The Business Object Management System

by M. Schlatter
R. Furegati
F. Jeger
H. Schneider
H. Streckeisen

The Business Object Management System (BOMS) is a distributed resource manager that generalizes and extends the concepts of shared corporate information to include not only data that are structured such that the data can be held in relational tables but also generalized, complex business information objects. BOMS allows enterprises to store, manage, and query the totality of their documents, business transaction records, images, etc., in a uniform and consistent way. With this system, businesses can make more effective use of information that has in the past been inaccessible to thorough and systematic queries and that could not be integrated effectively into existing or new business processes. BOMS is targeted toward very large collections of information objects (on the order of a billion objects, equivalent to terabytes of data) and allows enterprises to unlock information treasures that would otherwise remain hidden in collections of that size. BOMS is influenced by theoretical concepts, such as object-orientation and hypermedia, but relies on proven relational database and transaction processing concepts. BOMS has been implemented with DATABASE 2™ (DB2®) and Customer Information Control System/Enterprise Systems Architecture (CICS/ESA™) and has been in productive use since 1991.

The systematic and effective use of data today is just a small fraction of what it potentially could be—given appropriate methods and procedures to access, exploit, and share that data, and based on a common semantic understanding. Data are often plentiful, but the problem is to transform the data effectively into information that contributes to the achievement of business and competitive capabilities. "Data" and "information" are not the same, although many people often use these words as synonyms; ironically, an enterprise can be data-rich, but information-poor. In order to transform data into valuable information, the data must be associated with semantics and put into context before being made available to the knowledge worker (one who extracts and organizes information from data). A major problem for the knowledge worker is to select the data that are pertinent in a given situation and put that data into context with other data. A related problem is the failure to recognize links and similarities between pieces of data that are stored in different formats and in separate locations or libraries.

Information (or document) retrieval system design has been the poor stepchild of the computer revolution, and, although relational databases[1,2] such as DATABASE 2* (DB2*)[3] and enterprise-wide data models [4–9] provide a solution for data that are structured such that they can be held in tables, the wealth of unstructured information is still largely inaccessible to systematic queries and cannot be integrated easily into existing or new business

processes. Relational database management systems (RDBMS) were designed to avoid the disadvantages of earlier hierarchical and network database systems and to solve the needs of business data processing applications. Although they have succeeded at this task admirably, standard relational databases are insufficient for computer-aided design data, documents, etc., and need to be extended.[10-13] Similarly, full text search information retrieval systems were designed to avoid the disadvantages and complexities of primitive indexing systems. However, they often provide unsatisfactory results: for example, in a primitive full text search system in which each document contains a line *Document Number* = . . . , searching for documents about *documents* (e.g., about document processing, archiving, handling, management, etc.) can become nearly impossible. A fundamental problem here is that pure full text search systems typically lack clearly defined data items with limited meaning and role(s).

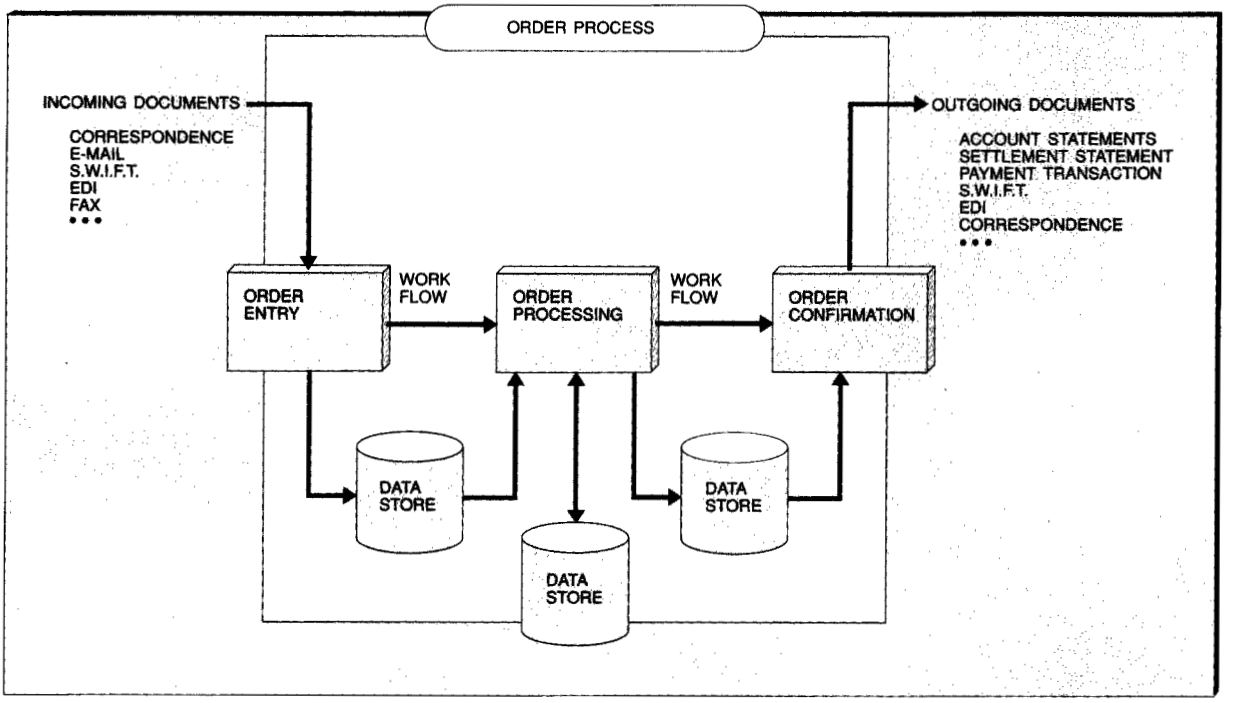
The logical foundations of information retrieval and data retrieval systems are fundamentally different.

As a consequence, even large enterprises with sophisticated, state-of-the-art data processing systems exploit surprisingly small portions of their data systematically. In most cases, only data that lend themselves to tabular structuring, such as accounting data, customer reference data, etc., are accessible for data processing. The vast majority of data entering an enterprise, especially data that are generated through formal or informal internal business processes, or that the enterprise communicates to external involved parties, are "dead data" because there are no effective ways to transform that data into useful and valuable business information. Figure 1 schematically shows the flow of work and related data through an enterprise. Our example is from the world of financial institutions but is also typical of many other types of enterprises. Incoming documents include traditional correspondence and machine-readable documents that are transmitted through EDI (electronic document interchange) or specialized networks, such as S.W.I.F.T. (Society for Worldwide Interbank Financial Telecommunication). The data stores shown are typically transient, work-in-process queues. Because the representation form and structure of most of the information in these queues is not sufficiently standardized, there is no straightforward way to store the information in the operational database in a manner that would allow users to query, join, and retrieve the information by a unified method.

In our example, we assume that business professionals and clerical workers use word processors and spreadsheets to prepare customer contracts and related analyses and memos, and the contracts and the supporting material are sent either as plain, traditional paper documents or through the internal electronic mail system of the enterprise to the manager that must approve the contracts. However, this work is usually done ad hoc, in isolation, and related to specific business process instances. Consequently, once a business transaction is completed, the associated information is cleared from the transient work queues (in our example, the documents and memos prepared on word processors and spreadsheets) and is no longer accessible for future online investigation. Much information that could be valuable, for example, for later customer or market segment analysis, to define future marketing plans, etc., is lost, or the information is impractical to locate and consolidate from paper archives or from electronic archives that were designed as, or just happened to become, "logical islands."

Commercial development of information retrieval has frequently treated document retrieval as merely a variant of data retrieval. However, the logical foundations of information retrieval and data retrieval systems are fundamentally different.[14-16] A data retrieval system directly answers deterministic questions, such as what is John Doe's account balance? The fundamental criterion of success for a data retrieval system is correctness, i.e., one needs only to ask: does the system correctly answer the question? An information retrieval system is more indirect and probabilistic, i.e., it provides references to a document or to a set of documents that will *likely* contain what the user wants. For example, an

Figure 1 Flow of work and related data through an enterprise



investment adviser in a bank may want to determine which customers are promising prospects for a new product that the bank is about to launch. The criterion for successful information retrieval in such cases is utility, as in this case: "Did the investment adviser find all the useful information?" This is a much more subjective criterion than correctness that can be applied in the case of a data retrieval system.

Because of the fuzzy nature of information retrieval, its effectiveness is less dependent on the physical searching speed of the system and is more dependent on the number of logical decisions the user must make in a search. These decisions include the construction of formal search requests, evaluating the usefulness of document references, and revising formal search requests if the results from an initial search are not satisfactory.

Some electronic document storage systems are designed to emulate traditional paper or COM (computer output on microfiche) archives in the sense that they provide only a small number of predetermined access points (for example, customer number, or creation date). Many such document storage systems provide only static "folders" (files used to store and organize electronic documents). Documents must be linked to these folders when they are stored. Consequently, users cannot define their own folders to collect all documents that are relevant in a given context, perhaps even years after the documents have been stored. Because of their static classification schemes and their limited flexibility, such designs must pretend that it is possible to foresee how information will be used in the future. Unfortunately, that is seldom possible, and such designs are, therefore, unable to solve some of the most pressing needs. Instead, in order to allow a user to view stored document collections according to current information needs, and in order to find relevant information with as few logical decisions as possible, the document retrieval system must provide a large number of individual access points to a single document, and access must be allowed via a large number of ad hoc specifications of Boolean combinations of these access points.

Moreover, it must allow folders to be defined dynamically.

In this paper, we first describe the background setting for the Business Object Management System (BOMS). We then describe the basic BOMS concepts to manage the organizational and administrative complexity that is inherent to very large collections of complex information objects. We show that BOMS is influenced by theoretical concepts, such as object orientation and hypermedia, but takes a pragmatic approach to solve a real and pressing business need.

In presenting the concepts and system design of BOMS, we first discuss a classification scheme to divide a collection of information objects into *families*, with family-specific attributes that can be used as search terms, for access control, for presentation, and for storage management purposes. Then we describe the main components of the logical structure of BOMS, i.e., the libraries, the catalog, and the environment store. We show how the environment store and the catalog, together with access control rules and other metadata are used to provide users with a single-system view, and with flexible ways to search for and to establish dynamic relationships between objects that are relevant in a given business context.

In a separate section, we describe an access control scheme that allows security policies to be specified in a high-level language. Because the specification of security policies is orthogonal to the administration of user and object attributes, the scheme adapts well to large numbers of users and objects in a dynamic environment with the need for fast, flexible, and reliable adaptation of access constraints to the needs implied by changing circumstances.

Finally, we describe the system design and a practical implementation of a BOMS distributed resource manager that was driven by the stringent capacity and throughput requirements of a major Swiss bank. Under the control of an integrated, CICS/ESA*-based (Customer Information Control System/Enterprise Systems Architecture-based) transaction management system, BOMS services can be accessed by human users and by distributed client processes. The BOMS transaction manager mediates between the users' single-system image and the underlying hardware and software complexity and heterogeneity. In particular, the BOMS transaction manager hides any physical segmentation of the BOMS information store from the users.

Enterprise information management

Changing the nature of the work itself. In markets with little or no growth, the overall economic climate, increasing cost pressure, and aggressive competition demand more effective ways of doing business to maintain or increase market share and profitability. Information systems play a vital role in this effort,[17–21] but they are also an important element in growth strategies. Such strategies must be designed to meet tomorrow's main business needs, i.e., the need to deliver new and enhanced services, both within the enterprise and to customers or prospects, while keeping administrative costs under control.

