The Megadoc Image Document Management System

1 Abstract

Megadoc image document management solutions are the result of a systems engineering effort that combined several disciplines, ranging from optical disk hardware to an image application framework. Although each of the component technologies may be fairly mature, combining them into easy-to-customize solutions presented a significant systems engineering challenge. The resulting application framework allows the configuration of customized solutions with low systems integration cost and short time to deployment.

2 Electronic Document Management

In most organizations, paper is the main medium for information sharing. Paper is not only a communication medium but in many cases also the carrier of an organization's vital information assets. Whereas the recording of information in document format is done largely with help of electronic equipment, sharing and distribution of that information is in many cases still done on paper. Large-scale, paper-based operations have limited options for tracking the progress of work.

The computer industry thus has two opportunities:

1. Capture paper documents in electronic image format (if using paper is a requirement)

2. Provide better tools for sharing and distribution among work groups (if the use of paper can be avoided)

Organizations that use electronic imaging, as compared to handling paper, can better track work in progress. Productivity increases (no time is wasted in searching) and the quality of service improves (response times are shorter and no information is lost) when vital information is represented and tracked electronically.

Imaging is not a new technology (see Table 1). Moreover, this paper does not document new base technology. Instead, we describe the key components of an image document management system in the context of a systems engineering effort. This effort resulted in a product set that allows the configuration of customized solutions.

Those who first adopted the use of image technology have had to go through a long learning curve-a computer with a scanner and an optical disk does not fully address the issues of a large-scale, paper-based operation. Early adopters of electronic imaging experienced a challenge in defining the

right electronic document indexing scheme for their applications. Even though the technology is now mature, the introduction of a document imaging system frequently leads to some form of business process reengineering to exploit the new options of electronic document management. The Megadoc

image document management system allows the configuration of customer-specific solutions through its building-block architecture and its built-in customization options.

The Megadoc system presented in this paper is based on approximately 10 years of experience with base technology, customer projects, and everything in between. In those years, Megadoc image document management has matured from the technology delight of optical recording to an application framework for image document management. This framework consists of hardware and software components arranged in various architectural layers: the base system, the optical file server, the storage manager, and the image application framework.

The base system consists of PC-based workstations, running the Microsoft Windows operating system, connected to servers for storage management and to database services for document indexing. Specific peripherals include image scanners, image printers, optional full-screen displays, and optional write once, read many (WORM) disks.

The optical file server abstracts from the differences between optical WORM disks and provides the many hundreds of gigabytes (GB) of storage required in large-scale image document management systems.

The storage manager provides storage and retrieval functions for the contents of documents. Document contents are stored in "containers," i.e., large, one-dimensional storage areas that can span multiple optical disk volumes.

The Megadoc image application framework contains three sublayers:

1. Image-related software libraries for scanning, viewing, and printing

2. Application templates

3. A standard folder management application that provides, with some tailoring by the end-user organization, an "out-of-the-box" image document management solution

The optical file server and the storage manager store images in any type of document format. However, to meet customer requirements with respect to longevity of the documents, images should be stored in compressed format according to the Comité Consultatif Internationale de Télégraphique et Téléphonique (CCITT) Group 4 standard.

In addition to image document management solutions, Megadoc components are used to "image enable" existing data processing applications. In many cases, a data processing application uses some means of

identification for an application object (e.g., an order or an invoice). This identification relates to a paper document. Megadoc reuses the application's identification as the key to the image version of that document. Application programming interfaces (APIs) for terminal emulation packages that are running the original application in a window on the

Megadoc image PC workstations allow integration with the unchanged application.

The following sections describe the optical file server, the storage manager, and the image application framework.

3 Megadoc Optical File Server

The Megadoc optical file server (OFS) software provides a UNIX file system interface for WORM disks. The OFS automatically loads and unloads these WORM volumes by jukebox robotics in a completely transparent way. Thus, from an API perspective, OFS implements a UNIX file system with a large on-line file system storage capacity. Currently, up to 800 GB can be reached with a single jukebox.

We implemented the OFS in three layers, as shown Figure 1:

1. The optical disk filer (ODF) layer, which enables storing data on write-once devices and providing a UNIX file system interface.

2. The volume manager (VM), which loads and unloads volumes to and from drives in the jukeboxes and communicates with the system operator for handling off-line volumes.

