A storage subsystem for image and records management

by H. M. Gladney

Digital storage and communications are becoming cost effective for massive collections of document images with access not only for nearby users but also for those who are hundreds of miles from their libraries. The Document Storage Subsystem (DocSS) provides generic library services such as searching, storage, and retrieval of document pages and sharing of objects with appropriate data security and integrity safeguards. A library session has three components: a manager of remote catalogs, a set of managers of large-object stores, and a manager of cache services. DocSS supports all kinds of page data—text, pictures, spreadsheets, graphics, programs—and can be extended to audio and video data. Document models can be built as DocSS applications; the paper describes a folder manager as an example. What differentiates DocSS among digital library projects is its approach to data distribution over wide area networks, its client-server approach to the heterogeneous environment, and its synergism with other components of evolving open systems.

Replacing paper in massive administrative applications by raster image data is of high current interest. Electronic libraries for scientific and cultural collections are drawing similar attention from a different community. This paper describes a Document Storage Subsystem (DocSS) that knits together more basic software components to create the digital analog of a conventional library.

DocSS originated in 1987 inquiries into supplanting the use of paper by engineers, doctors, lawyers, and other professionals for the well-known advantages of digital media. Within the Research Division of IBM, it quickly became apparent that replacing even a portion of the division's dependency on paper would be a massive task involving undeployed technologies and some invention. Support for a professional staff is particularly difficult because of the heterogeneity of the tools, working methods, and objectives even within a single profession or a single enterprise. In contrast, we observed that large clerical staffs have relatively homogeneous needs, especially for their core activities. We further found that some massive clerical automation opportunities—in governmental human services, documentation for regulated industries, and civil engineering, in addition to the usual insurance and banking applications—required only functional subsets of what a professional staff needed.

As a practical matter, it was possible to devise, finance, and bring to market an offering for such applications, with a storage system architecture that supports a long-term goal of IBM. This offering, the IBM Image and Records Management (IRM) system,[1] combines a partial implementation of DocSS with sibling parts of a toolkit—scan, print, and presentation services, optical character recognition and related information capture routines, distributed work list management, and a folder manager. By showing how DocSS is re-

lated to other parts of the IRM product, the paper suggests how it can contribute to other packaged offerings.

Library service is seen by many people as an enhancement of existing services that could store data from many applications. Encoded data are usually more useful and cheaper to handle than raster image representations of the same information. Raster image is needed for photographs, for rescuing existing paper like the 40000000 California birth, death, and marriage certificates deteriorating in archives, and for dealing with incoming paper such as income tax returns. DocSS handles all kinds of objects. Raster image data are prominent because they dominate performance. The reader should accept a broad construction of the word "image"; in what follows, an image is a representation of something other than itself. Sometimes the coined word "blob" (*b*inary *l*arge *ob*ject) replaces "image" to emphasize that DocSS does not interpret the data that it holds and catalogs.

DocSS is responsive to requirements gleaned from more than a dozen in-depth application studies. Since the needs of a state highway department illustrated generic requirements better than any other single case and because of the quality of a consultant's analysis, the paper uses this case to motivate the architectural aspects emphasized. A recent term exercise[2] for students, closely related to an American Physical Society projection,[3] proposed national distribution of scholarly publications; it will be seen that, in the service niche that it addresses, DocSS comes remarkably close to providing what such diverse studies call for. What differentiates DocSS from any competitive technology is its approach to data distribution over wide area networks, its client-server approach to the heterogeneous environment, and its synergism with other components of evolving open systems.

Document library applications and requirements

This study was stimulated by inquiries about replacing significant amounts of paper with digital representations—inquiries from within IBM and from associates in state government. In the IBM research environment in which the author is employed, storing and cataloging images and graphics present problems that each scientist must address individually, or forego using such data. Consequently people use and share such data much less than might be desirable.

Doubting that our own working environment typified more common situations, we analyzed two dozen outside applications,[4] mostly in the government of the State of California. Their needs seemed to be spanned by four cases: a vital records archive, collection of commercial income taxes, historical data in a large law office, and an engineering collection called "Bridge Books." Only the last one is described below. Of the library applications we inspected, it attracted us most as a research target. It intermixes image, graphics, text, geographic, video, and audio data. It is big, challenging, and contains most of the ingredients found in all of the others collectively.

Highway engineering. In a 1985 highway records management study, Arthur Young and Company (AY) recommended replacement of mostly manual document systems by a comprehensive optical disk records management system. The study found that "The California Department of Transportation (Caltrans) is responsible for the planning, design, construction administration, and maintenance of State highway facilities. These activities have resulted in ... documents ... which date back prior to 1930. Estimates ... indicated that approximately forty-eight million pages of environmental, project development, structures maintenance, right-of-way, and construction documents were being maintained in various manual filing systems at headquarters, district offices, and ... record centers. Not included in this estimate were documents maintained by Planning and Programming, Maintenance and Transportation Operations, and Administration and Finance."[5]

Over 2000 Caltrans engineers work in 11 district offices separated by hundreds of miles. A highway project may last several years and involve many different groups: headquarters evaluation and approval, design squad, traffic survey, right-of-way, appraisals, and environmental impact. Central document repositories reside in Sacramento and Los Angeles. Caltrans has supplied many engineers with intelligent workstations and CAD (computer-aided design) software and requires a shared document repository that is unobtrusively accessible.

"Bridge Books" collects into binders the construction and inspection history of each of 36 000 bridges and highway overpasses. Although grey scale and color photographs are important, what primarily makes this an image application is existing paper. The 50 million pages of drawings and reports are classified into 48 categories. How many pages are originals is not known; AY estimated 2 to 30 copies of each original. Six million new originals enter the system each year, i.e., about one arrival every three seconds if input occurs during normal office hours. Because of inadequate housing, the sole copy of many documents is at risk from fire or water damage.

Document retrieval rates were projected at 1600 per day and 500 during peak hours; sizes of documents were not specified. Refinement of the interactive search is an explicit requirement. Workstation responsiveness targets were stated as: eight seconds to identify candidate documents from indices; 20 seconds to present the first sheet of a selection; and one second for subsequent sheets when the system is 80 percent loaded. We believe that users will not be satisfied with this performance.

AY judged that the cost effectiveness of optical character recognition is still to be demonstrated for Caltrans.[6] The size of the conversion needed to digitize "old paper" will force the question to be reconsidered. Samples suggest that each document in a class has the same kind of indexing information located in one of a few places on the paper. An obvious start would be to scan several thousand samples, test whether simple transformations yield numerical signatures that sort the images into types and the extent to which document analysis[7] and optical character recognition can replace manual indexing.

What data search and information representation features individual engineers want was not carefully studied. The consultant's analysis[5] of current procedures emphasized the quality control problems inherent in a massive paper system:

- Loss of individual documents or index entries, or both
- Loss of historical and legal documents because of varying retention procedures
- Inability to search on secondary keys
- Inability to locate the sole copy if checked out
- Distribution of related data among several locations
- Difficulty of updating records when indices change, e.g., post mile limits
- Cost of managing project splits and combinations

More subtle quality controls may be helpful but were not mentioned. In this collection, as in every other we have observed, controls end at the level of folders. Nothing prevents improper insertion or removal of a sheet from a folder or misfiling, and there is no help for auditing adherence to prescribed procedure.

Quite simple searches into the collection can have significant impact. Caltrans sometimes loses tort litigation because it is not able to produce records in a timely fashion. We even heard of a bizarre episode in which the highway agency had been negotiating with a city to buy some real estate, only to discover that it already owned the parcel. Apparently the city had started using the idle land many years previously.

Role of document storage services. Document storage and access software can be realized in two layers above a base of file systems and database managers (Figure 1). The lower layer runs DocSS, which stores and retrieves objects to and from each library collection, updates and searches library catalog records, and limits who can manipulate which data, giving only services that are identical for all types of documents. Instances of the higher layer, called *document managers*, help applications or end users with their special kinds of documents. The storage subsystem layer provides generic document storage services; each of potentially many document managers implements a model such as hypertext. Application programs are workstation programs; the storage subsystem embeds needed interprocess and intermachine communications.

Typical document managers interpret scanned data to create catalog entries automatically, manage interrelationships among documents, facilitate the most common search methods, and help move information among workers. For instance:

- A folder manager might scan electronic memoranda, letters, contracts, and financial records; such a manager would extract names, addresses, and dates to cross-index information

received[8] and associate each document with an account folder. It might further model the information flow of a business core, such as "back room" operations in an insurance company.

- The entities of a second document manager might be movies; it would communicate with its users in terms of movies, reels, and frames and depend on a storage subsystem with video delivery channels.
- A third document manager might feature a CAD system and be applied to maintenance records of government buildings; it would generate and display building plans with a graphic editor and maintenance contracts with a customized text editor.
- A fourth document manager might model what is found in a university library—books and pamphlets with individually viewable pages, loose collections of papers in folders, manuscripts, videotapes, and so on.

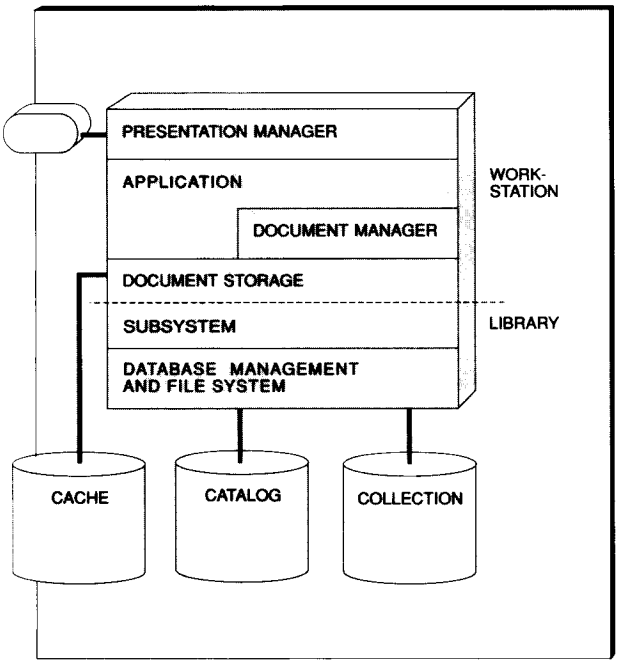
Generic document managers for applications like personnel services and enterprise-specific ones administering conventions and document quality standards may evolve over time. DocSS attempts comprehensive coverage of functional requirements by relatively primitive operations with many options; good document managers would offer less flexibility and fewer options but would be much easier to explain and understand.

Applications and document managers execute in users' machines; DocSS provides storage services and manages intermachine communications, hiding them to the extent possible. Implementation follows a client-server approach.

Comparison with offices based on paper. The digital library reproduces essential characteristics of systems based on paper, emulating aspects of using or managing a library—whether public, private, professional, or school—which has books, pictures, and other material objects. For instance,

- Users are usually somewhere away from the library information they need and often need items from several libraries concurrently.
- Whoever wants to use a library must show that he or she has permission to do so.
- Users are not necessarily those to whom permission was given, i.e., requesters are active entities distinct from patrons.

Figure 1 Partitioning of library service software



- Different patrons are permitted different actions and allowed to see different parts of the collection.
- The catalog and the collected objects are distinct entities, used differently and not necessarily housed in the same place.
- The catalog may describe items not actually held as part of the library collection.
- Translations of a document may express essentially the same information, e.g., versions of classic literature in different languages.
- Document identifiers are different from document names; a document may have several names, one for each context, e.g., *Tales of Hoffmann* in English, *Les contes d'Hoffmann* in French, and *Hoffmanns Erzählungen* in German.
- Documents are cataloged with text descriptors and also with conventional properties, such as author names.
- Documents contain cross references to other documents.
- To find anything, each user must understand the catalog structure.

Like its material counterpart, the digital library is intended for objects that are worth retaining for

long periods and are valuable to many people. Such documents are used differently than papers carried in briefcases and stored in desks. They tend to be static, e.g., a 1965 photograph of a building does not change without being considered to be some other object. In any period of several months, most of the collection is not looked at.

Picture storage must blend into each user's existing hardware and software environment.

The advantages of a digital library over a paper library are similar to those of any digital database over its paper counterpart: faster addition to the data collection, improved search functionality, faster distribution from the point of storage to the point of usage, better history tracking, and finer granularity of control.

The benefit of improved control is not only improved data quality, but also more freedom and reduced bureaucracy for individual users. Only a librarian may add to the collection of a paper library because of the discipline essential to create a quality catalog. In a digital analog, cataloging discipline and search restrictions to authorized data can be automatically enforced. An electronic library can allow each patron a wider range of services than is practical with a paper library.