Information systems are an important element in growth strategies.

The main goal of traditional image and document processing systems is often to increase the efficiency of "back office" operations (those not interfacing with customers). Increasing the efficiency of the "front office" is more or less a side effect, and most of the knowledge worker's needs (or dreams) remain unfulfilled. We are, however, not satisfied with making current processes more efficient but want to allow and even encourage new and more effective business processes. Therefore, we propose a more radical new approach that allows knowledge workers and other business professionals to exploit greater portions of the large percentage (typically 95 percent or more) of information that is today still inaccessible for on-line analysis, thereby enabling enterprises to introduce new types of products and

services. Such an approach becomes increasingly important for enterprises that want to become more effective and gain a competitive advantage by actively and consciously exploiting larger portions of the information that has in the past been inaccessible to systematic and consistent analysis, and that could not be integrated into existing or new business services. From our own practical work, we know that these advantages are substantial for large financial institutions, but we believe that similar considerations hold for other types of enterprises as well. For example, there are strong indications that the pharmaceutical industry could achieve significant reductions in their "time to market" if they had a uniform and flexible way to access and consolidate the wealth of information that is produced before a new drug is submitted for registration.

Computerized collections of millions and billions of complex information objects are, or will soon become, a practical business requirement. Moreover, such collections will often grow at rates of up to one hundred million new objects per annum (p.a.), and new objects will have to be kept for very long times. Large financial institutions typically have the need, for audit and for legal reasons, to keep business records for 10 or more years. For large banks, up to ten million account statements p.a. and up to one hundred million payment transaction records p.a. are not unusual, and, consequently, one billion (10^9) objects will soon become the typical order of magnitude that must be handled.

The challenge for enterprises with such ambitions is to find a practical solution to the problem of managing the organizational and administrative complexity that comes along with collections of that size. The key to solving this problem is to hide the complexity from the users and to provide them with a single-system view of all of the complex information objects of an enterprise, including both current and old (i.e., archived) objects. Moreover, users need a unified and coherent way to query and handle all objects in the collection.

Access control. Managing and actively exploiting such new orders of magnitude also requires fundamentally new security concepts. In order for legitimate users to be able to easily access information for which they are authorized, without having the information compromised by unauthorized users, such huge collections of valuable business information require new types of access control.

Table 1 Main objectives of access control

Privacy	Ability to decide whether, when, and to whom information is released, and to enforce such decisions. Included is the ability to decide that certain information is not to be released except to selected individuals and to enforce the decision.
Secrecy	Ability to present the release of secret information to individuals who are not cleared (i.e., authorized) to see such classified information. This is equivalent to the privacy requirement except that the decision on whether information is released is based on two particular information and user attributes, i.e., *secret* and *cleared*, which are kept under light administrative control.
Integrity	Ability to prevent unauthorized modification of information.

Access control is an aspect of information security or, more specifically, an aspect of information risk management, traditionally having the three main objectives listed in Table 1.

For the enterprise-wide exploitation of shared corporate information objects, being able to effectively manage users' rights, i.e., to specify granular access controls without incurring an unacceptable administrative workload, becomes critically important. However, when collections of information objects reach the size that we envision for BOMS, and when they are in a dynamic environment with the need for fast, flexible, granular, and reliable adaptation of access constraints to changing circumstances, most currently available access control models become impractical. The main problems are the number and the complexity of the administrative decisions and actions that are required to enforce the access control policies of an enterprise. For the expected number and frequency of personnel, organizational, and environmental changes, currently available models make it difficult to figure out how to specify access controls, and it is difficult to verify that the controls in place indeed correspond to what is intended. Consequently, the number and complexity of administrative decisions and actions required to enforce adequate security policies could prevent the implementation and exploitation of the foreseen enterprise-

wide information object stores. Without fundamentally new concepts to replace traditional clerical procedures of resource access control specification and administration by more efficient, flexible, and to a large extent automatic procedures, either huge graveyards of inactive and largely inaccessible data would result, or unacceptable security exposures would occur.

The Business Object Management System. The key to our approach is the notion of a Business Object Management System (BOMS) for complex information objects defined as logically connected sets of information that can be referred to and manipulated in their collective form. The size of such objects can vary from a few bytes to megabytes. Note that our definition of an object is different from the definitions that are normally used in the context of object-oriented programming.

BOMS is a separate transaction management layer on top of one or multiple, potentially heterogeneous, database management systems. BOMS attempts to combine the advantages of RDBMS with some of the advantages of object-oriented database management systems (OODBMSs). The strengths of an RDBMS include the capability to support multiple logical views of shared data and set-oriented queries, whereas an OODBMS provides support for complex objects with encapsulated semantics. Views permit each application or query to see data organized in its own preferred way, and encapsulation shields programmers from irrelevant implementation details and forces them to access data only through strictly controlled interfaces. BOMS provides all of that but is not meant to be a general-purpose OODBMS. For example, BOMS is not meant to extend procedural programming languages with support for persistent data types, and inheritance is supported only in a very limited sense.

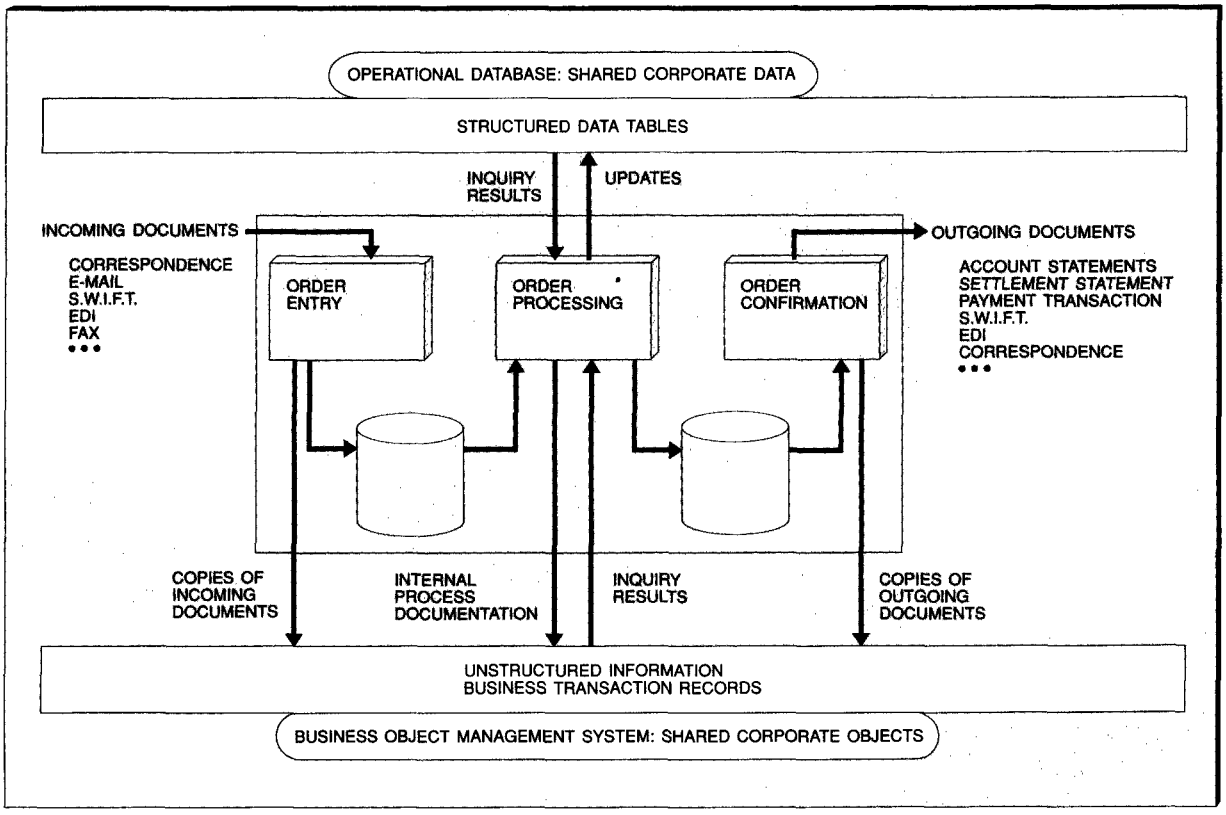
BOMS provides a methodology to structure and position all of the complex information objects of an enterprise such that they become a known, integrated, and well-managed part of the information assets of the enterprise. Conceptually, the BOMS methodology includes elements of traditional data modeling methodologies that allow operational data to be structured by way of classification schemes. However, these concepts are extended such that they apply not only to structured data that can be held in relational tables but also to complex business information objects that are insufficiently structured for relational database management systems.

BOMS is first of all designed to support the knowledge worker in new ways that go beyond what image and document processing systems normally intend and are capable of. In addition, BOMS also provides new opportunities to make the front office not only more efficient, but also more effective, and provides opportunities for new front office services that were previously impossible. Examples are the capability for a total customer view, and immediate responses to customer inquiries even when they relate to complex relationships between involved parties.

Figure 2 shows how BOMS complements the traditional operational database where structured data are stored, normally in the form of tables. BOMS is enterprise-wide, i.e., it covers the complete work flow: from order entry, through order processing, to order confirmation. Moreover, as Figure 3 shows, BOMS also supports business processes that traditionally received little or no support from the operational database, such as marketing. Postprocessing of order confirmations seen in the figure is an example of a business process that extracts information that is expected to become useful for later analyses. Figure 4 depicts how BOMS, together with the traditional operational database, adds new qualities to existing business processes, such as the capability to provide business professionals with a total customer view, or the capability for immediate responses to customer inquiries. Moreover, and perhaps more importantly, new services become possible, such as fundamentally new types of information research and analysis, an information subscription for business professionals, and, potentially, a wealth of other, new business processes that still wait to be devised—all to make an enterprise more flexible and more competitive.

Related work. A series of articles[16,22–27] elaborate on the hypothesis that new methods are needed to filter and control the potentially unlimited flow of information that the information age promises. They argue that information retrieval systems have in the past ignored some aspects of the more general area of information filtering, and they discuss how current information retrieval models could be extended.