3. The device layer, which provides device-level access to the WORM drives and to the jukebox hardware. This layer is not discussed further in this paper.

Optical Disk Filer

When we started to design the ODF, the chief prerequisite was that it should adhere to the UNIX file system interface for applications. The obvious benefit was that the designers would not have to write their own utilities to, for example, copy data, create new files, and make new directories. All UNIX utilities would work as well on WORM devices as on any other file system.

Current UNIX implementations provide two kernel interfaces for integrating a new file system type into the kernel: the file system switch (FSS), in UNIX versions based on the System V Release 3; and the virtual file system (VFS), in UNIX implementations like the System V Release 4, SunOS, and OSF /1 operating systems. We introduced the optical disk filer in the FSS and later ported it to the VFS.

The key challenge for the design of a file system for write-once devices is to allow updates without causing an "avalanche" of updates. Note that any update to a sector on a WORM device forces a rewrite of the full sector

at another location. If pointers to an updated sector exist on the WORM device, sectors that contain those pointers have to be rewritten, also. For example, if a file system implementation is chosen where the list of data blocks for a file, or just the sector location of such a list, is part of the file's directory information, any update to that file would cause a

rewrite of the directory sector and the sectors for the parent directories, all the way up to the root directory.

A second issue to be addressed for removable optical disks is performance. Access time for on-line disks is at least eight times slower than for current magnetic disks. (The average seek time for a WORM device is 100 milliseconds; rotational delay is about 35 milliseconds.) Fetching a disk from a jukebox storage slot, loading it, and waiting for spin-up takes between 8 and 15 seconds, depending on the type of jukebox.

Caching solves both issues. We decided that the usual in-memory cache would not be sufficient for the huge amounts of WORM data, and therefore, we use partitions of magnetic disks for caching.

ODF WORM Layout. To avoid duplicating previous efforts, we used classical UNIX file systems as a guideline for the definition of ODF's WORM layout. However, we had to add some indirect pointer mechanisms to avoid update avalanches. Each file system is mapped onto a single WORM partition. These partitions are written sequentially, reducing the free block administration to maintaining a current write point.

The ODF reuses many notions from UNIX file systems, such as i-nodes, superblock, and the functional contents of directory entries.[1] Applying these UNIX notions to the optical file system resulted in the following ODF characteristics:

- The superblock contains all global data for a file system.
- Each i-node contains the block list and all the attributes of a file except the file's name.
- An i-node number identifies each i-node.
- A directory is a special type of file.
- Entries in a directory map names to i-node numbers.

A new notion in the ODF, as compared to UNIX file systems, is the administration file (admin file). One such file exists for each file system. The file is sequential, and its contents are similar to the first disk blocks in classical UNIX file systems: the first extent contains the superblock, and all other extents form a constantly growing array of i-nodes; the i-node's number is the index of the i-node in the file's i-node array. An important difference between UNIX file systems and the ODF is that the 2-kilobyte (kB), fixed-size extents of the ODF admin file are scattered over the WORM device, instead of being stored as a sequential array of disk blocks, as in UNIX systems. As a result, any update to an i-

node, as a consequence of a file update, causes the invalidation of at most one admin file extent. Since the logical index in the admin file of this i-node, i.e., the i-node number, does not change, the parent directories do not have to be updated.

However, this scheme needs an additional indirect pointer mechanism: a list of block numbers representing the location of the admin file extents. The ODF stores this list in the admin file's i-node (aino). The aino is a sequential file that contains slightly more than block numbers and is a sequence of contiguous blocks on the WORM disk that contain the same information. Hence, an update to an admin file extent always invalidates the entire aino on the WORM device, which makes the aino a more desirable candidate for caching than the admin file extents.

The following example, shown in Figure 2, illustrates the steps involved in reading logical block N from the file with i-node number I:

1. Read the aino to obtain the block number of I's admin file extent.

2. Read the admin file extent to get file I, which is used to translate the logical block number N into the physical block number I(N).

3. Read physical block I(N).

If the file system is in a consolidated state, i.e., all data on the WORM disk is current, the aino and the superblock are the last pieces of information written to the WORM device, directly before the current write point. Blocks written prior to the aino and the superblock contain mainly user data but also an occasional admin file extent, fully interleaved. Figure 3 shows the WORM layout. Since ODF requires the first admin file extent and the complete aino to be in the cache, introducing a disk with consolidated file systems to another system requires searching the current write point, reading the superblock, determining the aino length from the superblock, and finally reading the aino itself. Searching the current write point is a fairly fast operation implemented through binary search and hardware support, which allow the ODF to distinguish between used and unused data blocks of 1K bytes.