A summary of requirements. Any social unit (school, business, department, individual) might create and manage its own library, and most individuals want access to many libraries. Picture storage must blend into each user's existing hardware and software environment; it must be minimally obtrusive to the user's favorite applications and support all of the user's object types. All libraries should do certain things similarly, e.g., adhere to certain standards, so that people do not need to learn new methods for each library and so that information can be exchanged.

Our 1987-1988 application analyses identified several hundred specific requirements—too many to tabulate here. However, several broadly applicable elements emerged and are summarized below because they determine the structure of DocSS.

Least surprise for users. The concept of "library" has been refined over several centuries. It would be injudicious to depart from what people expect merely because a digital service is replacing a material one. Except where explicit reasons suggest an improvement that is easily explained (e.g., in query services), library services should implement a model familiar to everyone.

Distribution. People are often distant from needed information, frequently in locations for which high-speed links are not affordable. Mobile clients, such as police officers, want access over radio links. Recipients of large objects often want delivery over common carrier links delayed to times when communication tolls are low.

Performance. Updating a stored document is likely to be a rare event and not subject to stringent responsiveness objectives. In contrast, retrieval should be rapid, and a search to identify which objects are worth retrieving should be even more rapid.

Large and small objects. Object sizes range from about 1000 bytes for financial transaction records to 10 million bytes for technical pictures. When digital video and audio libraries become practical, even bigger objects will have to be delivered with controlled pacing.

Accessibility from all workstation platforms. Different workers in the same department may have different kinds of machines because of history, function needed, or personal preference. Each library must be accessible from whatever workstation has been chosen.

Catalog service from all kinds of operating system platforms. A large enterprise may have different kinds of database servers in different locations and should be able to provide compatible library catalog services from these database servers.

Support for all kinds of image storage. Custodians should be able to house image collections as economically as possible within their operational

and policy constraints. They should be able to augment capacity with whatever is the currently most effective storage medium and attach this medium in the network wherever needed to minimize communication costs and maximize responsiveness.

Low entry point, with growth to giant collections. Library service offerers want to start cheaply and to grow without disruption or breakage to large numbers of clients and very large databases. There may be 10 000 subscribers to each library, with 1000 connected at once. State of California paper collections typically contain 10 to 100 million items. International banks are considering collections of 1 to 10 billion items held for up to 30 years.

Low installation and administration overhead. Installation and custodial responsibilities for a library should require only a small addition in time and training for data administrators. Installation and use of the workstation portion of library services should be easy given only "shrink wrap" materials. Protecting the catalog and collection will require the infrastructure and discipline of a "glass house" (traditional air-conditioned, large-computer) environment.

Joining libraries to other databases. People want easy use of library data in unanticipated ways, joining library catalogs to enterprise databases and combining data across agencies (e.g., toxic waste data with death certificates) and sometimes across administrative jurisdictions (interstate, county-to-state, etc.). The ability to do a particular correlation on short notice and with low cost is of keen interest.

Application independence. Cataloging documents is economical primarily under the presumption of future pertinence to multiple, unanticipated applications. The utility of stored data is hampered by anything that tailors the data to one application or one usage paradigm in preference to alternatives. An application-neutral interface is needed.

Document managers. The kind of library layer implied by the above requirement is too primitive for most enterprises. It should be augmented by document managers that support broadly applicable application paradigms or the needs of specific kinds of customers, e.g., university libraries, or both.

Tool for "amateur" application programmers. Library services should be as accessible to knowledge workers as editors and spreadsheets are, requiring a very short learning period. Given that many people entering the work force have some programming training, providing library access to shell languages will permit *ad hoc* applications.

Customer-defined data formats. Each enterprise should be able to define the format in which it stores images and other objects, with the understanding that it must store enough collateral information for object interpretation. For industry-standard or IBM-standard data formats, the library service must provide the collateral information (this may be within the data objects themselves, as in MO:DCA, or Mixed Object Document Content Architecture).

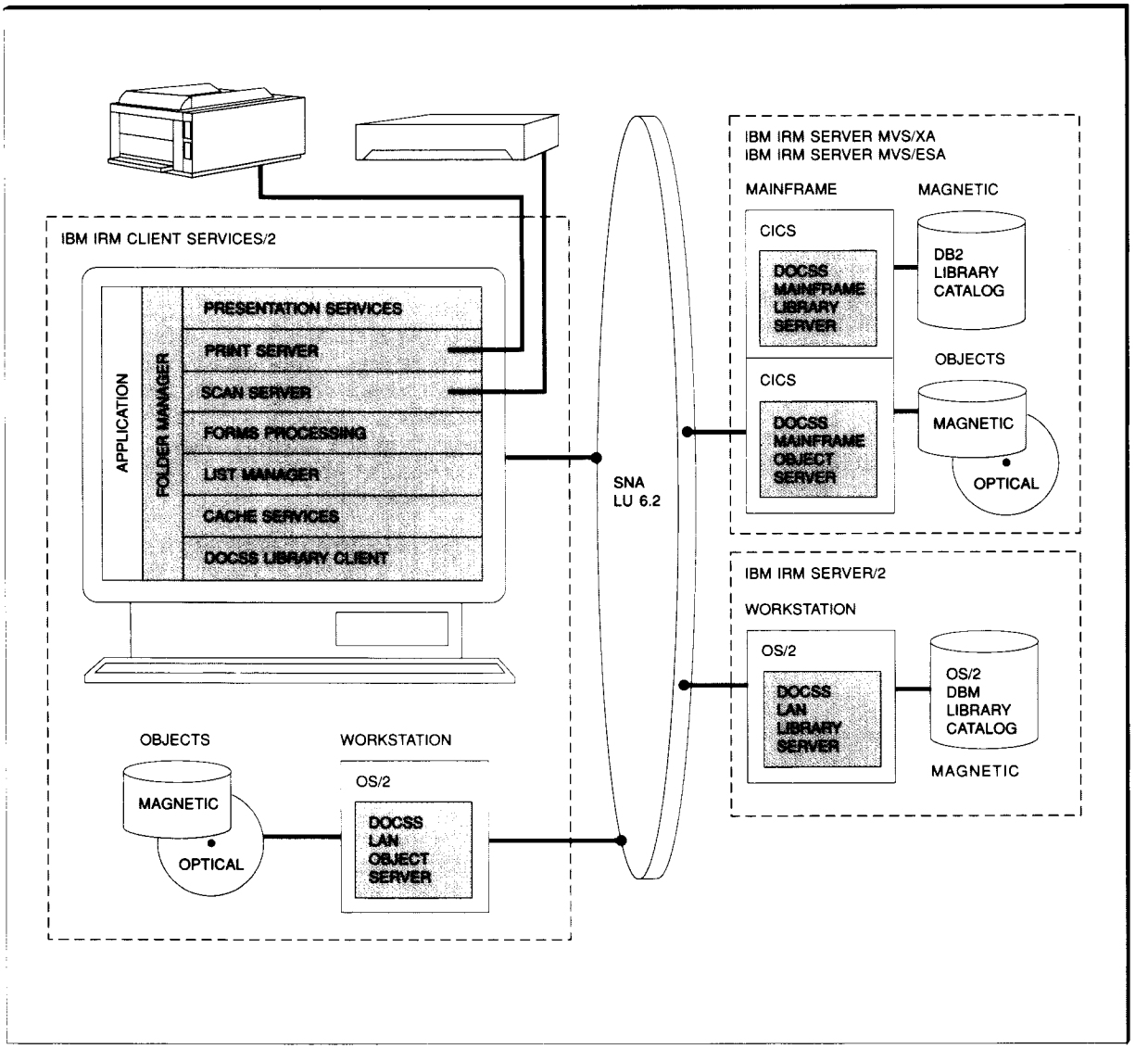
Automatic capture and indexing. Today the biggest inhibitor of large digital collections is the cost of capturing information from paper and indexing it for search and retrieval. Giant libraries can be achieved only with automatic means of capturing information.

Open subsystem. Emerging workstation application packages—text, graphic, image, and audio editors, spreadsheets, CAD packages, and industry support packages such as those for hospitals and for doctors' offices—are potential sources and sinks for large numbers of electronic documents. Value-added vendors have much to offer users, who therefore want systems with easy access to programming interfaces.

Standard interfaces and protocols. The previous requirements imply a long-term commitment to an application programming interface for library services and to protocols for the interchanges among library clients, library servers, and image servers. Library service implementations should conform to pertinent national and international standards and programming conventions needed for application program portability, such as Systems Application Architecture* (SAA*) and Open Software Foundation Distributed Computing Environment (OSF/DCE**).

The Image and Records Management product. The IRM product[1] complements operating system of-

Figure 2 Structure and platforms of IRM V1R2



ferings with a document management toolkit. Application programs execute in Personal System/2* (PS/2*) machines running Operating System/2* (OS/2*); library catalog servers execute either in mainframes running Multiple Virtual Storage/Enterprise Systems Architecture (MVS/ESA*) or Multiple Virtual Storage/Extended Architecture (MVS/XA*) with Customer Information Control System (CICS*) and DATABASE 2* (DB2*) or in PS/2 machines with OS/2 Extended Edition; "blob" servers are available for both mainframe or workstation environments and support both magnetic and optical disk storage. The mainframe blob server is, in fact, based on the Data Facility Product-Object Access Method (DFP-OAM) part of the ImagePlus* System MVS/ESA product.[9]

In addition to DocSS, the product contains presentation modules exploiting OS/2 Presentation Manager*, print and scan servers for raster im-

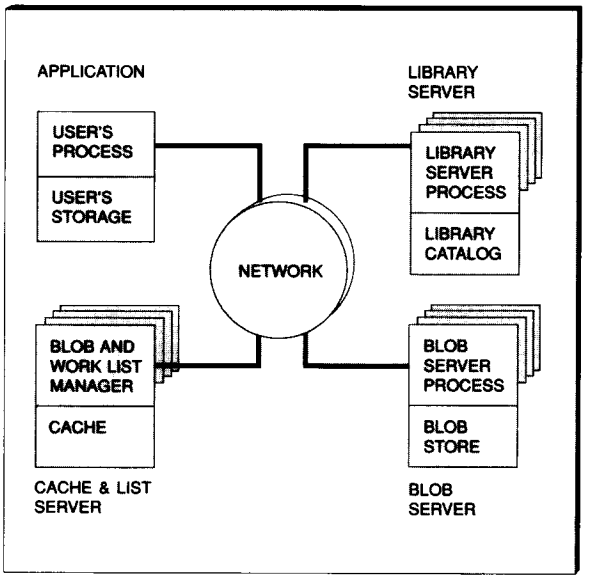
ages, forms processing subroutines, a distributed list manager that is a partial basis for work flow management, and a folder manager. The forms processing portion implements novel algorithms for deciding whether a raster image represents a known kind of form (e.g., an I.R.S. 1040 form, birth certificate, etc.), dropping out and later adding back form "boilerplate" for image compression and preparation for optical character recognition,[10] and an adaptation of the field extraction and optical character recognition package called Intelligent Forms Processing (IFP).[11] Figure 2 depicts how these components are packaged. The colored boxes in the figure show what is part of the product. The darker areas represent DocSS. The dashed outline indicates how the product is packaged.

The IRM folder manager is a document manager that emulates a folder filing system; its programming interface models the connections, descriptors, and integrity rules described by joint study partners; its graphical interface presents a Common User Access* (CUA)-compliant emulation of manila filing folders and file cabinets with suggestive icons for spreadsheet, for image, and for document pages, among others. Editors for some kinds of data objects (images, text and word processor files, spreadsheets, etc.) are invoked automatically when a user "clicks" with a mouse on an associated page icon.

The smallest possible installation combines the IRM workstation components with a library catalog server and a blob server all running in a single workstation. This configuration is useful for application preparation and test, particularly for iterative design involving end users. Production environments are likely to be much larger; one customer is deploying an unemployment insurance application with several workstations in each of 60 statewide offices and a single library catalog held in the state capital.

Much of what motivates the internal design of DocSS—concurrent execution of operations initiated by users competing for the same data, communication protocols that are economical for moving large objects over wide area networks, combination of data from many libraries, delayed availability of remote servers, protection of sensitive information—is encountered only when large-scale deployment is considered. DocSS features that otherwise might seem unnecessarily

Figure 3 A document service network



complex are essential for scaling to very large environments. These features include the ability to activate and deactivate multiple concurrent sessions with library servers, to cache objects arriving from and destined for object stores, to combine commands into packaged requests, to defer work for delayed execution, to enqueue work for other people and for background services, and, more generally, to decouple the pace of application execution from that of library servers.