Figure 2 BOMS complements the traditional operational database

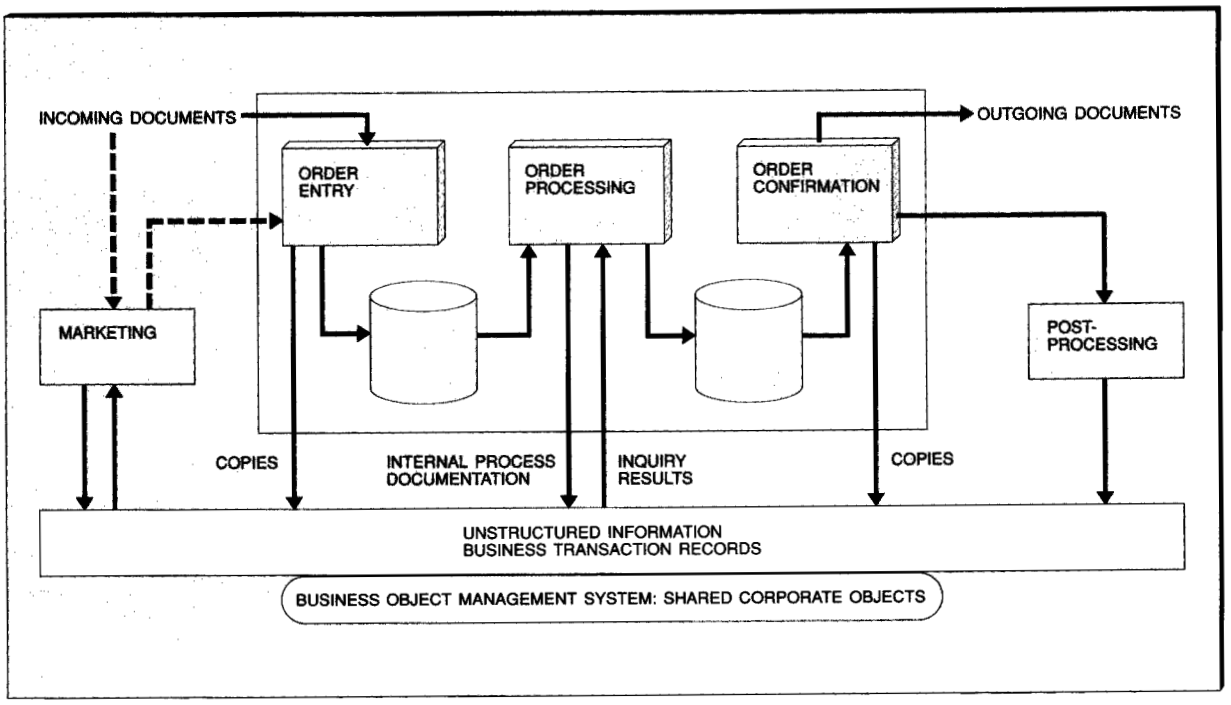


Linnemann et al.[10] discuss the "misuse" of traditional database systems as "byte containers" for complex data objects that are not sufficiently structured for the underlying database. A consequence of such a misuse is that the database system cannot support search predicates on the contents of these containers. It is left to the application programs to interpret these byte strings and to implement the functions that manipulate them. The resulting high dependency between the physical data representation and individual application programs negates one of the major advantages of database systems. To overcome these drawbacks, an extension of the traditional relational model is proposed that supports "nested" relations and that has an SQL-like language interface for complex objects. An extended version of SQL (Structured Query Language) with object-oriented features for structured complex objects is also discussed in Gardarin et al.[11]

With the IBM Information Warehouse* concept,[28] some aspects of which are discussed in other papers in this issue of the *IBM Systems Journal*, BOMS shares the focus on corporate information assets and the goal to provide knowledge workers with easy access to such assets. A related goal that BOMS shares with the Information Warehouse architecture is the concept of providing users with a single-system view of a potentially heterogeneous and distributed set of information.

Although BOMS has been conceived, designed, and implemented independently, BOMS could be seen as a way to extend the scope of the current Information Warehouse architecture to provide support for nonformatted and compound information. Such extensions could include libraries for documents and for business transaction records or, more generally, for arbitrary collections of data of different types which together consti-

Figure 3 BOMS support of business processes not supported by operational database

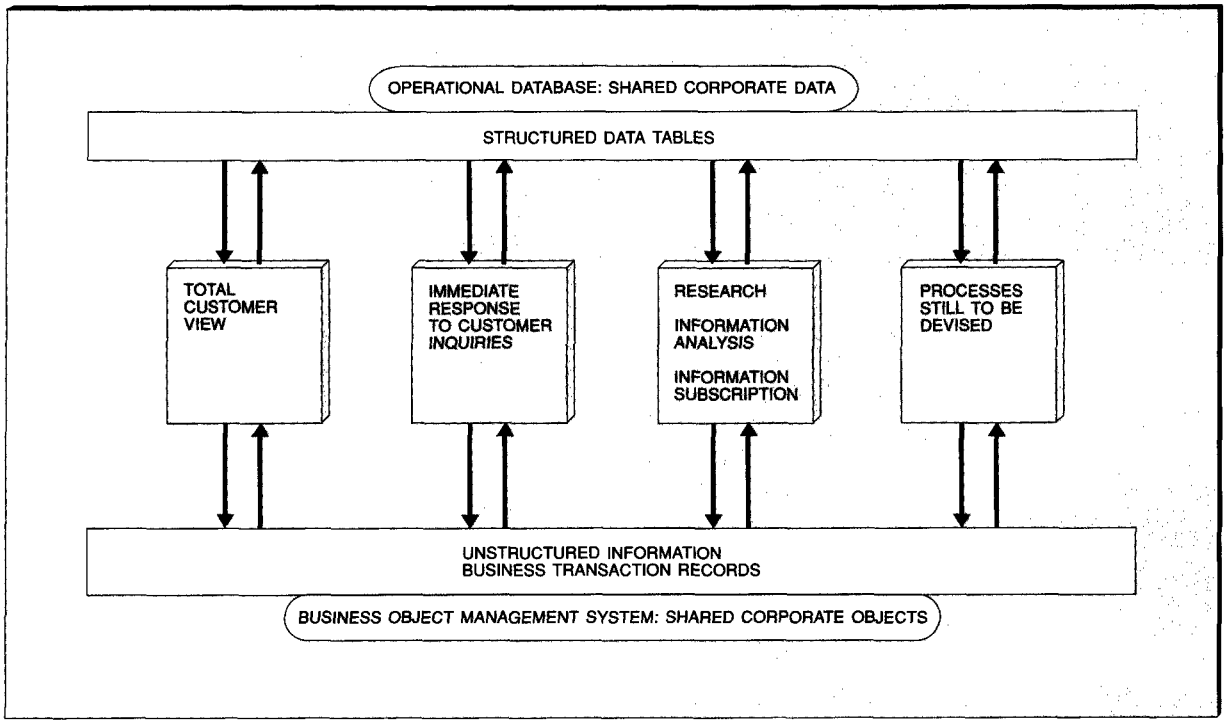


tute complex business objects with many complex relationships.

Although BOMS does provide support for complex information objects as we have defined them above, BOMS is not an OODBMS, at least not in the sense as it is described by many current researchers.[29–38] OODBMSs typically focus on removing the semantic gap between application domains and their representation in persistent storage. An important goal of many current OODBMSs is to alleviate the mismatch between procedural programming languages and traditional database systems by allowing applications to store arbitrary programming or GUI (graphical user interface) objects directly into persistent storage. The approach followed by many currently available OODBMSs is to enhance object-oriented programming languages with functions to access and manipulate persistent program objects. However, imbedding persistent data into a procedural programming language such as C++ precludes many of the advantages of a nonprocedural, set-oriented query language such as SQL. Compared to relational systems, that procedural approach results in a loss of logical data independence, i.e., the single most important advantage of relational database systems. In addition, OODBMSs typically represent relationships as distinct from data values, whereas RDBMS represent relationships by data values. A resulting drawback is that many current OODBMSs cannot easily support multiple logical views of shared data, i.e., the kind of queries that can be supported depends on how individual objects and object collections are designed. A practical requirement that follows is that, with an OODBMS, one should know in advance (i.e., when the database and the object collections are designed) how the information will be used in the future. Because that knowledge is nearly impossible to obtain for information with a useful lifetime of ten or more years, that requirement can effectively limit the potential use of information stored in an OODBMS.

Examples of prototype database systems that have explored new concepts such as object support are ORION,[31] POSTGRES,[39] and Starburst.[13] Whereas ORION is typical for the revolutionary approach that starts from scratch, POSTGRES and

Figure 4 BOMS adds new qualities to existing business processes



Starburst are typical for the evolutionary approach that extends SQL with new features. POSTGRES was developed at the University of California, Berkeley. POSTGRES adds object and rule management capabilities to the functions offered by a traditional database management system. The POSTGRES object management capabilities are designed to support nontraditional data types such as bit maps and polygons that are required for computer-aided design and other engineering applications. The POSTGRES rules system supports triggers, i.e., event-driven programming and a more flexible and more powerful way to specify and to enforce integrity constraints—compared with the referential integrity rules normally supported by current database management systems.

The Starburst project at IBM's Almaden Research Center is another example of an evolutionary approach to extend existing relational databases. Starburst is not a pure OODBMS, but it addresses many of the issues that OODBMSs raise, including complex object support, extensible types and methods, encapsulation, inheritance, overloading, and late binding. Overloading denotes the concept whereby a method may have different implementations. When a method is called, the implementation that is dispatched and executed depends on the type of the object on which the method is invoked. Starburst is more ambitious than earlier object-oriented systems because the focus is not only on accommodating objects, but also on extensions based on a set-oriented, declarative query language.

Extensions to the conventional relational data algebra to model the evolution of database schemas are described in McKenzie and Snodgrass.[40] Conventional databases allow only one schema to be in force at a time. Consequently, when the schema is modified, for example, when new attributes are introduced or when existing attributes are either merged or split, the database must be restructured, or *reorganized*. For BOMS, and for other databases that store past states, such reorganizations are no longer adequate. Instead, multiple schemas must be in effect simul-

taneously, each of which applies to a specific interval in the past. In that sense *schema versioning* refers to the retention of past schemas that result from a schema evolution. References 41 and 42 are papers related to this subject.

A central problem of information retrieval is how to represent information for retrieval.

Melampus,[43] a research prototype (named after a great seer in Greek mythology who could understand the speech of animals and birds), addresses the problem of finding related data that may originate from different sources and that may lack common formats and even semantics. It is argued that the lack of a comprehensive way to manipulate the wealth of information in a system is a fundamental reason why the worth of the information resource is often only latent and cannot be fully realized. Melampus intends to provide a computing environment that will enable data to be used in unanticipated ways, ease the formation of new relationships among data, and promote the sharing of data between applications.

Rufus[44] is an object-oriented data model and a storage system with associated search methods. It is built around a centralized description of data types and formats that supports the construction of applications operating across data types. By integrating the data attributes in a central place, retrieval on a semantic level rather than at a purely syntactical (i.e., text-oriented) level becomes possible. Rufus attempts to eliminate the problems of more conservative approaches in which the semantics of the data formats are locked away in individual application programs.

Document and image processing have recently caught widespread interest.[45–49] However, many publications show a bias toward small-scale applications. Practical aspects such as systems administration and integration into enterprise operations are mostly omitted.

Through the availability of relatively cheap optical storage media, scanning of incoming documents[50,51] has recently become, for some enterprises, a means to reduce the costs that are associated with the handling of paper, i.e., a task that does not require specialized skills but that is labor-intensive. With ImagePlus*, when paper first enters the business, its information is captured as an electronic image.[52] From that point it is distributed, tracked, and processed electronically. In addition to providing cost-effective ways to replace warehouses filled with paper, optical disk storage is also used to replace COM archives to store, in a bank, for example, financial transaction statements as they are generated by the operational data processing applications. Although these uses of new technology are valuable, they are mainly targeted at automating existing processes to make them more efficient.

The majority of information retrieval research has been aimed at more experimentally tractable small-scale systems, but it is increasingly apparent that retrieval systems with large numbers of documents are a fundamentally different genre of system than small-scale systems, and that quantitative growth of an information retrieval system causes qualitative changes in its structure and processes.

A series of research reports[53–56] and a recent *IBM Systems Journal* paper[57] describe the requirements analysis, architecture, design, and implementation of a document storage subsystem that has evolved to IRM, the IBM Image and Records Management System.[58–60] IRM is a toolkit that provides components for image scanning, displaying, and printing services, and for object library, folder management, and work list management services. These services can be customized and integrated to produce comprehensive, versatile image processing and work flow management systems with custom graphical interfaces.

