working from a computer's physical file system.

Detailed Description

We chose to implement our prototype using Visual Basic (VB). We felt VB would allow for simplicity in describing our approach. XML was used to create the group tag prototype, again for simplicity.

Our program begins by creating an index of all existing files on the host computer. To create this index, we begin at the root folder. Using the VB "directory.getfolders" method, we retrieve the names of all files present at that level of the physical file system from the operating system (any subroutine or method call can be used that retrieves the directory and file names). During this process any additional folder names are placed in a queue. When all files and folder names have been indexed from the root folder the task is repeated for all folders listed in the queue (any algorithm that generates a covering set will suffice [9]). The result of this process is a list of all directories and their associated folders and files on the host computer. Files may then be viewed in the operating system's original view or the user interface to our program will allow the user to switch to any of logical views they have created.

Alternatively, we could construct an index from shadow files created by a native DFMS. This would separate file information from the physical directory/file management system at the operating system level. However, our prototype does not function at the native file management system level, and hence we did not explore these options.

Each file in the native file management system is associated with an instance of a data structure class, which we call the fundamental data structure. Such an implementation is desirable, and is used to construct the interface between a DFMS and its host operating system. For platform independence, however, we discuss our implementation without this step. Our software imposes a logical structure over the underlying physical structure, which we refer to as our Directory/File Management System. This system creates logical file pathways from files located using the original operating system/file management system software. Files containing matching criteria are made available to the user, independent of all other files. This creates the user's "logical view". One or more logical views are available depending on the limitations set by the user. To alter which files are being viewed, a user simply redefines the file criteria.

This DFMS prototype, as described above, is accessed through a front- end GUI interface. This point and click interface allows the user substantial versatility. Users can choose to define their own views and directories or utilize those pre-defined by the system. The system can also be requested to construct an extension organization automatically, and uses natural language processing to organize the files automatically. The automatic organization is not error free. However it is convenient and sometimes helps the user achieve a better (more logically related) search result.

In the DFMS, when a user associates a file with a logical directory, a shadow file is created from the physical file. This shadow file allows the user access to the original file from any of the logical views to which it is linked. Regardless of how many logical directories are associated with a given file, no more than one reference is active at any given time. When no reference exists to the active logical view, the file is no longer visible to the user, unless they switch to the native file management system. Our Prototype, thus defined, is platform independent. While native file management systems are not platform independent, the DFMS should be constructed as a layered architecture, allowing the majority of the programming task to be platform independent, as our prototype is. This will allow user access to the higher level functions of the DFMS without installing or using other system's software programs –

decoupling it from the underlying file management system, and will enable more flexible DFMS development cycles. The look and feel of the interface may change as the application changes, but the interface to the underlying platform can remain stable. For each platform, however, similar groups of tools (combo boxes, text boxes, dialogue pop-ups etc…) will be used to guide the user through the full functionality of the DFMS software. This will allow the seamless use of the system without ever having to reach into an operating system's file explorer, to correctly work with the data and files that the user is managing.

Architecture Details

With a Physical File Structure only, the user accesses the files through the Operating System in combination with the file management system. This is shown in Fig. 1.