A library and document model

Library service can be viewed as the managed flow of information within a distributed storage network. The circumstances and statistics described earlier force stores with four roles: library catalogs, blob stores, nearby caches, and the workspace of the application (Figure 3). Library catalogs and blob stores are owned and controlled by library custodians, and caches and application stores are owned and controlled by end users. DocSS limits data flow among stores to enforce clear, explicit rules that define what is meant by data integrity and security and hides platform differences in a heterogeneous computer network.

Stores may or may not be co-located. Typically, each of many libraries, each one consisting of a

catalog and several blob stores, holds many objects for long periods and is accessible to many users. Each of many caches holds a smaller number of objects for much shorter periods and is accessible only to a few closely associated users. Each store may itself be distributed over several machines; for a library catalog this might be done with a distributed relational database.[12] As a practical example, descriptions of California bridges could be split among regional data centers (San Diego, Sacramento, San Francisco, etc.) but still give each user the illusion of a statewide catalog of bridges.

The conceptual structure of the storage network is sketched in the following sections. For precision, we need to establish for the reader key terms of reference more carefully than has been done so far.

Classes of storage and storage contents. Addressability is needed at several levels: to entire stores, to store portions called substores, and to individual objects within stores. A *store* is a place where data are held (or the data held in that place) and for which the operating system or network, or both, provide addressability conforming to well-known, standard schemes. Examples are workstations in a local area network (LAN) environment and independently-administered Structured Query Language (SQL) databases. Stores are administered by operating system components that are insensitive to the structure of contents. A *substore* is a store portion that holds logically related objects. For example, a library catalog (a substore) is a set of tables (objects) within a database (a store); a database can house several library catalogs in addition to other tables.

A *blob* (binary large object) is a finite sequence of bits—what has been loosely called an object; "blob" is used to imply that the internal structure is unimportant for the discussion of the moment. Blobs are the units of data transfer among stores. The *collection* part of a library is a set of blobs.

An *item* is a structure of blobs that represent closely related things, such as the pages of a book. It is the smallest collection that can be fully described in a library catalog; for instance, users can attach named attributes to items but not to item parts. An item is also the smallest collection of information that library services independently control for security or other administrative purposes. The term *item part*, a synonym for blob, is used to suggest the relationship of a blob to an item. An item part could correspond to a page in a document manager's model of books. An item can be without parts; a part can be without content or have content of zero length.

A *library* is a named collection of items together with descriptive catalog entries. Each item has a unique identifier and is accompanied by certain obligatory catalog fields.

A library differs from a *cache* in its proximity, availability, accessibility, data administration methodology, and support for finding information. Libraries may be distant and relatively expensive with which to communicate. Caches are typically close and available whenever wanted. Table 1 summarizes important differences.

Item descriptions—the library catalog. Items can be named, labeled, and described by text of arbitrary length. *Item identifiers* are chosen by library services and have no mnemonic value and no operations apart from the identity test. *Names*, usually chosen by human users, can be ambiguous and are therefore distinct from identifiers. *Labels* chosen by users are enforced to be unique within a library; they are useful for applications such as insurance policy numbers. Names, labels, and descriptors are commonly used as search indices.

Each item has a *container* attribute intended to model the notion that it is a part of something larger, as a steering wheel is part of an automobile. This container relationship is the most efficient method for modeling file cabinets holding folders that themselves contain documents. An item is not allowed to contain itself, even indirectly.

If limiting every item to exactly one container is not what the application at hand wants, other folder models are readily implemented. Any query that returns a set of item identifiers effectively creates a *virtual folder* that can be presented to users in the same ways as folder objects modeled by the container attribute.

The catalog can link objects. A *link* relationship is similar to the notion of attaching one end of a string by thumbtack in one document page and the other end in another page. Each link is tagged

Table 1 Differences between caches and libraries

Characteristics	Cache	Library
Size	Zero to one hundred thousand blobs	Ten thousand to one billion item parts
Availability	Needed whenever document management services wanted	Intermittent availability is acceptable
Content lifetime	Typically minutes to months	Typically years to decades
Accessibility	To a single user, or to collaborating users	To anyone permitted by a library administrator
Data retention	Semiautomatic, with old blobs being discarded	Protected by automatic and manual procedures to prevent data loss
Confidentiality of contents	Log on and cryptographic security	Access control for individual objects and actions
Integrity of contents	Work group members can disrupt colleagues' data	Every change is checked for authorization and consistency
Search tools	Simple table of contents indexed by blob name	Relations that can be joined to external relations by SQL queries

with a *link type* and may be bound to a *link description* item. For instance, if the first item describes a man and the second item a woman, the link type might be marriage and the link description the marriage contract. Linking is particularly useful for interrelating text and pictures, such as geographic maps.

Each item can have any number of *properties*; each is recorded as a *property type/property value* pair, such as color/purple. A nonobvious use of a property value entry is to relate the image of a business form to all instances of filled-in forms of that type, as might be needed to support suppressing boilerplate. For example, in a tax application with many items of type tax1040, the property type and value of the image of the blank form might be template and tax1040.

Each item has an obligatory property called its *semantic type*, indicating a kind of thing—memo, picture, contract, or the like. Each item part has a *representation type* classifying either how the information is encoded or which version of a thing it is. Representation type as encoding complements semantic type. For instance, a photo (semantic type) could be represented in 600-pel four-bit grey scale, 300-pel three-bit grey scale, or 50-pel half-tone. Fingerprints could be represented either as 10 print (pictures of the ink impressions) or minutiae (standard encoding of loops and whorls).

Item part transformations and versions. Relationships among object versions are complex intrinsically and because there are few broadly accepted models for such relationships. One kind of version is a subset of the base information, selected to be seen by a specific kind of individual; for instance, a design engineer would want to show different aspects of some work to a manager, a customer, a patent attorney, and a product test engineer. Another kind of version is a product variation for a submarket. Yet another is one of several tentative designs. Such versions are modifications of some base instance—modifications made by humans for purposes that are usually only incompletely recorded.

Other versions correspond to algorithmic transformations: geometric projections, text formatting, graphic rendering, scientific visualizations of physical models, data compression or encryption, and so on. A simple case is selection of a contiguous bit sequence from a blob; for text objects this is called *partial document access* and might be used to return one page of a document stored as a single blob.

A version can be created by a *transformation* step. Sometimes this process can be defined by a program; often it involves human steps that are inconvenient or impossible to reduce to explicit expressions. Sometimes a user wishes to store some new object and simply declare that it is a transform of an existing object, without any computing system control that the claim is valid.

Library access control. A library *custodian* is a person or organization offering a library service and committing to users the integrity and security of library-held data. A *library administrator* is an agent of the custodian and is responsible for admitting patrons to the library, defining their privileges, and otherwise administering basic access control tables.

Security is conformance to proper authorizations for the movement of data out of one store into another and for changes made in one store responsive to instructions originating in another store. Library users will be most concerned about how workstations interact with libraries and secondarily with what applications can do to caches. *Access control* is the security component for defining who may do what and administering such rules. Other security components enforce the rules and create an audit trail for compliance checking.

Access control and security depend on identifying patrons and items uniquely. The library catalog associates a single patron with each item as its *owner*. Conceptually, a patron is a potential user defined by a library administrator. Formally, a *patron* is a set of permissions to store into, search in, or retrieve from a specific library, or combinations of these activities.

Ownership of an item implies other privileges, including alteration of its access list. However, even an owner can be selectively blocked. For instance, a library administrator can block all updates to primary object instances, i.e., ensure that a library is read-only in the sense that no item may be altered after it has been stored.

Document storage subsystem design

The Document Storage Subsystem is an answer to the question, "Given existing and emerging file, database, and communication systems, what is the least amount of software needed for a digital analog to conventional library services and for managing the electronic equivalent of paper as it flows into and from each user's workspace?"

Ideally, each user would have prompt library access, without interference from the other activity. The service interface would be simple, with the information sought in each library request available before the user wants further interaction. Although this is sometimes possible, practical and economical considerations often impose less favorable circumstances:

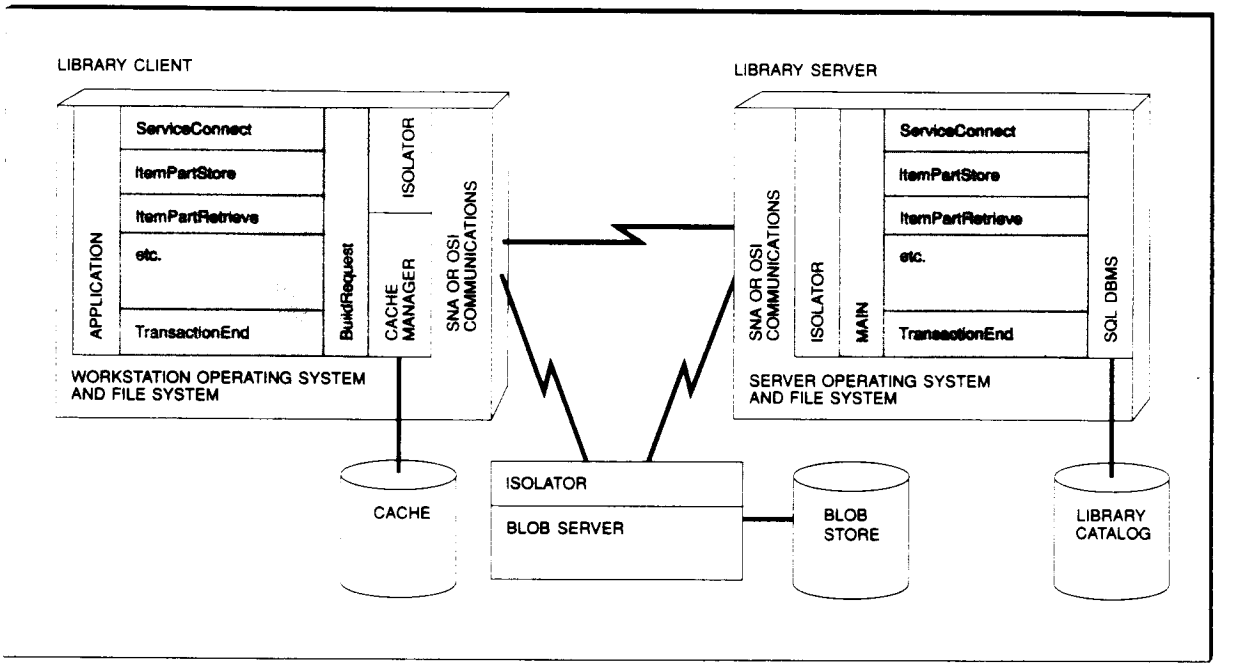
- Data requested may be too large for delivery while human users wait, or may not be promptly accessible, e.g., if held on tape volumes.
- Library service connections may be disrupted; this possibility should not impede users from submitting requests or using library data already collected into their workstations.
- Massive data input, such as scanning all of the birth, death, and marriage records of California, must be executed with sustained throughput but without interfering with information inquiry.

Fortunately many applications do not need a prompt response provided that the users are not impeded in their collateral tasks. DocSS provides enqueued as well as interactive service, exploiting workstation and server multitasking to buffer interactions and to create a storage hierarchy that holds data for repeated use in the workstation.

A single library service instance. A library service instance consists of cooperating processes in a workstation, a library catalog machine, and a blob storage machine (Figure 4). The 50 or so elements of the application programming interface can be classified into cache manipulation, library catalog query, library catalog update, and blob movement primitives.

Each library catalog is implemented within a relational database. The query language is SQL, enabling all queries permitted by the database, including queries that range beyond the current library catalog. For instance, a highway department might have an independent database relating maintenance contractor invoices to bridge numbers; a query join would permit a search for "inspection reports of bridges for which the XYZ Company delivered concrete." Each library operation visible to applications (ServiceConnect, ItemPartStore, etc.) is effectively a single program

Figure 4 Structure of a library service instance



running partly in the client environment and partly in a library server. In Figure 4, a server portion on the right implements a primitive library operation for each application programming interface on the left. The client portion does those input validity checks that are possible without remote data and translates its inputs to a standard form. The server portion repeats and extends the validity checks before making persistent changes or returning library data to the client. When network connections are unavailable or slow, degraded service can continue with the cache content.