BOMS concepts and system design

Object families. Object families, which we define as sets of objects sharing some common properties, implement a basic BOMS concept that we have adapted to an enterprise scale and demonstrated in practice. They address the organizational and administrative requirements of very large object collections by means of an n-dimen-

Figure 5 Two-dimensional categorization of information objects

	TOP MANAGEMENT	LEGAL DEPARTMENT	AUDIT DEPARTMENT	PAYMENTS DEPARTMENT	SECURITIES DEPARTMENT	...	OBJECT SEMANTICS
MINUTES	F_{11}	F_{12}	F_{13}	N/A	N/A		
INCOMING LETTERS	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}		
ORDERS	N/A	N/A	N/A	F_{34}	F_{35}		
STATEMENTS	N/A	N/A	N/A	F_{44}	F_{45}		
OUTGOING LETTERS	F_{51}	F_{52}	F_{53}	F_{54}	F_{55}		
CONTRACTS	F_{61}	F_{62}	N/A	F_{64}	F_{65}		
REPORTS	F_{71}	F_{72}	F_{73}	N/A	N/A		
...							
	ORGANIZATIONAL SOURCE						

N/A: NOT APPLICABLE

sional classification scheme that assigns each object to exactly one class. Families are uniquely identified by *n family descriptors*, i.e., attributes with values that are shared by all family members. In addition, each family is characterized by sets of attributes that are mandatory for all members of the family, but for which each family member (i.e., each object) has its own values; we call these attributes "object descriptors."

In principle, BOMS allows an arbitrary number of family descriptors, i.e., $n = 1, 2, 3, \ldots$, and n can be changed over time. However, as seen in Figure 5, $n = 2$ is often an intuitive and practical choice:

1. The enterprise's *organizational unit* that creates information objects, or that receives the information object from an external involved party; examples are: top management, legal department, audit department, etc.
2. The *semantic type* of an information object, such as minutes, incoming letters, orders, statements, etc.

As illustrated in Figure 5, setting up BOMS with $n = 2$ family descriptors leads to a two-dimensional categorization of all information objects, which is intuitive and appealing, which appears to reflect the business reality of many enterprises well, and which appears to be relatively stable over time. Obviously, the values of the family descriptors "organizational source" and "object semantics" are installation-specific; the values of the columns (top management, legal department, etc.) and the rows (minutes, incoming letters, etc.) in Figure 5 are only examples. Moreover, not all possible combinations may be applicable in a given scenario, i.e., there may be families with no members. For example, minutes may be generated by top management, by legal, and by the audit department, but not by either the payments department or the securities department. Similarly, orders and statements may relate to the payments and securities departments, but not to top management, nor to legal or the audit department.

BOMS information objects. A central problem of information retrieval is how to represent information for retrieval. BOMS information objects are logically connected sets of information that can be referred to and manipulated in their collective form. Conceptually, BOMS information objects contain the following five structural elements: object profile, body, search terms, comments, and transforms.

Object profile. The object profile structural element contains object descriptors that can be used

as search arguments, for access control, to control presentation and storage management, and for administrative purposes. It also contains meta-information about how the object is represented, i.e., whether it is an EBCDIC (extended binary-coded decimal interchange code) string, an RFT (revisable form text) data stream, an image, or information that is encoded in some other form. The types and names of mandatory and optional descriptors in the object profile are defined by the *family* to which the object belongs.

A basic BOMS concept is the requirement that all object profiles must be time-invariant and that all descriptors in an object profile must describe only the object itself. More specifically, the descriptors in the object profiles must not describe facts that apply to multiple objects. For example, they must not hold information about the environment in which the object was created, such as information about the organizational structure of the enterprise. Instead, such information is kept—separate from the object profiles and redundancy-free—in the environment store that we will describe presently.

Object descriptors can be scalars, or n-dimensional vectors. Figure 6 shows, as an example, an object from the hypothetical family F_{45} (see Figure 5), i.e., a security settlement statement. In this example, the order number is a scalar because it is a descriptor that consists of a single value. This descriptor contrasts with a vector descriptor that consists of an ordered set of numbers. In our example, an item called *To the Debit of Account* is a three-dimensional vector, the three dimensions being account number, value date, and amount. Objects can have multiple instances of descriptors. For example, the object in Figure 6 has two instances of the descriptor commissions.

Body. Body is the main information content. From the BOMS perspective, the body is an uninterpreted bit string that is handled in its entirety. Conceptually, there is no upper limit to the size of the body, and the information that is contained in the body can be represented as an EBCDIC string, an RFT data stream, a binary encoded image, or in any other form. However, BOMS maintains information about the representation form of the body, i.e., so-called meta-information, in the object profile.

Search terms. The search terms structural element contains additional items that can be used for queries.

Comments. The comments element contains annotations that users may attach (i.e., "staple"), over time, to the object.

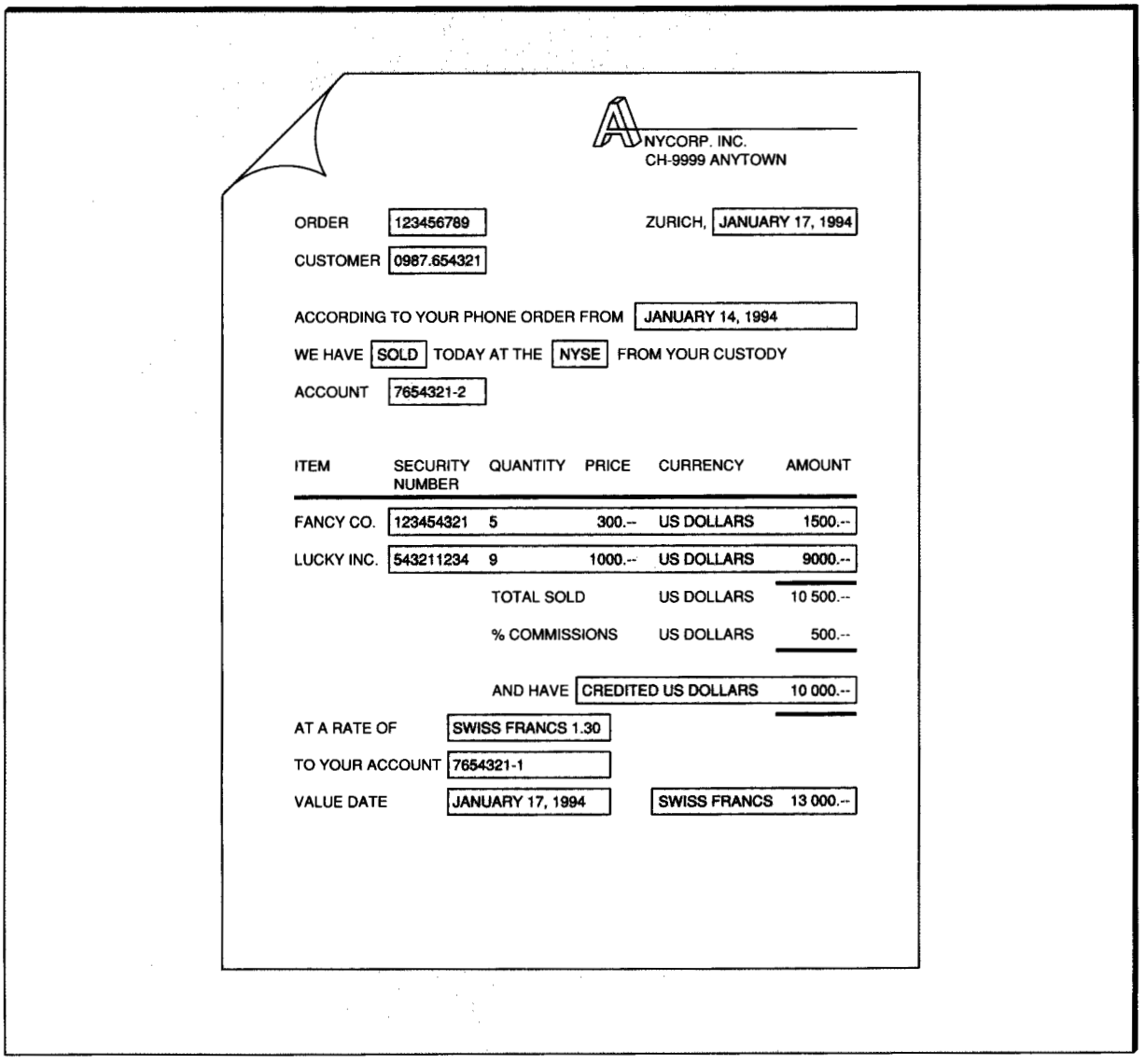
Transforms. The transforms structural element contains transformations of the object body, such as an abstract of a text document, a verbal description of an image, a low-resolution, compressed form of an image, etc.

All structural elements and the object descriptors in the object profile are self-defining, i.e., have their associated element profiles that describe the data format and the number of instances contained. This allows us, for example, to relate an arbitrary number of comments with a given object and with multiple transforms. Moreover, schema evolutions[40] and changes in the representation form of object descriptors can be hidden from the users. This support for multiple versions of descriptors allows BOMS to adapt to changing requirements without the need to change previously stored objects, and without disturbing the users' consistent and stable single-system view.

BOMS structure. Figure 7 shows the basic BOMS structure. It consists of two main processes: the service request manager, which is the main front-end process, and the library manager, which is the main back-end process that accesses the libraries (Library 1, Library 2, . . . , Library n) where information objects reside.

The service request manager provides the interface through which human users and automated client processes interact with BOMS. The service request manager is a complex information object resource manager that supports requests to store objects and to query or retrieve objects. It operates asynchronously, i.e., once a terminal user or client process has issued a service request, it is free to continue with other tasks. Service requests are persistent across sessions, i.e., they survive user logoff and system restart operations. When a service request has been processed, the service request manager puts the reply into the service reply queue for subsequent display on the

Figure 6 Security settlement statement example



ANYCORP. INC.
CH-9999 ANYTOWN

ORDER 123456789 ZURICH, JANUARY 17, 1994

CUSTOMER 0987.654321

ACCORDING TO YOUR PHONE ORDER FROM JANUARY 14, 1994

WE HAVE SOLD TODAY AT THE NYSE FROM YOUR CUSTODY

ACCOUNT 7654321-2

ITEM	SECURITY NUMBER	QUANTITY	PRICE	CURRENCY	AMOUNT
FANCY CO.	123454321	5	300.--	US DOLLARS	1500.--
LUCKY INC.	543211234	9	1000.--	US DOLLARS	9000.--
		TOTAL SOLD		US DOLLARS	10 500.--
		% COMMISSIONS		US DOLLARS	500.--
		AND HAVE	CREDITED US DOLLARS		10 000.--