ODF Caching. Caching in the ODF is file oriented. We suggest a magnetic cache size of approximately 5 percent of the optical disk space. If data from a file on a WORM disk is read, the ODF creates a cache file and copies a contiguous segment of file data from the WORM disk (64 kB in size, or less in the case of a small file) to the correct offset in the cache file. The cache file is the basis for all I/O operations until removed by the ODF, after having rewritten all dirty segments (i.e., updated or changed segments) back to the WORM device. The ODF provides special system calls (through the UNIX fcntl(2) interface) to flush asynchronously dirty file segments to the WORM device and to remove a file's cache file. The flusher daemon monitors high and low watermarks for dirty cache contents. The daemon flushes dirty data to the optical disks. The flusher daemon flushes data in a sequence that minimizes the number of WORM volume movements in a jukebox. The ODF deletes clean data (i.e., data already present on the

optical disk) on a least-recently-used basis.

The admin file has its own cache file. The minimum amount of admin file data to be cached is the superblock. The ODF gradually caches the other admin file extents, which contain the i-nodes, while the file system is in use. The ODF writes i-node updates to the WORM device as soon as all i-nodes in the same admin file extent have their dirty file data written to the WORM device. The aino has its own cache file, also, and is always completely cached. If all file data and i-nodes have been written to the WORM device, the file system can be consolidated by a special utility that writes aino and superblock to the WORM device, hence creating a consolidation point.

For reasons of modularity and ease of implementation, we chose the UNIX standard magnetic disk file system implementation to perform the caching. An alternative would have been to use a magnetic disk cache with an optimized, ODF-specific structure. We opted for a small amount of overhead, which would allow us to add a faster file system, should one become available. Our performance measurements showed a loss of less than 10 percent in performance as compared to that of an ODF-specific solution. The cache file systems on magnetic disk can be accessed only through the ODF kernel component. Thus, in an active OFS system, no application can access and, therefore, possibly corrupt the cached data.

Volume Manager

In addition to hiding the WORM nature of the underlying physical devices, the OFS transparently moves volumes between drives and storage slots in jukeboxes that contain many volumes ("platters"). The VM performs this function.

The essential characteristic of the volume management layer is its simple functionality, which is best described as a "volume faulting device." The interface to the VM consists of volume device entries, each of which gives access to a specific WORM volume in the system. For example, the volume device entry /dev/WORM_A gives access to the WORM volume WORM_A. This volume device entry has exactly the same interface as the usual device entry such as /dev/worm, which gives access to a specific WORM drive in the system, or rather to any volume that happens to be on that drive at that moment. Any access to a volume device, e.g., /dev/WORM_A, either passes directly to the drive on which the volume (WORM_A) is loaded, or results in a volume fault. This last situation occurs when the volume is in a jukebox slot and not in a directly accessible drive. Note that since /dev/WORM_A has the same interface as /dev/worm, the OFS could function without the VM layer in any system that contains only one worm drive and one volume that is never removed from that drive. However, since this configuration is not a realistic option, the OFS includes the VM layer.

The internal architecture of the VM is more complicated than its

functionality might indicate. The VM consists of a relatively small kernel component and several server processes, as illustrated in Figure 4. The kernel component is a pseudo-device driver layer that receives requests for the volume devices, e.g., /dev/WORM_A, and translates these requests into

physical device driver (/dev/worm) requests using a table that contains the locations of loaded volumes. If the location of a volume can be found in the table, the I/O request is directly passed on to the physical device. Otherwise, a message is prepared for the central VM server process, and the volume server and the requesting application are put in a waiting state.

The volume server uses a file to translate volume device numbers into volume names and locations. It communicates with two other types of VM server processes: jukebox servers and drive servers. The jukebox servers take care of all movements in their jukebox. Drive servers spin up and spin down their drive only on request from the volume server.