Every item is uniquely identified within its library; no identifier is ever reused for a new item. Since every library in the world also has a unique name, no item can ever be confused with any other item.

Privilege to library data is granted at three levels: to each library, to each operation or closely related set of operations on each library, and to each item within a library. For instance, a patron who wants to discard something needs permission to use the containing library, permission to use ItemDiscard in that library, and discard permission on the item in question.

Different communication methods are hidden from other programs by Isolator modules. Given a library identifier, the client Isolator chooses a path to a library server and communication protocols (Systems Network Architecture LU 6.2, Transmission Control Protocol/Internet Protocol, NetBIOS, etc.). The same Isolator code runs in clients and servers, except in MVS-based servers, where part of the function is provided by CICS.[13] The special cases in which the client and server are in the same machine or on the same LAN are detected in the isolator, which chooses an efficient communication protocol, e.g., OS/2 queues if both processes are in a single PS/2. Thus communication between coresident processes is almost as efficient as direct calls would have been.

The computers and operating systems in a library network can be of different types; the data conversion needed is the same as that for electronic mail and distributed database management systems (DBMS), viz., EBCDIC-ASCII conversions and differing representations of integers. If the ma-

chine type and operating system into which an object is retrieved is the same as that where it originated, it is bit-wise identical to what was stored. Otherwise, the data conversion needed is identical to that encountered when a file is moved directly from the source environment to the target environment.

The client-server split naturally protects against workstation programmer improprieties.

Blobs can be stored in the library catalog, in files on the machine housing the catalog, or somewhere else entirely. DocSS does not hide the location of where any particular blob is stored, but does not allow application programs direct control of where and when blobs are placed or moved among library stores. Instead, applications can hint at what treatment will balance economy and performance, according to the concepts of system-managed storage.[14] Blob servers are file servers, except that blobs move on different paths than commands.

Multitasking in remote service delivery. A *library service* is a set of processes that mediate access to one or more libraries. A *library session* is a collaboration between an application agent executing in a workstation—the client process, a library server that executes a single execution thread in a catalog machine, and blob servers that deliver or accept data from blob stores wherever they may be in the network.

In each library catalog machine, a single process with a published address helps applications connect to the libraries in that environment; the other processes there are library servers (Figure 5). Each user might have several open library sessions; each library might have several blob stores and use more than one within any session. Each *library server* controls access to one library at a time. A *requester* is a (workstation) application process that makes demands on one or more libraries—demands mediated by clients. A *library client* is a process that acts on behalf of a patron and exploits a library server. A workstation may concurrently execute many requesters; each requester may have several active clients.

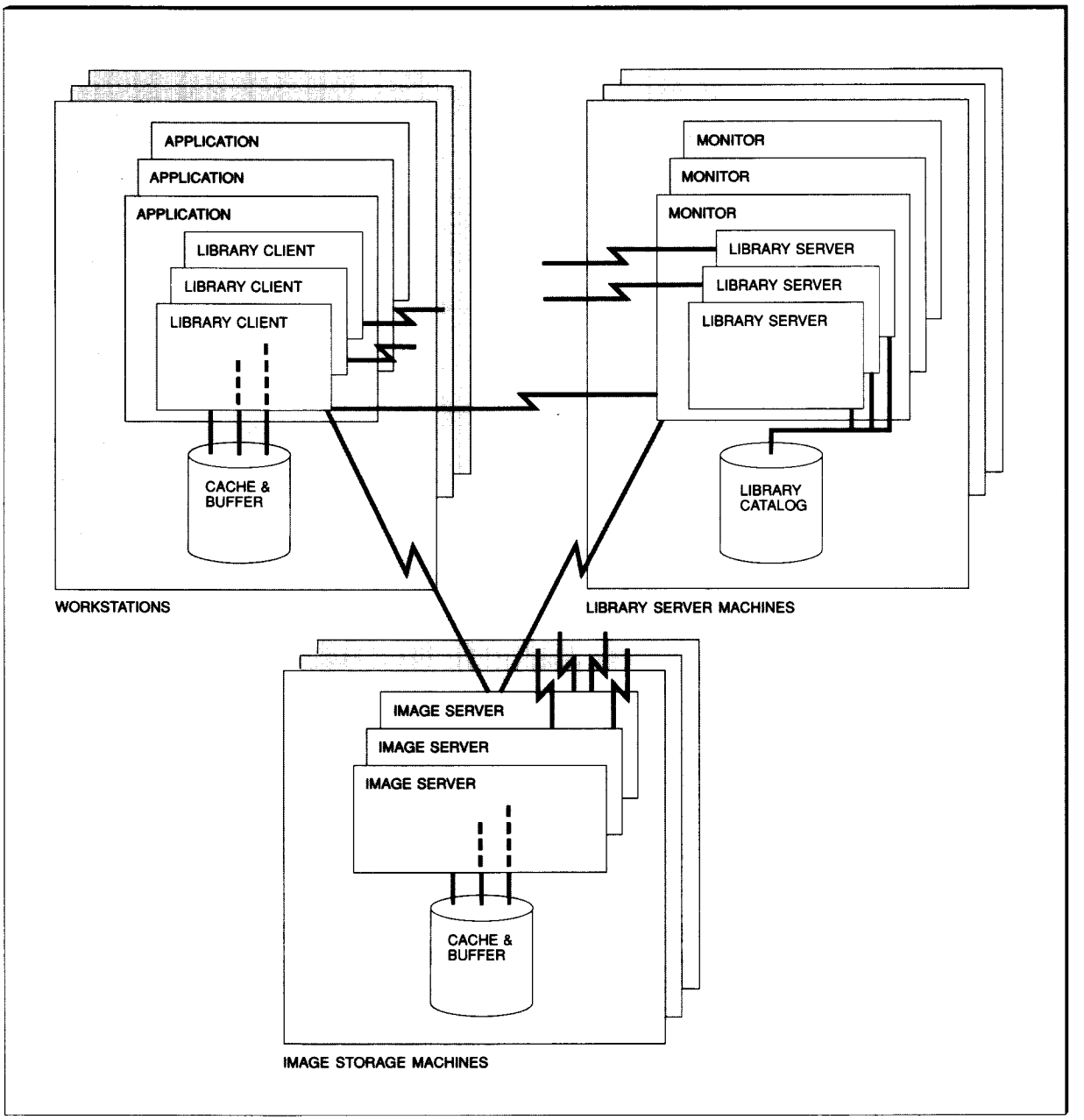
Workstations may be installed wherever users want them and therefore are not assumed to be secured by physical or administrative measures. Library server machines control the integrity of the data that they catalog. Images are moved directly between workstations and blob stores. Each blob store accepts commands exclusively from library servers associated with a single library; the database that houses each library catalog is the sole point of control for that library. Protection against information theft, unauthorized changes to library contents, and disruption by playback attacks is achieved by well-known means. Encryption is available during object storage or transmission, or both, if other protection is deemed insufficient.

The client-server split naturally protects against workstation programmer improprieties. Library security as good as that of the server machines is achieved without any new administrative tasks. Server programs must be installed by a trusted administrator, who must guard against "Trojan horses." Apart from this risk, the possible library damage caused by a deliberate invader is limited to the data for which the invader has valid or purloined passwords. Improper changes to any workstation program can at most damage the service at that workstation. How security is achieved has been described and analyzed elsewhere.[15]

DocSS manages changes as *atomic units of work* or *transactions*. Each library session is treated as an independent application. The concepts and behavior are identical to SQL transaction management, which is described in any basic text on database management systems.[16] To help applications avoid overwriting each other's updates, each library includes a check-out/check-in[17] registry.

Buffering traffic to and from libraries. Before an application can manipulate items or their descriptors, it must call SessionStart to establish a library session. The SessionStart call includes a *priority* relative to other sessions competing for scarce resources, such as long-haul communications bandwidth, and a *style* to indicate the format of error and information messages. It returns a to-

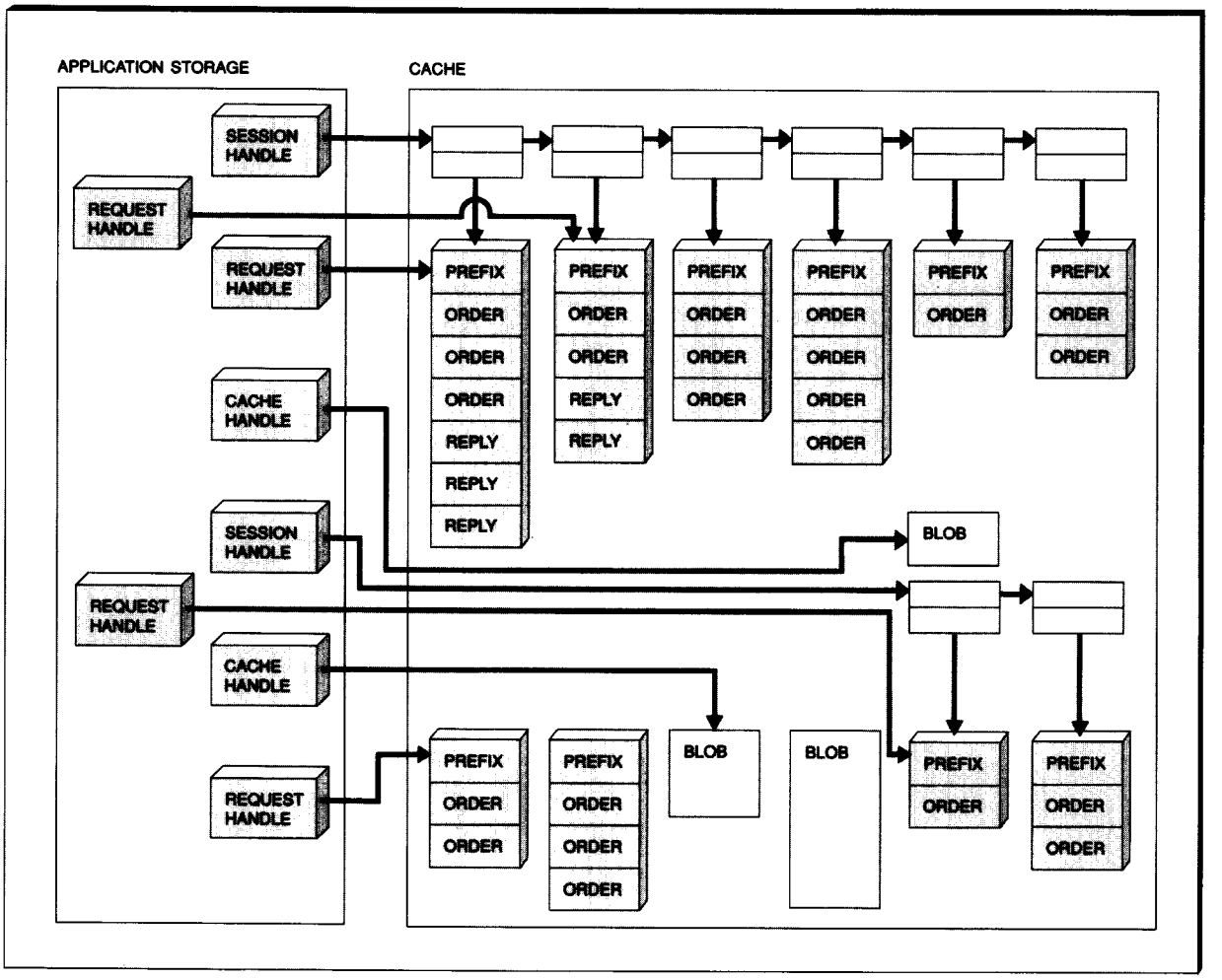
Figure 5 Multitasking in a library network



ken called a *session identifier*; the application must use such a token to direct subsequent library requests to one of a multiplicity of sessions it may have established.

Library service calls create primitive library orders: move one blob, change one catalog record, etc. Each order could be transmitted as a separate network message, but doing so would often be

Figure 6 Access to library requests, responses, and blobs in a cache



unnecessarily costly and slow. Instead, library clients accumulate orders into batches called requests and buffer these requests to permit applications to run ahead of their library sessions.

The client part of a library session is a request list and a response set. Each element of the request list is a sequence of orders for a library server. Each request list consists of any number of ready requests followed by a single incomplete request (Figure 6). Request lists and individual requests are anchored by application-held handles. Library service calls append orders to the incomplete request. Each response is a sequence of replies—one for each order in the corresponding request. Whenever the application chooses, it calls on RequestEnd to complete the open request and return a request handle for access to the eventual response. RequestEnd also implicitly creates a new request consisting of a prefix that identifies the target library and patron and that carries addressing and authentication tokens.