AT A RATE OF SWISS FRANCS 1.30

TO YOUR ACCOUNT 7654321-1

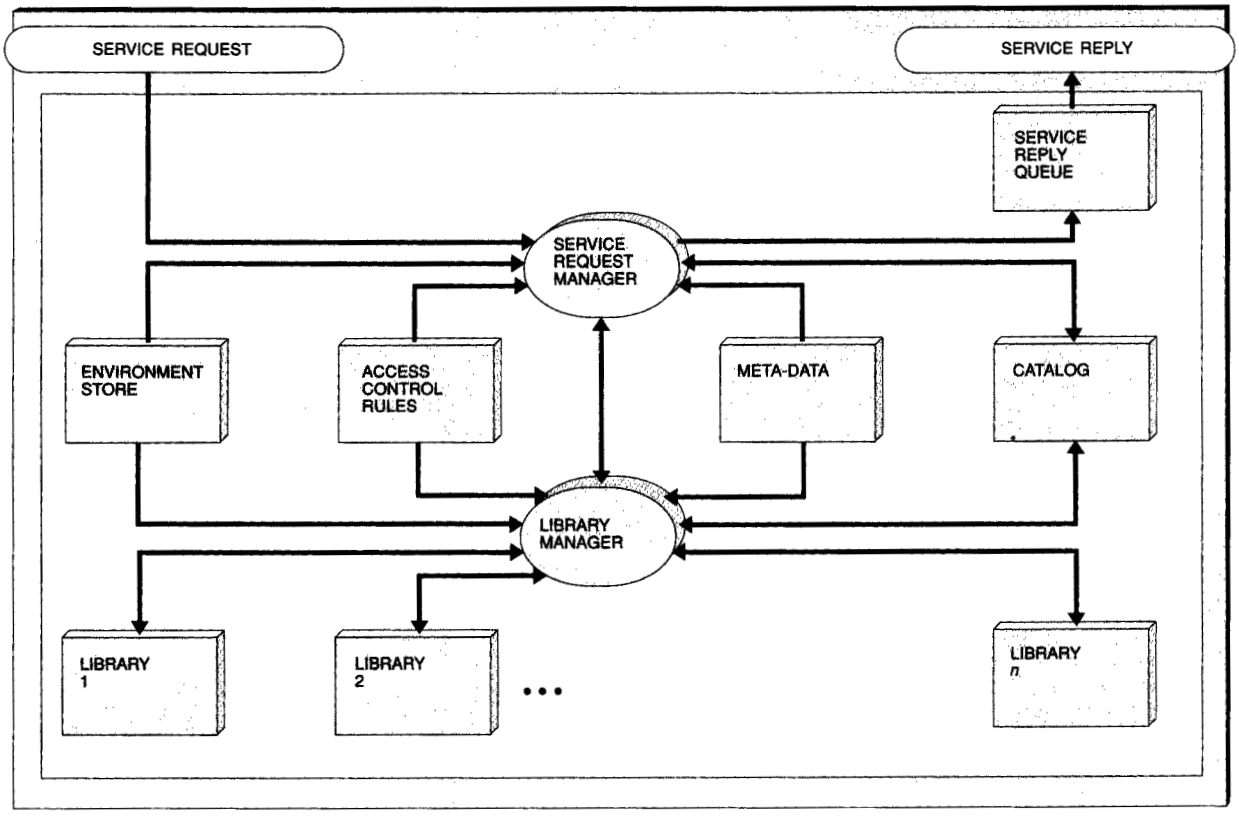
VALUE DATE JANUARY 17, 1994 SWISS FRANCS 13 000.--

user's terminal or for further manipulation by the client process. For simplicity, Figure 7 shows only a single service reply queue. In reality, however, each client process has its private area in that queue and sees only the replies that pertain to its own requests. These private areas are also used to synchronize and recompose the results from multiple background processes that may run independently and asynchronously.

The service request manager handles two main types of requests:

1. Object retrieval requests, requests for objects that meet certain criteria. Such requests are transformed into one or multiple *library orders* and then passed on to the library manager. The transformation relies on the catalog, on access control rules, on information from the envi-

Figure 7 The basic BOMS structure



ronment store, and other meta-data. When the library manager returns the requested objects, the service request manager places them into the service reply queue and notifies the requesting process, or terminal user.

2. Object query requests, requests for lists of objects that meet certain criteria. Such requests do not require that the objects actually be retrieved, i.e., they can be handled by querying and joining information from the catalog, from the access control rules base, from information about the environment, and from other meta-data. However, the replies to such object query requests are also put into the service reply queue, similar to retrieved object bodies.

The service request manager provides terminal users and client processes with a view that emphasizes the semantic commonalities of information objects and hides possible differences in the representation and storage formats. For example, all objects of the semantic type *contract* can be handled in a uniform and coherent way, even if they manifest themselves in different data stream formats, such as typed letters and scanned images. Objects of different data stream formats can be kept under the control of different storage subsystems without impacting the single-system/single-library view that users have.

The library manager fulfills the library orders received from the service request manager, i.e., stores and retrieves the *bodies* and other structural elements of the BOMS information objects. It is a binary object resource manager in the sense that it is not aware of the nature (i.e., both the format and the semantics) of the objects it is handling.

The Libraries 1 to *n* are the stores for the *bodies* and for related structural elements of the BOMS information objects. All libraries have the same

architecture, but they may vary in physical implementation. Each library consists of the library shell, which provides the interface code to the library manager, and the library core, implemented with the underlying DBMS or access methods, or both. This layered structure allows BOMS to integrate multiple storage subsystems that can be based on different technologies—without disturbing the users' single-system, or single-library view. It allows enterprises to take advantage of newer storage subsystems with better price-performance ratios, as they will inevitably become available through advances in technology. Typically, new objects will be stored in libraries that are implemented with new storage technology, while the old objects can continue to reside in the older libraries that are implemented with older storage technology. The BOMS design guarantees a single-library view, i.e., both the old and the new objects are seen by the users as if they were stored in a single, uniform library.

In addition to the two main processes, Figure 7 also shows the catalog, the environment store, the access control rule base, and the meta-data store, all of which provide input to both the service request manager and the library manager processes.

Conceptually, the catalog is a redundancy-free table in which each row represents the time-invariant attributes of an information object and pointers to the object and its related elements, such as comments and transformations, in the BOMS libraries. The attributes include the object profiles and all element profiles. They are represented as scalars or as complex structures, such as vectors or sets of vectors.

The environment store contains time-stamped information about the state of the environment both in the past and present, i.e., at the time when BOMS processes an object query or retrieval request. From the environment store, BOMS can infer information about the state of the environment at certain times during the life cycle of an object; for example, when the object was created, five years after it was created, or now. Such information could, in principle, also be stored in the catalog, but the resulting number of catalog updates required to reflect environment state changes would be impractical for the size of object collections that we envision—which is the underlying motivation for our requirement that the catalog should be redundancy-free and contain only time-invariant information.

Information from the organization chart of an enterprise is typical of what is in the environment store that may be used to describe certain aspects of information objects. For example, let us assume that d was the managing director of the organization unit u_1 at time t_1 when a certain object o was created by employee e_1 in the organization unit u_{11}, which was at that time a part of the organization unit u_1. It is easy to imagine situations in which one might be interested in identifying all objects of a certain type (say, contracts worth more than one million dollars) that had been created, in a certain period of time, in all organization units directed by d. One might therefore be tempted to store d as an attribute of o in the catalog entry that describes o. However, this would probably have to be repeated in many catalog entries and would lead to the well-known problems related to redundant information. The idea to avoid redundant information by storing d as a family descriptor for *contracts* which would then be shared by all objects in that family also fails, because we do not want to define a new family of contracts whenever a new director is assigned to manage u_1. More generally, it is impossible to know in advance all possible ways in which users may wish to identify objects.[61] It is, therefore, best to normalize all descriptive information kept about objects in the same way as it is traditionally done for relational databases. Based on these insights, all descriptive information that would violate normalization rules is stored external to the catalog. In our example, it suffices to store, in the catalog, the fact that o was created at time t_1 by employee e_1. The fact that e_1 was then assigned to u_{11}, that u_{11} was a part of u_1, and that d was then the managing director of u_1 is kept in the environment store.

The environment store contains time-stamped information about the state of the environment.

The historic dimension of the environment store also allows BOMS to bridge schema evolutions, i.e., to provide users with a stable view of object characteristics even when the structure and representation of certain descriptors changes over time. Through an automatic mapping of object queries to multiple descriptor schemas that correspond to multiple historic periods, users are shielded from the possibility that certain object characteristics may have been represented differently during multiple periods of time. Consider, for example, a case where a descriptor CUSTOMER_NUMBER had for some time been represented as BIDnnnnnn, where BID was the identifier of the branch with which the customer had a business relationship, and where nnnnnn was a number that was unique across the enterprise. Searching for objects related to a certain customer across all branches of the enterprise therefore involves wild card clauses of the form

```
WHERE CUSTOMER_NUMBER ='$nnnnnnn'          (1)
```

where $ is the wild card character that matches all branch identifiers.

Now, let us assume that, at some point in time, it is decided that the enterprise would like to be able to better manage its business relationships with all types of involved parties. Therefore, the enterprise may choose to introduce a new, enterprise-wide descriptor INVOLVED_PARTY_NUMBER that will supersede the CUSTOMER_NUMBER. The new descriptor INVOLVED_PARTY_NUMBER will be of the form mmmmmm, i.e., it will no longer include a reference to a particular branch. However, because the identifier of the branch where certain business transactions are handled is still considered to be important, a new descriptor BRANCH_IDENTIFIER is introduced. For new objects, which are described according to the new scheme, searches related to a certain customer now require clauses of the form

```
WHERE INVOLVED_PARTY_NUMBER ='mmmmmmm'     (2)
```

However, if a user searches for older objects with descriptors of the older type, BOMS will automatically map the query to the old form in Equation 1 with the wild card search across all branches. In general, the user does not have to be aware of the possibility that older objects may have been described according to schemas that were different from those commonly used today. Moreover, there is no need for the user to be aware of the time when the new descriptor types were introduced. BOMS can infer all that from the historic dimension of the environment store and map the external user query into a series of BOMS-internal queries that will retrieve all relevant objects, even if different descriptor types were used during multiple historic periods. The motivation for having this activity transparent to the user is threefold: (1) Schema evolutions are inevitable; we cannot ignore changes in the business environment and must, therefore, be able to adapt descriptor schemas so that they always reflect current reality. (2) We want to provide the users with a consistent view and avoid the need for users to be aware of, and understand, the consequences of schema evolutions. (3) We want to avoid the need to modify the descriptors (i.e., catalog entries) of stored objects to reflect schema evolutions. For the size of object collections that we envision, and given the requirement that BOMS must be almost continuously available for user queries (close to 7×24 hours), it would be impractical.

Querying, retrieving, and organizing business objects. The ability to identify and retrieve objects that are relevant in a given business context depends on being able to describe the properties that separate relevant from irrelevant objects. Together, the catalog and the environment store allow associative queries that can take into account not only properties inherent to the objects themselves, but also the state of the environment in which the objects were created. Because both the catalog and the enterprise store are designed as collections of relational tables, and because of the flexibility of the relational model, interobject relationships do not have to be predefined. The limitations of standard approaches to structure document, image, and object databases with relatively few and predefined links from individual documents to index terms are avoided. Instead, with the relational join operator, objects can be dynamically related based on attribute values; i.e., access is associative through a value-based, nonprocedural specification of a collection of relevant information objects. Consequently, searching for information closely matches the activity of the human mind, which is inherently associative. More specifically, there is no need for static, predefined and hard-coded links between objects to define folders, for example, to keep all objects together that belong to a certain organization unit in the enterprise. Folders can be defined dynam-

ically and separately by each user by specifying in a query what common properties all objects in a folder should have. In fact, a BOMS folder is simply made up of one or more lists of objects that share certain characteristics.

Object queries allow users to work with information objects in the same way in which they are used to working with paper documents. Objects that meet the selection criteria of a query can be assigned to private subject folders, and folders can be subdivided with file tabs. New objects can be added to existing folders. For example, with the push of a button, a user can request that all new objects that entered BOMS since the last execution of a query and that meet the selection criteria of the query are added to the current content of a folder. It is an easy and powerful way to keep subject folders up to date. Moreover, objects can be copied or moved between folders, or from one file tab to another. Entire file tabs can be moved or copied between folders. However, this action affects only a particular user's view of these objects. The objects themselves, and all other users' views of these objects, remain unchanged by such operations.