4 Storage Manager

The storage manager implements containers, as mentioned in the Electronic Document Management section. Large-scale document management uses indexing of multiple storage and retrieval attributes, typically with the help of a relational database. Once the contents of a document are identified through a database query on its attributes, a single pointer to the contents is sufficient. Also, there is little need for a hierarchically structured file system. Containers provide large, flat structures where the contents of a document are uniquely defined by the container identification and a unique identification within the container. The document's contents identification is translated by the storage manager in a path to a directory where one or more contents files can be written. For multipage image documents, the Megadoc system stores each page as a separate image file in a directory reserved for the document. This scheme guarantees locality of reference, avoiding unnatural delays while browsing a multipage image document.

A container consists of a sequence of file systems, typically spanning multiple volumes. Due to the nature of the OFS, no distinction has to be made between WORM disk file systems and magnetic disk file systems. The storage manager fills containers sequentially, up to a configurable threshold for each file system, allowing some degree of local updates (e.g., adding an image page to an existing document). As soon as a container becomes full, a new file system can be added.

Containers in a system are network-level resources. A name server holds container locations. Relocation of the volume set of a container to another jukebox, e.g., for load balancing, is possible through system management utility programs and can be achieved without changing any application's indexing database.

5 RetrievAll-The Megadoc Image Application Framework

Early Megadoc configurations required extensive system integration work. RetrievAll is the second-generation image application framework (IAF). The

first generation was based on delivery of source of example applications. However, tracking source changes appeared to be too big of an issue and hampered the introduction of new base functionality.

The Megadoc Image Document Management System

In cooperation with European sales organizations, we formulated a list of requirements for a second-generation IAF. The framework must

1. Allow for standard applications. Standard applications, i.e., scan, index, store, and retrieve, cover a wide range of customer requirements in folder management. Tailoring standard applications can be accomplished in one day, without programming effort.

2. Be usable in system integration projects. The IAF must provide APIs for folder management, allowing the field to build applications with functionality beyond the standard applications by reusing parts of the standard applications.

3. Allow image enabling of existing applications. RetrievAll should allow the linkage of electronic image documents and folders with entities, such as order number or invoice number, in existing applications. Existing applications need not be changed and run on the image workstation using a terminal emulator running at the image workstation.

4. Accommodate internationalization. All text presented by the application to the end user should be in the native language of the user. RetrievAll should support more than one language simultaneously for multilingual countries.

5. Allow upgrading. A new functional release of RetrievAll should have no effect on the customer-specific part of the application.

6. Provide document routing. After scanning the documents, RetrievAll should route references to new image documents to the in-trays of users who need to take action on the new documents.

Image Documents in Their Production Cycle

Image documents start as hard-copy pages that arrive in a mailroom, where the pages are prepared for scanning. Paper clips and staples are removed, and the pages are sorted, for example, per department. An image batch contains the sorted stacks of pages. The scanning application identifies batches by a set of attributes. The scanning process offers a wide variety of options, including scanning one page or multiple pages, accepting or rejecting the scanned image for image quality control, batch importing from a scanning subsystem, browsing through scanned pages, and controlling scanner settings.

The indexing process regroups image pages of an image batch into multipage image documents. Each document is identified with a set of configurable attributes and optionally stored in one or more folders. Folders also carry a configurable set of attributes. On the basis of the attribute

values, the document contents are stored in the document's storage location (container).

Many users of RetrievAll applications use the retrieve functions of the application only to retrieve stored folders and documents. Folders and documents can be retrieved by specifying some of the attributes. RetrievAll allows the configuration of query forms that represent different views

on the indexing database. The result of a query is a list of documents or folders. For documents, the operations are view, edit, delete, print, show folder, and put in folder. The Megadoc editor is used to view and to manipulate the pages of the document including adding new pages by scanning or importing. For folders, the operations are list documents, delete, and change attributes.

Document Routing Applications

A RetrievAll routing application is an extension to a folder management application. A route defines how a reference to a folder travels along in-trays of users or work groups.

Systems Management

The following systems management functions support the RetrievAll package:

- Container management
- Security, i.e., user and group permissions
- Logging and auditing
- Installation, customization, tailoring, and localization

Architecture and Overview

As illustrated in Figure 5, the RetrievAll image application framework consists of a number of modules. Each module is a separate program that performs a specific function, e.g., scanning or document indexing. Each module has an API to control its functionality, and some modules have an end-user interface. Modules can act as building bricks under a control module. For example, an image document capture application uses

1. Scan handling, to let an end user scan pages into a batch.
2. Scanner settings, to allow the user to set and select the settings for a scanner. The user can save specific settings for later reference.
3. Batch handling, to allow the end user to create, change, and delete batches.