ServiceConnect creates a *client daemon* process (Figure 7) that promptly starts submitting requests. This daemon removes and deals with the first list element, repeating this procedure whenever there are at least two list elements; the last request in the list is always incomplete. In anticipation that the cache itself may be remote in fu-

ture implementations, client stubs communicate with the cache managers and client daemons via isolators; for local service, the implied overhead is low. In contrast, interactions of each client daemon with a library server and a blob server on the channels depicted in Figure 5 are synchronous and serial. Thus execution is asynchronous, with the request list buffering any pace difference between the application and the remote service. The application can make it synchronous by waiting for each response before issuing the next request.

An application can ask for a background process by specifying a ServiceConnect delay for start of service. The daemon created by such a call takes ownership of the current list, leaving the incomplete tail request as the beginning of a new request list. After the prescribed delay it deals with the list, terminating itself when done.

Responses and blobs are returned from library and blob servers to the cache. The application may use ResponseGet to bind a response using a handle returned by RequestEnd, and may use BlobOpen, which exploits a cache directory, to locate blobs.

Storing and retrieving blobs. Storing a blob into a library is managed as a cascade of client-server interactions already suggested by the triangular configuration in Figure 4: a client daemon (Figure 7) acts as a library client; the library catalog server acts as a blob server client; the blob server acts as a client of a cache management task in the workstation. The sequence of events (Figure 8) is:

1. The client prepares its blob transmission port for a read and then sends the library server a request that includes the address and extent of the blob and a proposed blob identifier, id. The client then waits on its blob port.
2. The library server checks request validity and id uniqueness and chooses a blob server to which it sends a command containing id, the address of the client's blob port, the blob description, and a hint about where the blob should be stored.
3. The blob server sends to the client's blob port a request for the data.
4. The client compares an embedded authentication token to its copy; if they match, it sends the requested data.
5. After the blob server has stored the data reliably, it signals the library server.

Figure 7 Cache and client daemon to implement asynchrony

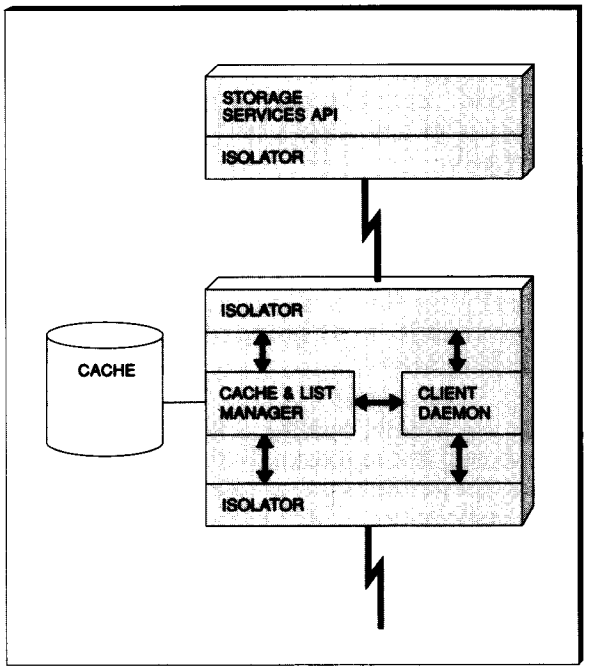
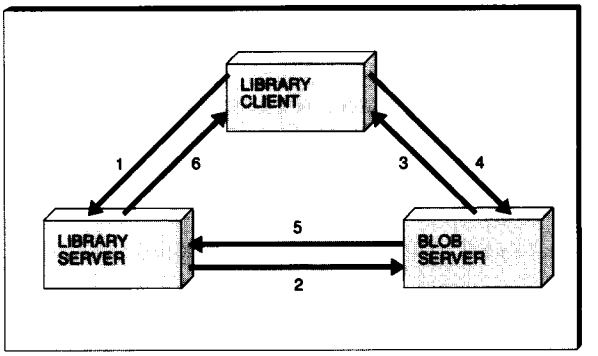


Figure 8 Message order for ItemPartStore and ItemPartRetrieve, equivalent to three client-server relationships



6. If the library server receives a timely positive acknowledgment, it updates the library catalog and signals overall success or failure to the client.

Retrieving a blob from a library to a client is similar, except that the blob server and the client

move the blob in the opposite direction. Discarding a blob is different:

1. The client sends a discard request to the library server.
2. The library server checks whether the request is valid, deletes the catalog record of the blob, and builds and saves a blob server request to discard.
3. At some later time in the same library session, the library server executes the client's request that its library changes be committed as an atomic unit of work.
4. Finally the library server causes each saved discard order to be executed by the appropriate blob server.

Replacing a blob, moving a blob between two blob stores, and replicating a blob in several blob stores (for performance or safety) are somewhat more complicated. These blob movement protocols have been described in more detail, together with demonstrations that they preserve the consistency of library catalog to blob store, that they can be protected against the expected kinds of security violations, and that they are the most economical safe alternatives for wide area networks.[15] This is under the condition that sessions intermix queries, retrievals, and library updates in an order that cannot be anticipated; for read-only sessions, improved average performance is possible by ordering retrieval requests to reduce optical jukebox platter mounts.[18]

Having the blob server act as a client relative to the workstation rather than as a server not only is an economical route to data security and integrity but also simplifies its internal catalog and administration compared to what would otherwise be needed. Blob servers do not need information about end users or about workstation addresses.

Managing transformations. Transformations represented by completely automatic programs (often called *filters*) could be managed entirely by DocSS. Transformations needing human assistance (e.g., for deciding unresolved characters in optical character recognition) must be executed at least partly in a workstation. This execution may be at a different time and in a different workstation than where the input originates; for instance, it will sometimes be effective to scan an entire collection of papers and to apply semiautomatic recognition routines only to subsets selected long after the scanning step.

Completely automatic transformations may be executed when the base information is stored, at times chosen by the storage subsystem, or during retrieval. Early generation may be desirable if it requires a great deal of processing, as in the case of document analysis and optical character recognition, or to achieve the best retrieval performance. For instance, calculations such as reduction of fingerprint images to encoded form (minutiae) are best done when processors would otherwise be unloaded.

Transformations represented by completely automatic programs could be managed entirely by DocSS.

Blob servers have been described as returning strings identical to the strings stored. More flexibility is possible. Projections of tables or pictures can be done at viewing time. For example, partial document access is a simple filter to avoid transmitting and caching unwanted data. The efficient implementation of delivery filters puts them in blob servers, with the library server passing transformation orders from the library client to the blob server. Being bound as server extensions, the transformation programs are inaccessible to user inspection or tampering—an effective protection for proprietary or otherwise sensitive programs.

Programming Document Storage Subsystem applications

The reader can deduce much of the programming interface from what precedes and from the observation that DocSS primitives are intended for construction of generic document managers. In this section a summary of a catalog and call subset emphasizing less obvious aspects is given. Full

Table 2 Some views defined in library catalog

ITEMS	Basic item attributes, with one tuple per item; the fields include: • The identity of another item—the container • Ownership and type identifiers • A user-chosen unique label, e.g., for an insurance policy number • Creation, reference, and planned expiration time stamps • Security classifications
ITEMPARTS	Part attributes, with one tuple per item part; the fields include: • Representation type and part number • Storage location and storage hierarchy control flags • A last-changed time stamp • Transformation parameters giving the provenance of versions
GRAVEYARD	Attributes of discarded items, describing each as in the ITEMS table
REPLICAS	Locators of duplicate item parts held to improve performance or availability
NAMES	Human-legible item names bound to particular patrons
DESCRIPTIONS	Text descriptions of items in free format and of arbitrary length
PROPERTIES	Attribute type/attribute values for items, e.g., color/purple
LINKS	Links of a directed graph among points in documents
...	...

details on the implemented subset can be found in the IRM programmers' guide.[1]

Summary of the library catalog. A library catalog is represented by tables that can be extended by service offerers. These tables store attributes concisely, using some formats inconvenient for human users. The catalog defines views that show legible mappings and that filter the data to implement read access constraints. DocSS functionality is defined in terms of these views, some of which are given in Table 2.

Not listed are tables for event logging and for access control.

A library can catalog and describe items it does not hold, such as external files, physical objects, and items in other libraries. Of course, such external items can be moved or removed without these changes being correctly reflected in the catalog.

Summary of the procedure interface. The DocSS client portion uses the cache (Figure 1) to buffer control information and blobs prepared for sending to libraries and to hold replies and blobs received. Caches are also available for other work, e.g., queues for work flow applications. An application can attach to several caches concurrently and to several work lists within each cache. An implementation can share caches so that several machines on a LAN can share retrieved objects. Applications obtain addressability to cached objects as handles to lists, blobs, etc. (Figure 6). Cache subroutine calls as listed in Table 3 cause no interactions with catalog servers or blob servers.

Except for RequestEnd, each library service call (see Table 4) prepares a primitive server order and appends it to the incomplete request at the queue tail. RequestEnd completes the open request and appends a new request stub to the queue (Figure 6). Request boundaries and atomic unit of work boundaries may be interspersed with other commands in any order, permitting any blend of interactivity with batching.

Library administration can be done with workstation application programs. Since permission to use each operator is separately granted by the library custodian, the distinction between an administrator and an ordinary patron is simply that an administrator is granted the use of privileged operators.

Table 3 Some cache subroutine calls

SessionStart	Establishes a client-server relationship with a cache instance
WorkQueueCreate	Establishes a library work queue
ServiceChoose	Associates a library work queue with a particular patron and library
ServiceConnect	Starts up a client daemon (Figure 7) to manage what is asked for in a library work queue
BlobCreate	Allocates cache space for a new blob, returning its handle and defining the cache retention wanted
BlobOpen	Binds a blob to an application, with a copy in application storage for editing
BlobSave	Saves an edit copy of a blob in the cache, replacing the old version
...	...

A model document manager—folders and file cabinets. Document storage subsystem services neither interpret object contents nor require particular descriptive attributes beyond some basics, e.g., time stamps, object type indicators, and owner identifiers. Practical applications require data models and enforced compliance to those models. A document manager can help users conform to common practices and rules in their enterprise, or to public standards. The IRM folder manager creates a digital analog of filing folders held in file cabinets; the IBM ImagePlus Folder Application Facility[9] is similar.

In descriptions of the document storage subsystem, we use neutral terms such as *item*, *blob*, and *part* to avoid implying any particular data model. In what follows, it is quite appropriate for the reader to think in terms of an analogy to paper, so we can use more connotative terms, such as *document*, *image*, and *page* without creating false expectations. These words, and also *folder* and *file cabinet*, are the terms of reference for the description of a folder manager.

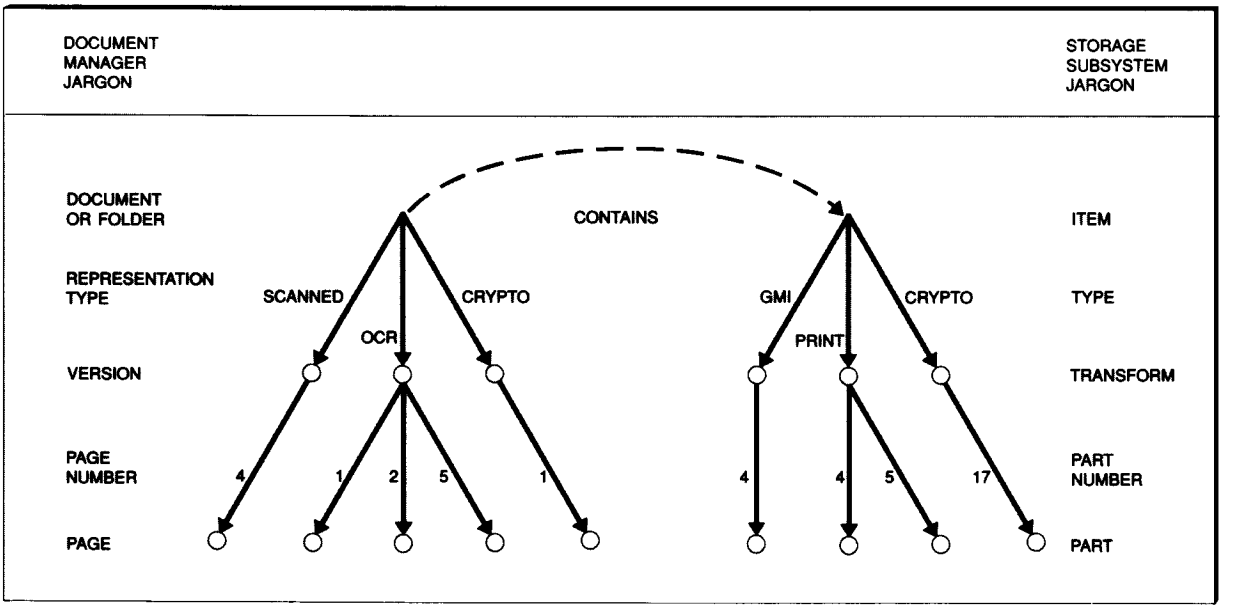
A *document* is a sequence of images, such as (a digital representation of) the pages of a book. A *folder* is simply a document that *contains* other items by reference. Folders are unordered groupings of documents; ordering can be induced by SQL queries.[19] Every document except one occurs in exactly one folder; the folder relationship is acyclic. In addition to a system-assigned unique identifier, each folder or document has an application-assigned *label* unique within its library. A *file cabinet* is a collection of folders. File cabinets are related to folders precisely as folders are related to documents.