The first step in a typical sequence of interactions with BOMS is to set up a query.

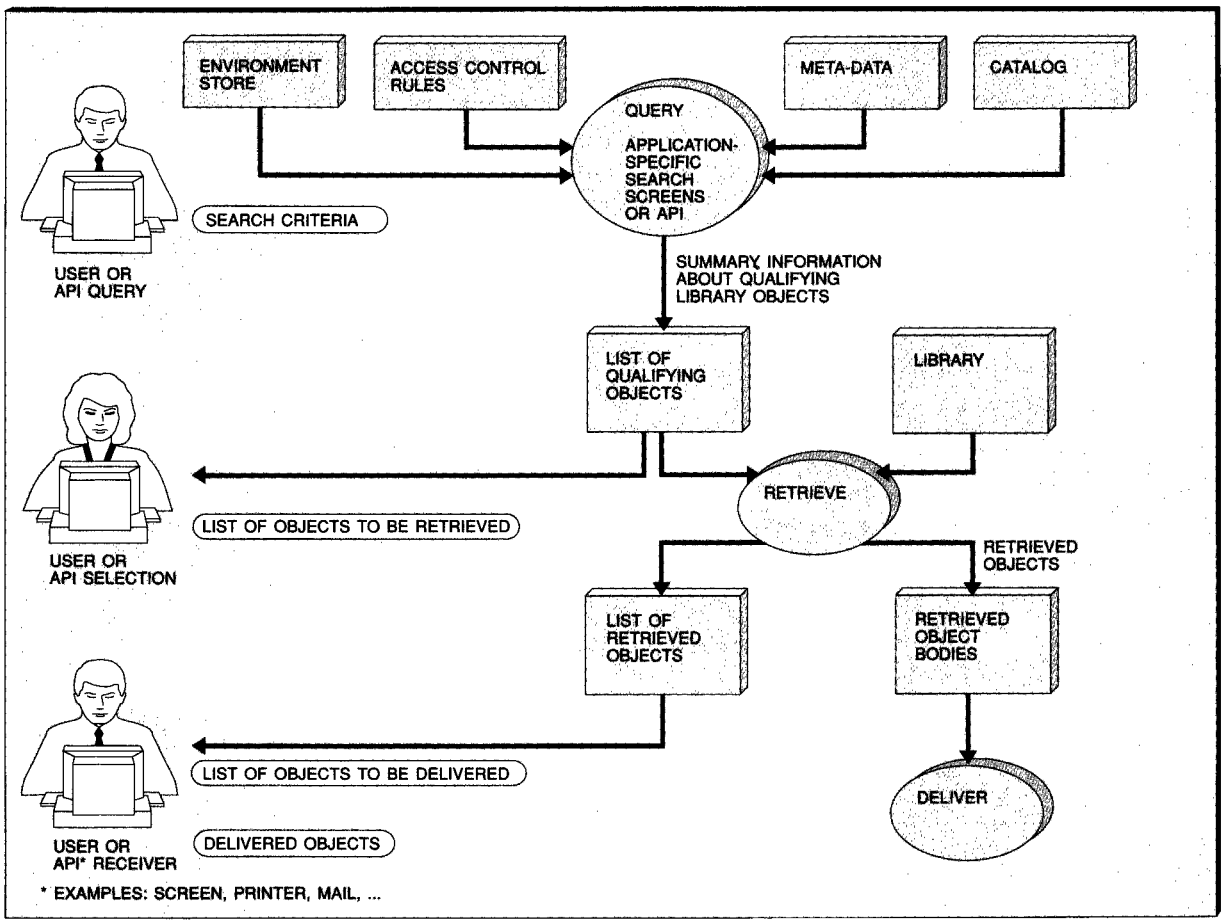
Figure 8 illustrates the main processes perceived by a BOMS user: *query*, *retrieve*, and *deliver*. Also shown are the data flows into and out of these processes. The query process is implemented by the service request manager; the implementations of the retrieve and deliver processes are distributed across the service request and the library manager (see Figure 7). The actions of the calling user or client process are represented by the colored arrows in Figure 8.

The first step in a typical sequence of interactions with BOMS is to set up a query, either by calling the BOMS application programming interface (API) with the search criteria as parameters (typically done by a client program) or by interactively specifying the search criteria in application-specific panels on a terminal. In that way, a conjunction of predicates on the attributes of the BOMS information objects is created to act as a filter through which the collection of information objects is presented to the user. The query process applies that filter to the catalog and to the environment store and returns a *list of qualifying objects* to the user. For each object that matches the specified search criteria, provided the user is authorized to know that the object exists (according to the access control rules), the list of qualifying objects contains a separate entry. That entry contains selected elements from information kept in the catalog about the corresponding object. What elements are in the catalog depends on the family to which the object belongs. From these elements the BOMS application designer can select the elements that are included in the list. In that sense, the list of qualifying objects is a dynamic folder with (references to) objects that pertain to the business issue or the question that was the origin of the query. The entries in the folder can come from multiple object families and from objects that are encoded in different data stream formats.

In many cases, the list of qualifying objects is sufficient to meet immediate information needs. For example, when customer 0987.654321 (Figure 6) calls to inquire about the quantity and the price of the securities that the bank had sold for him or her on January 17, 1994, normally retrieval of the complete settlement statement from the library is not required (see Figure 6). The list of qualifying objects will contain an entry that corresponds to the settlement statement for order number 123456789, and the entry will contain the vector descriptor {security number, quantity, price, currency, amount}. If the customer is satisfied with that information, no further action is required. Only if the customer insists on a copy of the settlement statement must the object itself be retrieved from the library and printed.

In that case, the user could then select the entry that corresponds to the settlement statement 123456789 from the list of qualifying objects, and specify that BOMS should *retrieve* the corresponding object from the library and deliver it to the user's local printer or, optionally, to the bank's

Figure 8 Main processes perceived by a BOMS user



automated print and mail factory, which would then print the statement, put it into an envelope, and place it in the external mail. More generally, the BOMS service request manager will deliver the retrieved object to the service reply queue, where it will be picked up by other processes, such as a print process or a process that displays the object on the user's screen for interactive browsing, with subsecond response time for page-up or -down operations. Because the retrieve process is asynchronous and because the retrieved objects are put into the service reply queue (see Figure 7), which is persistent, the user can submit a retrieval request and then return to some other work or even log off from the system, resuming work later with the retrieved (set of) object(s).

An important aspect is that users can initiate operations against entries in the list of qualifying objects, such as print or browse, without knowing the format in which the object is represented. Irrespective of whether it is an ASCII file, an RFT document, an image, or anything else, such requests will call the appropriate browse or print programs.

Another aspect is that users can build upon the results of a query, i.e., use the descriptor values of objects that meet the selection criteria of one query to formulate a subsequent query. For example, imagine a user who has issued a query about a customer and an account number. One of the returned objects may be an account statement in which a particular line item arouses the user's interest. The user can now trace the information flow forward and backward, for example, by issuing queries that search for related information about events that either preceded *and* followed or

preceded *or* followed the event that is represented by that particular line item. That related information may include different types of objects such as the corresponding accounting voucher and the original order, which may be a transaction record that was received through S.W.I.F.T., an e-mail message, or the scanned image of a letter. However, this navigation through related object families is transparent to the user. The capability to link information objects of different types and from different sources appears as a kind of hypertext facility allowing related pieces of information to be identified and retrieved in a unified and coherent way. Although this procedure is initially step-wise and iterative, the results can be accumulated in a folder for subsequent immediate access to the combined set of retrieved objects.

Access control. A fundamental problem with most current access control models is that they do not support direct, high-level specification of the access control policies of an enterprise. Instead, it is left to administrators to define low-level controls that they (the administrators) consider to be suitable for enforcing the policies. Consequently, with most current models, there is an inevitable semantic gap between the access control policies of an enterprise and their implementation and enforcement. For example, with available models, the following instances of least privilege and "separation of duties" cannot be specified and enforced directly:

"No system programmer must ever update information objects that are members of family F_{xy} (see Figure 5), irrespective of where in the storage hierarchy such an object might currently be."

"A person can access information objects only when the organizational source of the object (see Figure 5) corresponds to the organizational unit in which the person works. Exceptions to this rule are object families $F_{x,y}$, where, for example x,y = {(2,3),(5,5),(9,2)}. For these families, more permissive rules are allowed."

Instead, these and similar policies depend on procedures and administrative controls (e.g., approvals) that are external to the access control model—external in the sense that they are usually documented in procedure manuals but depend on humans to interpret and enforce them. In other words, the policies are not in a form that would allow a computer to interpret and enforce them.

A general problem with such external procedures is that they are often difficult to verify and to enforce. In addition, when the size of object collections reaches the orders of magnitude that we foresee for BOMS implementations, such access control schemes are no longer practical. The expected number and frequency of personnel, organizational, and environmental changes in an enterprise during the lifetime of an object (up to 10 and more years) would lead to a prohibitive administrative workload (due to the number and complexity of administrative decisions and actions required to enforce the security of policies and their changes over time). Alternatively, it would lead to cases where users are, often unknowingly, granted more rights than are justified by business requirements. In other words, there would be a growing risk of compromising (i.e., failing to enforce) existing "least privilege policies," as a consequence of the need to keep the security administration workload at an acceptable level.

Consequently, BOMS provides an access control scheme that allows the direct specification and enforcement of policies, as in the examples above, in order to make access control easier to understand and verify and to reduce the administrative workload.

BOMS access control is an extension and generalization of current mandatory access control as commonly used in military defense applications. Mandatory access control normally relates object classification labels of the type top secret, secret, etc., to the users' clearance levels and is based on rules such as:

"Users can access only objects with classification labels that are equal to, or less than, the users' clearance."

For example, users with top secret clearance can access top secret and secret objects, but users with secret clearance can access only secret objects.

BOMS access control is similar in that it does not put access control into the hands of individual users. However, BOMS access control does not rely on the standard classification labels and user clearance levels but allows enterprise-specific rules that can refer to arbitrary object and user characteristics and to information about the envi-

ronment. In particular, the access control rules can refer to object family attributes and to attribute values of individual objects. These rules are specified in a declarative language that is intuitive and close to the way in which people think about these policies—and that can be interpreted and enforced by the computer.

The BOMS usage of these rules is twofold:

1. The service request manager joins the access control rules with the selection predicates from the service request to limit the number of entries in the list of qualifying objects (see Figure 8). For example, if there is a general access control rule in place that prevents users from accessing certain information from outside their own departments, the list of qualifying objects will contain only entries about information from their own departments, even if the users' search criteria did not specify that restriction. This rule prevents careless (or malicious) users from impacting the BOMS performance by issuing service requests with insufficiently qualified search criteria, possibly leading to thousands of irrelevant objects being retrieved. An interesting side effect of that concept is that it also allows the use of security policies that prevent users from knowing whether certain information exists.
2. When the library manager returns the retrieved object bodies from the libraries, the access control rules are applied to the object descriptors. Only those objects that pass the test are delivered to the requesting user's service reply queue (see Figure 7).

For run-time performance reasons, the rule-based representation of access control policies is mapped (i.e., compiled) into a tabular form similar to traditional access lists. The important point, however, is that this mapping is done mechanically and not left to the discretion of administrators. Moreover, whenever changed circumstances require it, access control policies can be adapted easily and quickly, using the declarative high-level language. Without further administrative overhead, the changes will then be reflected mechanically in the tabular, compiled run-time form of the access control rules.