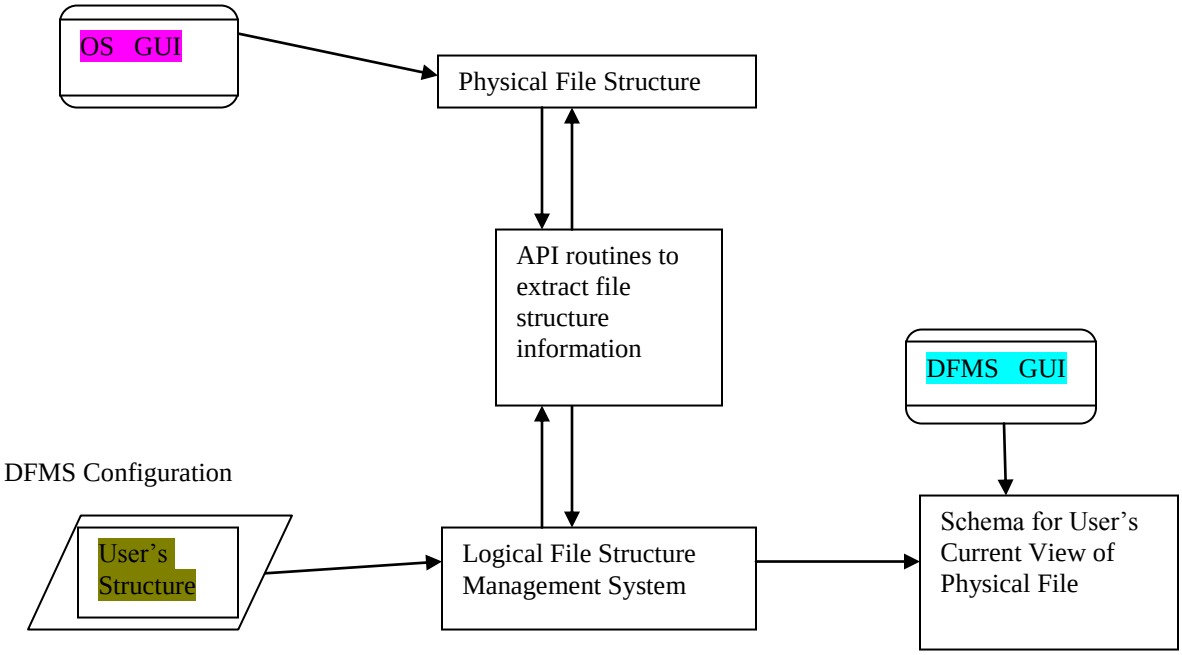

Fig 1. High Level Architecture of DFMS

To use the system, the user configures any number of schemas using the DFMS GUI Configuration Interface (highlighted in green). To change configurations and instantiate a new view they access the DFMS GUI to select a different logical view. For example, suppose our user is a salesman who is assigned to 200 suppliers. The salesman's physical file system may have a directory for each supplier. Suppose further that somewhere within each suppliers directory there is a list of supplies available from that supplier. Using the DFMS GUI Configuration interface, they can establish a file type as containing 'supply information'. They can still house these files with in the supplier directory, to keep their supplier information organized. But when they wish to compare the items offered by all of their suppliers, they instantiate the supply schema. To the user's view, they now have files organized to best present the supply information for comparisons. The files appear to have a different hierarchical relationship, one more conducive to the next task.

These schemas can have a structure imposed on them as well, so that file type 'supply information' may place the files within another directory structure. The user defines this structure (possibly hierarchical) through the DFMS Configuration Interface, and views this structure through the DFMS GUI.

Future Work and Conclusions

The extension to a DFMS moves the concept of data hiding up to the user interface level. The same approach that helps computer programmers manage complexity works for computer users as well. In addition, the layered architecture implementation, sitting on top of a physical file management system, requires very little communication between the two systems and is therefore a good candidate for such an architecture.

The next development step is to couple our DFMS to a file management system directly, instead of through the user interface to the file management system.

References

1. http://www.dummies.com/how-to/content/windows-7-for-dummies-cheat-sheet.html
2. Tom Rishel, Louise Perkins, **Sumanth Yenduri**, Farnaz Zand, "Determining the Context of Text Using Augmented Latent Semantic Indexing", Journal of the American Society for Information Science (JASIS), 58(17), pp 2197-2204, December 2007.
3. John Osborne David W. Russell, "Method and Apparatus for Storing and Retrieving Profile Data for Electronic Devices", United States Patent No. 7340244.
4. Computer Networks and Internets, Fifth Edition, Douglas E. Comer, Prentice Hall, ISBN-13:978-0-13-606127-4.
5. http://en.wikipedia.org/wiki/Hierarchical_database_model
6. Database Management Systems by Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill Science/Engineering/Math; 3 edition (August 14, 2002), ISBN-10: 0072465638.
7. http://www.columbia.edu/acis/webdev/unixgroups.html
8. Harris Wu, Michael Gordon, "Collaborative structuring: organizing document repositories effectively and efficiently", Comm. of the ACM, Vol. 50 , Issue 7, Pg: 86-91, 2007.
9. B. Berger, J. Rompel, P.W. Shor, "Efficient NC algorithms for set cover with applications to learning and geometry" 30th Annual Symposium on Foundations of Computer Science (FOCS 1989), Oct 30- Nov 1, ISBN: 0-8186-1982-1. 1989.