A document *version* is one of a set of representations of more or less the same information. Examples are a formatted version of marked-up text,

Table 4 Examples of library service calls

RequestEnd	Enqueues collected orders as a library request and starts a new request
TransactionEnd	Completes a library atomic unit of work
LibCatalogQuery	Sends an SQL query to library; just as for other orders, the query result comes back as part of a reply
ItemCreate	Creates a new, empty item in a library
ItemPartStore	Copies a cached blob into a library item part, creating a new part if necessary
ItemPartMove	Moves or copies a library item part, creating a new item part
ItemPartRetrieve	Retrieves a copy of a library item part into a cache
...	...
ItemLink	Builds or discards a LINKS table entry
TransformBuild	Creates a version from stored item parts and stores it as a new library item part, together with information on how it was constructed
TransformRetrieve	Creates a version from stored item parts and stores it as a cached blob
AccessRuleSet	Changes the access control list for a set of items

Figure 9 Nomenclature for the model of stored objects



a low-resolution derivative of a portrait, and a two-dimensional projection of a three-dimensional design. Folder manager applications cannot always determine whether an image is stored or generated for the occasion. A *page* is a member of the sequence that makes up a version. Allowing each document or item to consist of versions or transforms that themselves consist of pages or parts is more structure than is needed in principle. However, it corresponds so closely to real-world situations, such as editions of books, that it is often useful.

The mapping of the concepts of this particular folder manager to the structures of the document storage subsystem is illustrated in Figure 9. The diagram graphs an object which *contains* a second object and suggests a particularly simple document manager whose accessible entities map one-to-one to storage subsystem entities. Types reflect the DocSS concept without change. The type of each document, called a *semantic type*, indicates a human purpose, e.g., memorandum, purchase order, employee photograph, program code, etc., and consequently a schema for internal structure. The type of each version, called a *representation type*, indicates an encoding method; this might be a markup language (for revisable text), a compression scheme (for uncoded information such as image), or a programming language such as COBOL.

An OS/2 Presentation Manager interface is closely coupled to the IRM folder manager and shows icons suggestive of printers, scanners, file cabinets, folders, etc. Page icons differentiate spreadsheet, word processor, and image files. Selecting any kind of object invokes the right kind of editor for that object, and pull-down menus are a path to other appropriate operators. Such an interface is easily learned by novice users.

Simple folder models just described work well for topics whose individual subjects are human, such as welfare, taxation, insurance, and education applications. They are often insufficient for engineering, social, or cultural topics.

Discussion and futures

This paper articulates a stage in a cycle of designs, prototypes, and discarded code. For library services, this is the fourth iteration since 1988. For cache services, it is the third iteration since 1989. Each design iteration was critiqued by joint study partners; each prototype was only part

of what was envisioned at the time but taught much that was later incorporated. Part of what is yet to be addressed is shared below.

Extensibility of DocSS. The library service primitive routines—ServiceConnect, ItemPartStore, etc. in Figure 4—are relatively simple compared to the rest of DocSS. The client portion converts inputs to the format expected by the library (e.g., we might add a COBOL interface to the current C-language one) and checks input as much as possible without a library catalog connection. The server portion rechecks and further validates inputs before executing orders. The logic that distinguishes library service from other distributed data services is almost entirely in these routines, i.e., to extend DocSS functionality, one must add such primitive routines.

Many administrative tasks can be implemented as ordinary DocSS applications executing concurrently with end-user applications.

The colored portions (in Figure 4) of library clients have no I/O or other operating system service calls; their code is portable from OS/2 to UNIX**-family environments. The colored server portions have no system or I/O interfaces apart from embedded SQL. This is ANSI-SQL (American National Standards Institute) except for database connection services (which are not covered in the ANSI definition). Thus it is possible to program portable versions of these routines; portability—especially the server portions—helps make library semantics identical on all platforms (OS/2, Advanced Interactive Executive*, or AIX*, MVS, etc.).

DocSS is positioned to use OSF/DCE naming[20,21] and authentication[22] services when suitable implementations become available. Only the Isolator needs to be modified, replacing its table-based subroutines by network calls.

The data model has limited each image collection to being accessible by way of a single library catalog—that of the library that owns the contents of the image server in question. This restriction can be relaxed to permit retrieve-only access by way of a nonowning library that has cross-indexed some items. Catalog entries for objects in external libraries cannot, of course, be guaranteed to be correct. (This weakness is apparent also in libraries based on paper.)

The following subsections deal with a few kinds of extension in more detail than is possible for others—means for tuning and tailoring library service, contextual search, improved access control, versions, and multimedia.

Library administration, tuning, and tailoring. Many administrative tasks can be implemented as ordinary DocSS applications executing concurrently with end-user applications. Examples are:

- A purge executor for expired items (an end user's discretionary document disposition orders might be overridden to meet statutory requirements or enterprise policy)
- A predictor that senses blob requests, uses the library catalog to predict future requests, and stages the data
- An optical jukebox layout manager to group blobs likely to be used together
- A migration utility to move data to cheaper media within or among blob stores

Such operations can run in daemon processes that schedule themselves, making load measurements and other tests to avoid impairing performance for more urgent work. Since information systems nearly always have processor cycles and channel bandwidth that would otherwise be idle and therefore wasted, this tactic can be very effective.

Individual enterprises and individual users often understand the dynamic statistics of their applications in ways that are difficult to exploit in generic software. They would like to use this insight to optimize document service performance and cost. The opportunities include policies for:

- Scheduling of deferred work, e.g., retrievals, and background sessions
- Allocating shared resources, such as communication channels

- Controlling which objects are cleared from caches early and which late
- Choosing the kind of storage for arriving objects
- Choosing whether and when a library object should be replicated and which objects an arriving object should be stored close to
- Scheduling transformations, e.g., the more expensive ones described earlier in the subsection on managing transformations
- Choosing and scheduling automatic indexing processes, such as those for finding and tabulating specialized indices (e.g., city names in a highway database). (Such automatic indexing is a specialized form of transformation that is readily implemented as a library primitive.)

These opportunities require installation exits beyond what is in the IRM product.

Contextual search. A mature system would include the ability to choose objects not only by catalog queries but also by filtering blob contents. This ability is particularly important for textual objects, for which it is called *contextual search*, but extends to esoteric filters. Ideally, an extended SQL would combine predicates on relations with fuzzy predicates on object contents and structure. Practical implementations are unlikely in the near future.

The near-term solution is a two-phase approach using inverted indices. An independent contextual search engine would act as a library client retrieving blobs of types it can handle (probably text only) to create its own index structures. A new query interface would be needed for combined filtering based on contextual indices and library catalog attributes. There is a clear opportunity to invent better ways of integrating contextual search and relational attribute search than such a stopgap measure. The solution is likely to embed search engines in blob servers.

Access control. Adequate library security can be delivered by knitting together components that either exist today in commercial operating systems or are under consideration, except that, as far as we know, no extant access control model combines everything needed.

The problem is partly one of scale: libraries may range from instances with 10 users and 100 000 stored objects to instances with 100 000 users and one billion stored objects. It is partly one of heterogeneous applications: what a public service library needs (almost nothing) is very different from what governmental oversight of toxic waste disposal might demand, and both are very different from what an aircraft manufacturer needs. A comprehensive scheme must provide at least:

- Decentralized administration of resource pools, because for large pools no single individual or department can know what controls are appropriate for everything
- Clear definition of resource pool boundaries so that service offerers such as library custodians can confidently enter (implicit) contracts to protect other people's data
- Clear definition of reference scopes; for instance, the word "public" becomes fuzzy when data may be accessed across enterprise boundaries and has a different meaning for the Resource Access Control Facility (RACF)[23] than for SQL[19]
- Smooth synthesis of mandatory and discretionary access control, as might be needed by a company with both military and commercial contracts
- Conformance to generally accepted accounting principles, which hold that each person be limited to resources needed to discharge his or her responsibilities, that user actions can be reviewed by outside auditors, and that sensitive resources are accessible only to partial steps by independent users (separation of authority as is common for money management)
- Means of constraining specially privileged users, such as security officers, auditors, and data administrators, to their proper activities, and differentiation of user roles from individuals (e.g., "payments office manager" instead of "Jane Doe")
- Proxy support, in which a human user acting for another human, or one machine process working for another, temporarily gets partial privileges of the principal
- Fine granularity for data subsets, such as controlled access to certain fields in standard forms (for instance, hiding part of adoption records in birth certificates), and field-value-dependent constraints
- The possibility of an implementation with good performance, e.g., access control should not markedly increase the number of I/O calls over what uncontrolled service uses

Work in progress suggests a comprehensive solution; a DocSS subset is being designed.

Version management. No single model of versions encompasses everything needed. Office applications can be satisfied by a simple model; engineering applications need to capture intricacies of how people collaborate. A general model would permit users to represent different releases of a data object; different aspects such as (for integrated circuits) physical design, materials design, circuit parameters, circuit transfer functions, and timings; design variations (customization); design experiments ("what if . . . ?"); and algorithmic transformations that show views.

An implementation thus faces a double challenge: whether it permits all models that might reasonably be wanted, and whether it can efficiently realize (the features of) the particular models wanted in the near future.

- DocSS can model a version history as a directed acyclic graph with a single root and with a clear distinction between *derivative* and *alternative* arcs. The essential distinction between the most recent version and the current version is made, but DocSS must be extended to support policies for "(1) which users are permitted to reset currency, (2) how many currencies may be simultaneously active within a single version history, and (3) whether currencies can move backward."[24]

- In visualization and massive statistical applications, such as geophysical modeling, scientists demand a complete lineage of derived data. This is provided in the DocSS catalog structure and transformation operations.

- Labeling item parts with type and provenance fields helps with the fact that a "variety of representations are needed to describe a design artifact"[24] and with transformations creating versions abstracting more complete information.

- "Workspaces . . . can be *archive, group*, or *private*. . . . An archive workspace is readable by all, and anyone may append to it. Special controls . . . ensure that only fully verified (i.e., released) objects are placed into it. . . . Only the owner of a (private) workspace may read or alter its contents. . . . It turns out that two kinds of workspaces are not sufficient for the CAD environment. Sometimes it is necessary to combine the in-progress work of two or more designers before it can be determined that the assembly works as required. Group workspaces are meant to support this kind of activity: any member of a specific group may access the contents of a group workspace or append to it."[24]

  DocSS supports the archive/private distinction by its library/cache dichotomy. Departmental libraries are a possible basis for group workspaces; we must consider whether shared caches or libraries implemented on workstations are better.