These three modules can operate together under the control of the scan control module and in this way form a document capture application. The scan control module can, under control of a main module, perform the document capture function in a folder management application.

Modules communicate by means of dynamic data exchange (DDE) interfaces provided in the Microsoft Windows environment. Each module, except the main module, can act as a server, and all modules can act as clients in a DDE communication.

Main Module. Any RetrievAll application has a main module that controls the activation of major functions of the application. These functions include scanning pages into batches, identifying pages from batches into multipage image documents and assigning documents to folders, and

retrieving documents and folders. The main module presents a menu to select a major function. The main module activates the control modules of the major functions in an asynchronous way. For example, the main module can activate a second major function, e.g., retrieve, when the first major function, e.g., identification, is still active.

Control Modules. Each major RetrievAll function has a control module that can run as a separate application. For example, when a PC acts as a scan workstation, it is not necessary to offer all the functionality by means of the main module. Control modules can be activated as a server through the DDE API with the main module as client or as a program item from a Microsoft Windows program group.

Server Modules. All modules, with the exception of the main module, act as DDE server modules.

Configuration files hold environment data for each module. An application configuration file describes which modules are in the configuration. The layout of the configuration files is the same as the WIN.INI file used by the Microsoft Windows software, allowing the reuse of standard access functions.

Making an Application

An application can be made by selecting certain modules. Figure 5 gives an overview of the modules used for the standard folder management application. The installation program, which is part of the standard applications, copies the appropriate modules to the target system and creates the configuration files.

Modules can also be used with applications other than the standard ones. Image enabling an existing (i.e., legacy) application (see Figure 6), such as an order entry application where the scanned images of the orders should be included, entails the following:

- The existing application is controlled by a terminal emulator program running in the Microsoft Windows environment. This terminal emulator program must have programming facilities with DDE functions.

- While entering a new order into the system, the image document representing the order is on the screen. The function to include the image can be mapped on a function key of the emulator. Pressing the function key results in a DDE request to the identification function of the RetrievAll components. This DDE request passes the identification of the document (as known in the order entry application) to the identification function.

6 Summary

This paper has provided an overview of the many components and disciplines needed to build an effective image document management system. We discussed the details of the WORM file system, the storage manager technology, and the image application framework. Other aspects such as WORM peripheral technology, software compression and decompression of images, and the integration of facsimile and optical character recognition technologies were not covered.

From experience, we know that different customers have different requirements for image document management systems. The same experience, however, taught us to discover certain patterns in customer applications; we captured these patterns in the application framework. The resulting framework allows us to build highly customized applications with low system integration cost and short time to deployment. Future directions are in the area of enhanced folder management and integrated distributed work flows.

7 Reference

1. M. Bach, The Design of the Unix Operating System, ISBN 0-13-201757-1 (Englewood Cliffs, NJ: Prentice-Hall, 1986).

8 Trademarks

9 Biographies

Jan B. te Kiefte Jan te Kiefte is technical director for Digital's Workgroup Systems Software Engineering Group in Apeldoorn, Holland. He has over 20 years' software engineering experience in compiler development and in the development of office automation products. Jan has held both engineering management positions and technical consultancy positions.

He has an M.Sc. degree in mathematics from the Technical University of Eindhoven, Holland.

The Megadoc Image Document Management System

Bob Hasenaar Bob Hasenaar is an engineering manager for the Megadoc optical file system team, part of Digital's Workgroup Systems Software Engineering Group in Apeldoorn, Holland. He has seven years' software engineering experience in operating systems and image document management systems. Bob was responsible for the implementation of the first Megadoc optical disk file system in a UNIX context. He has an M.Sc. degree in theoretical physics from the University of Utrecht, Holland.

Joop W. Mevius A systems architect for the Megadoc system, Joop Mevius has over 25 years' experience in software engineering. He has made contributions in the areas of database management systems, operating systems, and image document management systems. Joop has held both engineering management positions and technical consultancy positions. He has an M.Sc. degree in mathematics from the Technical University of Delft, Holland.

Theo M. van Hunnik Theo van Hunnik is an engineering project manager for Digital's Workgroup Systems Software Engineering Group in Apeldoorn, Holland. He has over 20 years' software engineering experience in compiler development and the development of office automation products. Theo has participated in several international systems architecture task forces. He managed the development team for RetrievAll, the Megadoc image application framework.