Practical implementation of BOMS as a distributed resource manager

BOMS is designed and implemented as a CICS/ESA-based resource manager, i.e., both the BOMS service request manager and the library manager are implemented as a set of CICS* programs. The catalog, the environment store, and the meta-data store (see Figure 7) are implemented as sets of DB2 tables. The creation of these tables is adaptive in the sense that the family classification scheme of a particular BOMS installation and the particular enterprise-specific search terms are input to a semiautomated table creation process. In the first operational release, the object libraries have also been implemented with DB2, but, in principle, any existing or emerging storage technology can be used, for example, OAM (object access method) and optical libraries or emerging specialized BLOB (binary large object) servers.

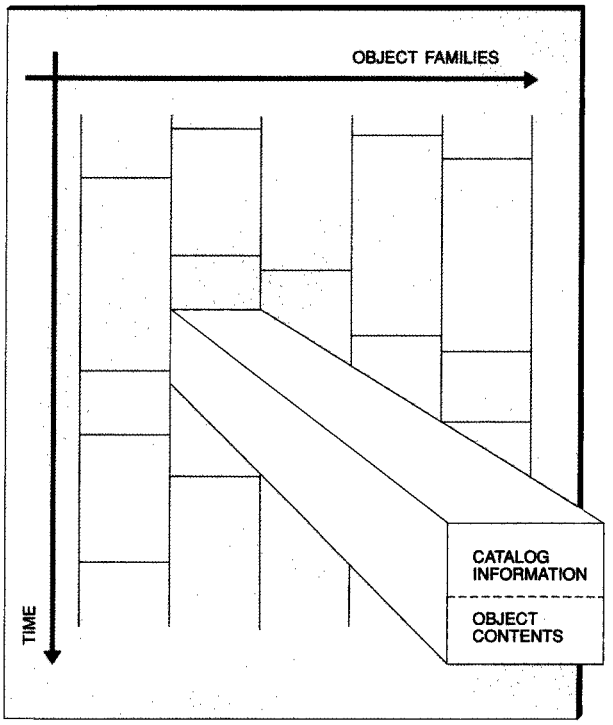
Irrespective of how the object libraries are implemented, and even when libraries of different types are mixed, users perceive the complete BOMS storage as a single, virtually boundaryless storage plane. Information objects meeting the criteria of a search or retrieval request may be distributed across multiple libraries that may, in turn, again be distributed across multiple nodes, but any such physical segmentation is completely invisible to the user. Access to information objects and navigation between them is provided exclusively on the basis of descriptors pertinent to the user's business view.

In order to allow multiple nodes to cooperate transparently, BOMS uses an integrated routing and coordination facility and a node routing table. Incoming service requests are split up into separate node requests when the service request manager determines, during the initial screening of a service request, that remote nodes contain libraries with potentially qualifying objects. The replies to these separate node requests are reported back to the source node, where they are consolidated and coordinated into one complete reply to the original service request. Node routing may be based on descriptor value ranges such as family, organizational source (branch office, department, etc.), data stream format, element type, status (production, training, test, etc.), or time period when the object was stored. Over time, the distribution of objects in a network of BOMS nodes can be changed without disturbing the users' sin-

gle-system view. New nodes can be added, and existing nodes can be "stabilized," in the sense that no more new objects are stored in these nodes. In this way BOMS installations can keep existing object collections and begin to store new objects in nodes that are implemented with newer, more cost-effective storage technology and still maintain the users' single-system view of both old and new collections. In that sense, BOMS is an integrator for different library technologies that will inevitably emerge over time.

A feature exclusive to BOMS (according to our knowledge) is that the size of object collections and the query performance are not limited by a requirement to keep the catalog information for all objects on line; similarly it is not required to keep all historic layers of the environment store on line. This arrangement is achieved by further segmenting the parts of the virtual storage plane covered by individual libraries into self-sufficient logical units of data (LUD), which contain well-defined sets of information objects, together with all the accompanying catalog and environment information that is required to access the objects and to navigate between them (see Figure 9). Consistent with the concept of a two-dimensional storage plane, the segments are rectangular areas containing the objects from individual *families* from a certain period. As Figure 9 illustrates, the splits along the time axis can be at irregular intervals. For example, a split can be made when the number of objects in an LUD reaches a certain, family-specific value, or when an LUD exceeds the capacity or performance limitations of a database management system. In any case, access to individual LUDs is through a pointer in the meta-data (see Figure 7), but this additional level of indirection is invisible to the user. If such a pointer is found to point to an LUD that is off-loaded to secondary storage, the BOMS service request manager automatically makes a call to the appropriate storage subsystem to bring the LUD on line again. In other words, when a query refers to an LUD that is off line, all of this action is handled transparently—very much like a "soft page fault" is handled by a virtual memory operating system reading missing pages from secondary storage into main memory in a manner transparent to the application program that made a reference to a memory location that was found to be "paged out." In the first operational release, each LUD is implemented as a separate DB2 database, i.e., a related collection of table spaces and their indexes, used together as an operational unit for starting and stopping all accesses and for off-loading to secondary storage.

Figure 9 Logical units of data in virtual storage plane



Because the amount of meta-data that must be kept on line to point to off-loaded LUDs and to the relevant slices of the environment store is orders of magnitude smaller than a typical LUD catalog, that approach allows BOMS collections to grow to orders of magnitude impossible with systems that rely on the concept that all catalog information for all objects must always be on line. In particular, the LUD concept decouples the size of BOMS object collections from the amount of data a given database management system can hold. In addition, we can limit the size of catalog and environment store portions so that we can keep the performance of relational queries and joins at an acceptable level. By adding a layer of software above traditional relational database management systems, we obtain the freedom to exploit evolving database management systems, but we also become, to a large extent, independent of product cycles.

A productive prototype with over 200 users and more than nine million documents was running successfully from 1989 until 1991. In October 1991, a production release was put into operation, and at the beginning of 1992, more than 10 million annual account closing statements (20 gigabytes) were stored; during that year, approximately two million objects were stored each month. In December 1992 we had a total of 40 million objects, and by January 1994 we had 95 million objects (220 gigabytes) in our BOMS implementation. Many objects are relatively small, typically 2000 bytes, but we have also some very large objects; the largest objects exceed 10000 pages. During the first days of 1994 we had once again over 10 million annual closing statements (24 gigabytes), which were inserted in less than two calendar days. Since January 1994 we have been storing 500000 new objects (1 gigabyte) every day, and we expect to have a total of at least 200 million objects by the end of 1994. The main reason for the steep increase in 1994 is that we have now begun to store statements generated by the payments application. The number of users has been growing steadily; on an average day we now have 600 users who generate approximately 4000 service requests. Most of the queries are complex, and the average response time of 5 to 10 seconds meets all practical user requirements. With the availability of the payments transaction statements, and with other applications gradually beginning to rely exclusively on BOMS to provide their users with access to historic data, we expect a significant increase in the number of users and the number of service requests. The potential number of internal users is on the order of 10000, and if customers of the bank were also allowed to directly access BOMS, that number could grow to even larger orders of magnitude. The maximum number of users that BOMS can support is limited only by the number of nodes and the number of CICS regions per node.

Conclusion

BOMS is based on, and extends, concepts from the relational database model. A resultant practical advantage is that relationships between objects can be established dynamically, based on attribute values, i.e., interobject relationships do not have to be predefined. When users search and collect information, they can dynamically define their own folders to meet the needs of a given situation. This method contrasts to alternative approaches requiring static folders, in which documents are put into folders when they are stored, often using hard-coded links. Because the BOMS folders are orthogonal to the BOMS object families, they can be defined independently by each user, according to the criteria that are relevant at the time the information is needed, rather than according to the criteria that appear to be important at the time when the information is stored. Thus, BOMS eliminates the dilemma of trying to guess in advance the contexts in which information could become useful in the future—a futile task indeed when one considers that information stored in BOMS has a useful lifetime of ten or more years.

Because the profile of an object contains only descriptors that are time-invariant and unique to the object, BOMS has a fundamental advantage over more simplistic approaches not having the concept of a separate environment store. The environment store allows us to maintain, redundancy-free, multiple historic versions of time-varying information potentially relating to multiple objects. We can, therefore, join information about individual object instances with information about the environment in which an object was created. Joining allows queries to arise that would otherwise be impossible to answer in collections of the size we envision.

Because of a clear separation between the BOMS application and system layers, based on client/server and resource manager principles, all implementation complexities are hidden from the users. Despite the inevitable underlying complexity and heterogeneity, users are provided with a single-system view.

The workstation provides a single point of access to what appears (to the user) to be an integrated set of information and processing resources. However, these resources can actually reside on a variety of platforms. Users are given a conceptual view of information objects, without having to know where and how these objects are stored.

BOMS is designed to provide the flexibility to exploit and introduce any convenient storage technology whenever it is cost-effective, without disturbing the single-library view. BOMS can act as an integrator of different library types, different (heterogeneous) database management systems, and different hardware and systems software plat-

forms. In particular, it removes most dependencies from the capability of a database management system to transparently manage the vast amount of information that will inevitably accumulate over the years. By transparently splitting object families into logical units of data, which can be held in separate DBMS instances, BOMS circumvents any potential limitation on the maximum amount of data that a given database management system can accommodate.

BOMS is a conceptual platform with a pragmatic implementation that can transform organizational structures and support new ways of making decisions. Because BOMS requires, or at least encourages, an enterprise to define a common, unified terminology to describe its information assets, it can be a catalyst and enabling platform to integrate otherwise isolated parts of an enterprise. The common terminology with BOMS as a powerful means to share common information can help to bridge potential semantic gaps that prevent effective communication and mutual understanding in a large enterprise. By adopting the BOMS concepts, enterprises can prepare to be able to filter and interpret ever increasing amounts of heterogeneous information in new ways that reflect new and changed situations. In that sense, we hope that BOMS is a contribution toward one of the most urgent mandates of our time, i.e., learning to thrive on chaos.[62]

In the future, we plan to further investigate possibilities to integrate different library types, including ImagePlus, and ways to provide users with a single-system view of ODA (office document architecture) and SGML (standard generalized markup language) document collections.

Acknowledgments

For many years, Henry Gladney from IBM Research has always been a stimulating discussion partner. We also acknowledge the feedback from Hans Keller; his questions have helped us to more clearly articulate what we believe to be unique to BOMS.