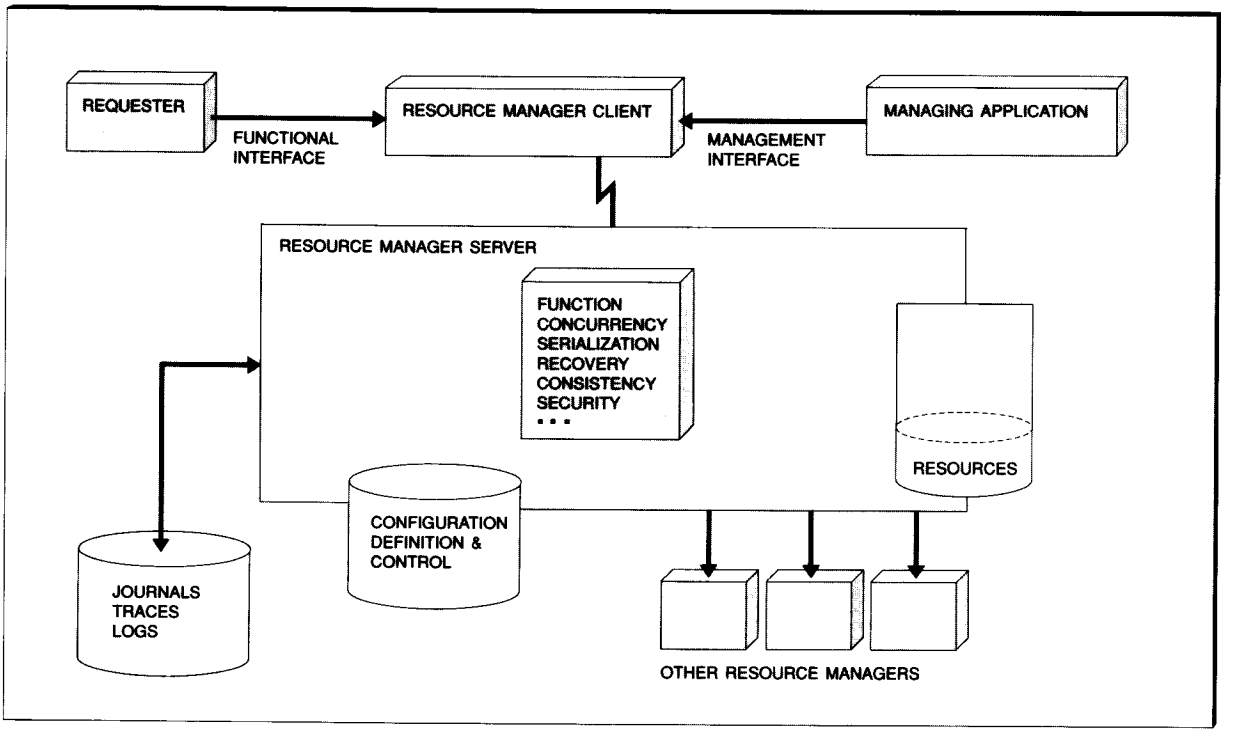
- Katz calls for propagation of changes and constraints.[24] Since propagation can be ambiguous, DocSS needs an interface with which designers can communicate their intentions for check out and replica management.

Given the extensibility described above, sophisticated version management is clearly feasible but has not been demonstrated.

Audio and video objects (multimedia libraries). It is taken for granted that video and audio libraries will require their own kinds of servers coupled loosely with digital catalogs. The blob movement protocol shown in Figure 8 has large but simple messages on the blob server to library client link, with most of the control and authentication data on the links to the catalog server. It is equivalent to three cascaded client-server relationships, with the perhaps unexpected twist that the blob server acts as a client to a workstation process. We believe that the control and catalog portions of DocSS can be extended to manage video and audio services synchronized with the demonstrated blob services.

Information capture from data. Often library applications are not intrinsically image applications but become so because of existing collections and because paper continues to enter the system from the outside—*uncaptured paper*. Also prominent is paper originating in a computer system, printed, and shared only to be needed in another system—*fugitive paper*. Information available in encoded form will nearly always be cheaper and simpler to use than images scanned from paper or microfilm. Converting raster images into encoded forms is always apt to be relatively expensive because it is likely to require human assistance if close to 100

Figure 10 What it means to be a resource manager



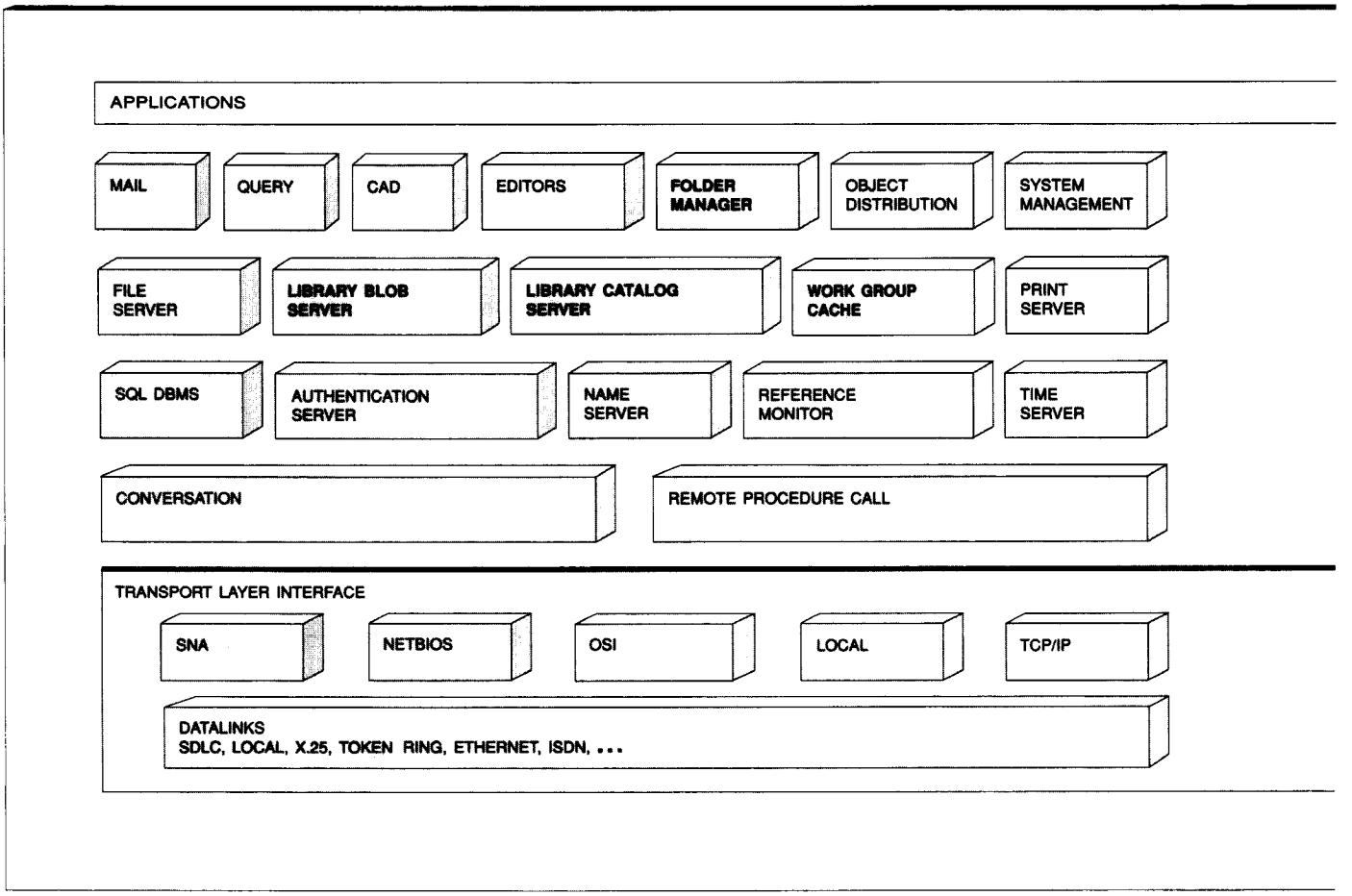
percent accuracy is wanted. In such cases, the cost of putting a page into a library may be larger than the cost of storing that page for a century.

Whatever procedural changes are instituted to avoid fugitive paper, residual information available only on paper will have to be captured. When there are large retrospective collections, the economics can be understood in readily comprehended terms. If each existing Caltrans document costs $1.00 to code and index, a modest estimate based on existing commercial methods, capture into the database will cost $50,000,000. Since this cost of capture is apparently the economic deterrent, it is surprising how little relative attention it is receiving. To realize giant libraries of administrative information, we must examine every aspect of information capture—scanning, encoding, type and structure determination, and cross-indexing—and eliminate every human step possible, without interfering with users' ability to intervene manually.

Document storage services within open systems. A library is one tool among many needed by each digital document user. Different users will use different tool mixes and different data models. Document management tools must integrate readily into existing and emerging environments. For each component—hardware such as storage volumes and software such as operating systems, image processing subroutines, etc.—there are many alternatives. Any specific application will require a mixture likely to be different from what is installed elsewhere and likely to change over time. Internally, DocSS is a modular toolbox that is partly implemented in the IRM product. Over time, we hope to extend this toolbox to the most important machine and operating system environments.

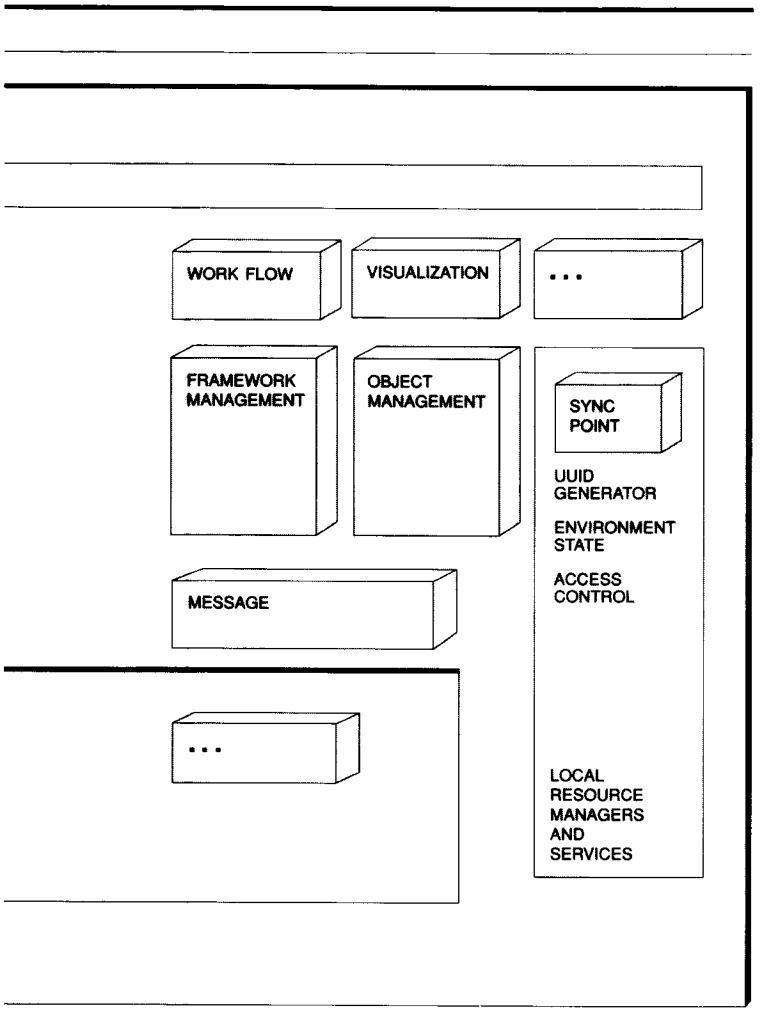
An industry direction emphasizes a network of mutually supportive resource managers. Each *resource manager* instance (Figure 10) combines state and processes and is accessible to remote concurrent clients, and may act as a client for services it itself needs. Resource managers avoid favoring one application class over another by particular data structures and typically depend on

Figure 11 Document storage and folder services as resource managers



one another; for instance, a database management subsystem draws on naming and authentication servers, which themselves may use database managers in their implementation. The DocSS library catalog service, blob service, and cache service may each be seen to be a resource manager in the sense suggested by the figure. Together with document managers, they fit into a pattern in which application enablers define data models that they implement by drawing on resource managers (Figure 11). A folder manager such as that sketched and previously described in the subsection on a model document manager belongs in the second row—application enablers that obtain remote services only by way of resource managers. The second row in Figure 11 depicts application enablers, and the third and fourth rows depict resource managers; the heavy line two-thirds of the way from the top depicts what is called a *Transport Layer Protocol Boundary* (TLPB). This paper describes components in the categories shown with bold font.

Electronic library projects for knowledge workers. A dream that motivates the work reported is ready access, from both the office workstation and the home computer to all the information alluded to. In addition to basic computing, storage, and communications support, parts of the technology needed are being pursued in many digital library projects in both commercial and academic circles. Commercial projects tend to emphasize business and quality controls, cost control, and improved responsiveness to ultimate clients. Ac-



ademic and professional society projects tend to focus on the dream.

Each project necessarily concentrates on aspects essential to creating a service within two to three years. The points of emphasis are more complementary than competitive. A National Library of Medicine project[25] showed that digital replacement of existing books is not economical, at least not yet.[26] As a consequence, academic and cultural library projects are concentrating on making unique and fragile materials broadly accessible and on enabling searches over archives too large for public shelves.

- An IBM joint study with the Japanese Museum of Ethnology allowed a search into a collection of color pictures of Pacific Island artifacts.[27] The implementation environment is a LAN-based database on Kanji-enabled PS/2 workstations.
- In another IBM joint study with the Spanish Ministry of Culture and the Ramon Areces Foundation, the 16th century papers of the *Archivo General de Indias*[28] are being digitized. By the celebration of the 500th anniversary of Columbus' first voyage to America, about eight million pages, representing about 10 percent of the collection, had been captured. The project uses about 60 PS/2 workstations and an Application System/400* (AS/400*).
- A third IBM joint study with the Brandywine Museum captured and cataloged high-quality renditions of Andrew Wyeth's work while the artist was still available to guide the project.[29] The project emphasized quality of color reproduction. The machinery is a network of PS/2s.
- In 1990, the U.S. Library of Congress started a project called "American Memory"; it intends to deliver on-line collections of unique value to libraries throughout the United States. The first collection comprises about 30 000 80-year-old photographs of Detroit. The distribution medium is optical disks to be shown on personal computers in one or two libraries in each state.
- The University of California, Berkeley, image database project "demonstrates the feasibility of on-line access to digital images of maps, slides, paintings, photographs, rare manuscripts, museum artifacts, botanical specimens, and other visual materials."[30] The eventual target is access to about 50 departmental collections from a campus-wide network. The project emphasis has been on the presentation and query interface; the prototype environment is based on the UNIX operating system.
- More recently, in the Sequoia/2000 project, the University of California and Digital Equipment Corporation are collaborating to create and use a prototype network for the investigation of global warming, identifying and solving key global circulation and distributed computing system problems.[31] In addition to creating an application prototype, the project is concentrating on network delivery of very large objects in real time, consisting of migrating file systems, database extensions for geographic information, repository management, and visualization.
- At Carnegie Mellon University, the Alexandria Project is focused on browsing tools in encoded

databases,[32] and the Mercury project is focused on a nationwide electronic publishing system.[2]

Object-oriented databases[33] and hypermedia[34] are closely related to image libraries but so well represented in the literature that they need not be described here. Standards efforts, such as that for bibliographic search,[35] are important but beyond the scope of this paper.

In contrast to other digital libraries, DocSS exposes views of library catalogs and other tables for queries (but not for updates). It invents no new query language; its query language is "SQL SELECT ... ". Apart from confidentiality restrictions, any question answerable from the database can be posed, including questions that join the basic catalog tables to catalog extensions defined by library custodians, or tables belonging to independent applications. Thus, if a database containing a library also contains the catalogs of other subsystems (e.g., electronic mail, telephone book, bill-of-materials, etc.), users can inquire how documents are related to these subsystems.

Summary and conclusions

This paper has described a client-server design that creates distributed electronic libraries as a modest addition to widely supported operating system components. Each library is a digital analog of a public or private collection of papers and pictures. The new services occur in layered abstractions, with families of document managers riding on the Document Storage Subsystem (DocSS).

The DocSS layer is primitive, providing the structure and mechanisms for storing and cataloging data objects of all types and for recovering the information stored. Document managers are workstation programs. The storage subsystem can be implemented with SAA-compliant products among which standard SQL relational database management systems figure prominently.

DocSS is structured for extensions to specialized requirements and for exploitation of emerging technologies, such as multimedia services, without disrupting what already has been built. It distributes processing and hides environmental details by client-server techniques, providing for: multiple concurrent library sessions from each application process, with interactive or batch service within each session; system-managed placement of stored objects for performance and cost optimization; identical treatment of documents and folders; links between documents, as required for hypertext and engineering design; assistance for managing different versions originating from common information; and batching, buffering, and caching to shield the progress of applications from weaknesses of networks—particularly wide area networks.

The work described complements other digital library projects described above. What distinguishes DocSS from anything else that we have seen is how it manages data distribution over wide area networks and its synergism with distributed SQL databases and other components of evolving open systems. A practical implementation exists.

Acknowledgments

Five years of conversations within the Computer Science Departments of IBM Research have influenced what is described, as have similar discussions in the IBM development laboratories in Bethesda, Dallas, and Sindelfingen. I would like to acknowledge individuals, but the list is impossibly long.

Many joint study partners—most notably members of teams led by Russ Bohart, Ray Wells, and Larry Kite, of the governments of California, Alabama, and Illinois, respectively—commented on early DocSS designs. Their contributions make what is described correspond more nearly to real needs than it otherwise would.

The IBM IRM product was managed by an IBM program office led by Joe Ganahl and an IBM marketing team led by Mike Miller. The product was largely developed and integrated by an American Management Systems, Inc., team led by Paul Hudecek. The product is made viable by hundreds of refinements suggested by these teams.

I wish particularly to thank the designers and implementers of the first prototype, James Antognini, Robert Cubert, Dave Hildebrand, Steve Horne, Ken Rothermel, Bob Schmiedeskamp, and Rina Walach, for their contributions, and Bill Bigley, Pat Mantey, and Robin Williams for their support.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of the Open Software Foundation or UNIX Systems Laboratories, Inc.

Cited references

1. *Image and Records Management (IRM) General Information Guide*, IBM Systems Reference Library, GC22-0027, IBM Corporation (1991); *Baseline User's Guide*, IBM Systems Reference Library, SC22-0031, IBM Corporation (1991); *Application Programmer's Reference Guide*, IBM Systems Reference Library, SC22-0030, IBM Corporation (1991); all available through IBM branch offices.
2. C. Brazdziunas et al., *Development Plan for an Electronic Library System*, Carnegie Mellon University Information Networking Institute, Technical Report 1990-1, Pittsburgh, PA.
3. "Report of the APS Task Force on Electronic Information Systems," *Bulletin of the American Physical Society* **36**, No. 4, 1119–1151 (1991).
4. H. M. Gladney, *Requirements Analysis for a Document Storage Subsystem*, IBM Research Report RJ-7085, IBM Corporation (1989).
5. *Automated Records Management System Feasibility Study Report*, Arthur Young International for the State of California Department of Transportation (October 31, 1986).
6. *Ibid.*, p. III-7.
7. F. M. Wahl, K. Y. Wong, and R. G. Casey, "Block Segmentation and Text Extraction in Mixed Text/Image Documents," *Computer Graphics and Image Processing* **20**, 375 (1982).
8. T. W. Malone, K. R. Grant, F. A. Turbak, S. A. Brobst, and M. D. Cohen, "Intelligent Information Sharing Systems," *Communications of the ACM* **30**, No. 5, 390–402 (1987).
9. C. D. Avers and R. E. Probst, "ImagePlus as a Model for Application Solution Development," *IBM Systems Journal* **29**, No. 3, 356–370 (1990).
10. R. G. Casey, D. F. Ferguson, K. Mohiuddin, and E. Walach, "An Intelligent Forms Processing Subsystem," *Machine Vision and Applications* **5**, 143–155 (1992).
11. R. G. Casey and D. F. Ferguson, "Intelligent Forms Processing," *IBM Systems Journal* **29**, No. 3, 421–434 (1990).
12. R. Reinsch, "Distributed Database for SAA," *IBM Systems Journal* **27**, No. 3, 362–369 (1988).
13. *CICS/MVS Version 2.1 Master Index*, IBM Systems Reference Library, SC33-0513, IBM Corporation (1988); available through IBM branch offices.
14. J. P. Gelb, "System-Managed Storage," *IBM Systems Journal* **28**, No. 1, 77–103 (1989).
15. H. M. Gladney, "Inter-Machine Protocols for Electronic Libraries," *13th International Conference on Distributed Computing Systems* (May 1993).
16. C. J. Date, *An Introduction to Database Systems*, Third Edition, Addison-Wesley Publishing Co., Reading, MA (1981).
17. R. A. Lorie and W. Plouffe,"Complex Objects and Their Use in Design Transactions," *Engineering Design Applications, Proceedings of the Annual Meeting*, Database Week, San Jose (1983), pp. 115–121.
18. P. E. Mantey and D. E. Levy, "Electronic Libraries and Optical Jukebox Document/Image Libraries," *IEEE Symposium on Electronic Imaging Science and Technology* (February 1992).
19. *SQL/Data System Application Programming for VM/System Product and VM/Extended Architecture System Product*, IBM Systems Reference Library, SH09-8019, IBM Corporation (1988); available through IBM branch offices.
20. B. W. Lampson, "Designing a Global Name Service," *Proceedings of the 5th Annual Symposium on Principles of Distributed Computing* (1986), pp. 1–10.
21. *Data Communications Networks Directory Recommendations X.500-X.521*, Volume VIII, Fascicle VIII.8, CCITT IXth Plenary Session, Melbourne (November 1988); *Open Systems Interconnection—The Directory*, ISO DIS 9594-1 to 9594-8, ISO, Geneva, Switzerland (1988).
22. J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," *Proceedings of USENIX Association Winter Conference* (February 1988), pp. 191–202.
23. *Resource Access Control Facility (RACF) General Information Manual*, IBM Systems Reference Library, GC28-0722, IBM Corporation (1985); available through IBM branch offices.
24. R. H. Katz, "Toward a Unified Framework for Version Modeling in Engineering Databases," *ACM Computing Surveys* **22**, No. 4, 375–408 (1990).
25. G. R. Thoma, S. Suthasinekul, F. L. Walker, J. Cookson, and M. Rashidian, "A Prototype System for the Electronic Storage and Retrieval of Document Images," *ACM Transactions on Office Information Systems* **3**, No. 3, 279–291 (1985).
26. C. Ruckman, letter in "The Rocky Road to 'Computopia'," *Physics Today* **45**, No. 1, 94–96 (1992).
27. M. Sato, M. Koda, M. Ioka, and J.-K. Hong, "Image/Text Retrieval System on a LAN," *IEEE Office Automation Symposium* (April 1987), pp. 200–204.
28. J. Bescos, J. P. Secilla, and J. Navarro, "Filtering and Compression of Old Manuscripts by Adaptive Processing Techniques," *Proceedings of the Society for Information Display Symposium*, Las Vegas **21**, 384–387 (1990).
29. F. Mintzer and J. D. McFall, "Organization of a System for Managing the Text and Images That Describe an Art Collection," *Image Handling and Reproduction Systems Integration*, W. Bender and W. Plouffe, Editors, *Proceedings of SPIE* **1460**, 38–49 (1991).
30. B. Morgan and S. Jacobsen, *The UC Berkeley Image Database Project*, preprint (August 1988); informal summary report (February 1991).
31. M. Stonebraker and J. Dozier (principal investigators), *Sequoia/2000: Large Capacity Object Servers to Support Global Change Research*, seminar given at the IBM Almaden Research Center (October 1991).
32. M. Horowitz, F. Hansen, M. McInerny, T. Peters, and M. Wadlow, *The Alexandria Project: In Support of an Information Environment*, seminar given at the IBM Almaden Research Center (November 1991).
33. W. Kim and F. H. Lochovsky, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publishing Co., Reading, MA (1989).
34. B. J. Haan, P. Kahn, V. A. Riley, J. H. Coombs, and N. K. Meyrowitz, "IRIS Hypermedia Services," *Communications of the ACM* **35**, No. 1, 36–51 (1992).

35. C. A. Lynch, "The Z39.50 Information Retrieval Protocol: An Overview and Status Report," *Computer Communication Review* **21**, No. 1, 58–70 (1991).

Accepted for publication March 31, 1993.

Henry M. Gladney *IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120-6099 (electronic mail: Internet: gladney@almaden.ibm.com, Inter-enterprise: USIBMF6R at IBMMAIL).* Dr. Gladney is a research staff member in the Computer Science Department of the Almaden Research Center. He joined IBM Research in 1963, and has been there since with short sabbatical periods in other IBM divisions and in universities. His research has focused on applications of computers to scientific problems and is represented by articles in such diverse periodicals as the *Physical Review*, the *Journal of Analytical Chemistry*, the *IBM Systems Journal*, and *ACM Transactions on Database Systems*. His recent work has been directed toward distributed library systems for very large collections of documents and images, with special emphasis on related security needs. Dr. Gladney received his B.A. from Toronto University in 1960, his M.A. from Princeton University in 1962, and his Ph.D. from Princeton University in 1963. He is a member of ACM and a Fellow of APS.

Reprint Order No. G321-5523.