Cited references

1. C. J. Date, *An Introduction to Database Systems*, Volume I, Addison-Wesley Publishing Co., Reading, MA (1981).
2. C. J. Date, *An Introduction to Database Systems*, Volume II, Addison-Wesley Publishing Co., Reading, MA (1983).
3. D. J. Haderle and R. D. Jackson, "IBM Database 2 Overview," *IBM Systems Journal* **23**, No. 2, 112–125 (1984).
4. P. P. S. Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems*, No. 1, 9–36 (March 1976).
5. *Entity Relationship Approach to Systems Analysis and Design*, P. P. Chen, Editor, North-Holland, Amsterdam (1980).
6. *Entity Relationship Approach to Information Modeling and Analysis*, P. P. Chen, Editor, ER Institute, Saugus, California 91350 (1981).
7. M. L. Brodie, "On the Development of Data Models," in *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, Editors, Springer, New York (1984), pp. 19–48.
8. R. W. Matthews and W. C. McGee, "Data Modeling for Software Development," *IBM Systems Journal* **29**, No. 2, 228–235 (1990).
9. J. E. Gessford, *How to Build Business-Wide Databases*, John Wiley & Sons, Inc., New York (1991).
10. V. Linnemann, K. Küspert, P. Dadam, P. Pistor, R. Erbe, A. Kemper, N. Südkamp, G. Walch, and M. Wallrath, "Design and Implementation of an Extensible Database Management System," *Proceedings of the 14th VLDB Conference*, Los Angeles (1988), pp. 294–305.
11. G. Gardarin, J. P. Cheiney, G. Kiernan, D. Pastre, and H. Stora, "Managing Complex Objects in an Extensible Relational DBMS," *Proceedings of the Fifteenth International Conference on Very Large Databases*, Amsterdam (1989), pp. 55–65.
12. M. Stonebraker, "Future Trends in Database Systems," *IEEE Transactions on Knowledge and Data Engineering* **1**, No. 1, 33–44 (1989).
13. L. M. Haas, W. Chang, G. M. Lohmann, J. McPherson, P. F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. Carey, and E. Shekita, "Starburst Mid-Flight: As the Dust Clears," *IEEE Transactions on Knowledge and Data Engineering* **2**, No. 1, 143–160 (March 1990).
14. D. C. Blair, "The Data-Document Distinction in Information Retrieval," *Communications of the ACM* **27**, No. 4, 369–374 (April 1984).
15. C. D. Blair, *Language and Representation in Information Retrieval*, Elsevier, Amsterdam (1990).
16. N. J. Belkin and W. B. Croft, "Information Filtering and Retrieval: Two Sides of the Same Coin?" *Communications of the ACM* **35**, No. 12, 29–38 (December 1992).
17. F. W. McFarlan, "Information Technology Changes the Way You Compete," *Harvard Business Review*, 98–103 (May–June 1984).
18. M. E. Porter and V. E. Millar, "How Information Gives You Competitive Advantage," *Harvard Business Review*, 149–160 (July–August 1985).
19. *Information Management: The Strategic Dimension*, M. Earl, Editor, Clarendon Press, Oxford (1989).
20. E. K. Clemons, "Strategic Investments in Information Technology," *Communications of the ACM* **34**, No. 1, 22–36 (January 1991).
21. J. M. Kerr, *The IRM Imperative: Strategies for Managing Information Resources*, John Wiley & Sons, Inc., New York (1991).

22. S. Loeb and D. Terry, "Information Filtering," *Communications of the ACM* **35**, No. 12, 26–28 (December 1992).
23. C. Stevens, "Automating the Creation of Information Filters," *Communications of the ACM* **35**, No. 12, 48 (December 1992).
24. I. Stadnyk and R. Kass, "Modeling Users' Interests in Information Filters," *Communications of the ACM* **35**, No. 12, 49–50 (December 1992).
25. P. E. Baclace, "Competitive Agents for Information Filtering," *Communications of the ACM* **35**, No. 12, 50 (December 1992).
26. P. W. Foltz and S. T. Dumais, "Personalized Information Delivery: An Analysis of Information-Filtering Methods," *Communications of the ACM* **35**, No. 12, 51–60 (December 1992).
27. D. Goldberg, D. Nichols, T. Hickey, K. C. Lee, W. H. Mansfield, J. Ratz, and A. Weinrib, "Using Collaborative Filtering to Weave an Information Tapestry," *Communications of the ACM* **35**, No. 12, 61–70 (December 1992).
28. *Information Warehouse Architecture I*, SC26-3244, IBM Corporation (1993); available through IBM branch offices.
29. S. Khoshafian, *Object-Oriented Databases*, John Wiley & Sons, Inc., New York (1993).
30. W. Kim and F. H. Lochovsky, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publishing Co., Reading, MA (1989).
31. W. Kim, F. Garza, N. Ballou, and D. Woelk, "Architecture of the ORION Next-Generation Database System," *IEEE Transactions on Knowledge and Data Engineering* **2**, No. 1, 109–124 (March 1990).
32. W. Kim, *Introduction to Object-Oriented Databases*, The MIT Press, Cambridge, MA (1991).
33. A. Otis and J. Stein, "The GemStone Object Database Management System," *AIXpert*, 54–58 (May 1992); available through IBM branch offices as Order No. G580-0010-00.
34. T. A. Andrews, "ONTOS DB: An ODBMS for Distributed AIX Applications," *AIXpert*, 59–62 (May 1992); available through IBM branch offices as Order No. G580-0010-00.
35. D. K. Barry, "ITASCA Distributed ODBMS," *AIXpert*, 63–67 (May 1992); available through IBM branch offices as Order No. G580-0010-00.
36. G. Landis, "Overview of the ObjectStore ODBMS," *AIXpert*, 68–72 (May 1992); available through IBM branch offices as Order No. G580-0010-00.
37. M. E. S. Loomis, "The VERSANT ODBMS," *AIXpert*, 73–76 (May 1992); available through IBM branch offices as Order No. G580-0010-00.
38. K. Parsaye, M. Chignell, S. Khoshafian, and H. Wong, *Intelligent Databases: Object-Oriented, Deductive Hypermedia Technologies*, John Wiley & Sons, Inc., New York (1989).
39. M. Stonebraker, L. A. Rowe, and M. Hirohama, "The Implementation of POSTGRES," *IEEE Transactions on Knowledge and Data Engineering* **2**, No. 1, 125–142 (1990).
40. E. McKenzie and R. Snodgrass, "Schema Evolution and the Relational Algebra," *Information Systems* **15**, No. 2, 207–232 (1990).
41. R. Maiocchi and B. Pernici, "Temporal Data Management Systems: A Comparative View," *IEEE Transactions on Knowledge and Data Engineering* **3**, No. 4, 504–524 (December 1991).
42. J. F. Roddick and J. D. Patrick, "Temporal Semantics in Information Systems: A Survey," *Information Systems* (UK) **17**, No. 3, 249–267 (May 1992).
43. L. Cabrera, L. Haas, J. Richardson, P. Schwarz, and J. Samos, "The Melampus Project: Towards an Omniscient Computing System," Research Report RJ7515, IBM Corporation, San Jose, CA (June 1990).
44. E. Messinger, K. Shoens, J. Thomas, and A. Luniewski, "Rufus: The Information Sponge," Research Report RJ8294, IBM Corporation, San Jose, CA (August 1991).
45. G. P. Michalski, "The World of Documents," *Byte* **16**, No. 4, 159–170 (April 1991).
46. D. A. Harvey, "Catch the Wave of DIP," *Byte* **16**, No. 4, 173–182 (April 1991).
47. D. A. Harvey and B. Ryan, "Practically Paperless," *Byte* **16**, No. 4, 185–190 (April 1991).
48. C. Locke, "The Dark Side of DIP," *Byte* **16**, No. 4, 193–204 (April 1991).
49. S. Diehl and H. Eglowstein, "Tame the Paper Tiger," *Byte* **16**, No. 4, 220–243 (April 1991).
50. L. C. Kingman III, R. E. Lambert, and R. P. Steen, "Operational Image Systems: A New Opportunity," *IBM Systems Journal* **29**, No. 3, 304–312 (1990).
51. B. Trammell, "Too Little, Too Late? Not at USAA," *Inform* **3**, No. 8, 24–52, Association for Information and Image Management (AIIM), AIIM, 110 Wayne Avenue, Suite 1100, Silver Spring, MD 20910 (August 1989).
52. C. D. Avers and R. E. Probst, "ImagePlus as a Model for Application Solution Development," *IBM Systems Journal* **29**, No. 3, 356–370 (1990).
53. H. M. Gladney, "Requirements Analysis for a Document Storage Subsystem," Research Report RJ7085, IBM Corporation, San Jose, CA (1989).
54. H. M. Gladney, R. M. Cubert, D. B. Hildebrand, J. Kleck, R. W. Schmiedeskamp, J. Antognini, and S. F. Horne, "Architecture and Design of a Document Storage Subsystem," Research Report RJ7223, IBM Corporation, San Jose, CA (1989).
55. H. M. Gladney, D. B. Hildebrand, and R. W. Schmiedeskamp, "External Design of a Document Storage Subsystem," Research Report RJ8267, IBM Corporation, San Jose, CA (1991).
56. H. M. Gladney, "A Storage Subsystem for Image and Records Management," Research Report RJ8626, IBM Corporation, San Jose, CA (February 1992).
57. H. M. Gladney, "A Storage Subsystem for Image and Records Management," *IBM Systems Journal* **32**, No. 3, 512–540 (1993).
58. *Image and Records Management (IRM) General Information Guide*, GC22-0027, IBM Corporation (1992); available through IBM branch offices.
59. *Image and Records Management (IRM) Planning and Installation Guide*, GC22-0029, IBM Corporation (1992); available through IBM branch offices.
60. *IBM SAA Image and Records Management (IRM) Baseline User's Guide*, SC22-0031, IBM Corporation (1992); available through IBM branch offices.
61. D. Erickson, "Hacking the Genome," *Scientific American*, 98–105 (April 1992).
62. T. Peters, *Thriving on Chaos*, Alfred A. Knopf, New York (1988).

Accepted for publication January 31, 1994.

Marcel Schlatter *IBM Switzerland, Hohlstrasse 600, CH-8048 Zurich, Switzerland (electronic mail: Internet: CHIBMT69@IBMMAIL.COM; Inter-enterprise: CHIBMT69 at IBMMAIL).* Dr. Schlatter is a solution development specialist in the Field Business Unit "Banking and Insurance" of IBM Switzerland. Since he joined IBM in 1982, he has spent most of his time working with various financial institutions, covering a broad spectrum of information technology issues. During 1988–89, he was assigned to the IBM European headquarters in Paris. In that capacity, he worked with a diverse set of customers to define and articulate the information security requirements of the banking and insurance sectors, the manufacturing, and the defense industry sectors. In addition to document management systems, his current professional interests include client/server application architectures and message-driven processing. Prior to joining IBM, his research focused on communication technology. He published papers in the *IEEE Transactions on Communications* and in the *IEEE Transactions on Information Theory*. Dr. Schlatter received an M.Sc. in electrical engineering in 1974 and a Ph.D. (Dr. sc. techn.) in 1981 from the Swiss Federal Institute of Technology in Zurich (ETH), Switzerland. He is a member of the IEEE.

René Furegati *IBM Switzerland, Hohlstrasse 600, CH-8048 Zurich, Switzerland (electronic mail: Internet: CHIBMVMP@IBMMAIL.COM; Inter-enterprise: CHIBMVMP at IBMMAIL).* Mr. Furegati is a solutions architect in the Field Business Unit "Banking and Insurance" of IBM Switzerland. He joined IBM Switzerland in 1966. Having accompanied Credit Suisse's and other large Swiss banks' highly integrated transaction systems along their entire development cycle, beginning in the late 1960s, he shifted interests toward handling documents with the help of information technology in the mid-1980s. Emphasis has been on the interaction of short-term local solutions with long-term corporate document management aspects in environments with potentially tens of thousands of users and billions of documents to be kept and reproduced or reworked over many decades.

Franz J. K. Jeger *IBM Switzerland, Hohlstrasse 600, CH-8048 Zurich, Switzerland (electronic mail: Internet: CHIBMSJP@IBMMAIL.COM).* Dr. Jeger is a solution development specialist in the Field Business Unit "Banking and Insurance" of IBM Switzerland. He joined IBM Switzerland in 1978. His recent work includes the development of a methodology to define user requirements for document management systems. He received an M.A. in mathematics in 1973 and a Ph.D. (Dr. sc. math.) in 1977 from the Swiss Federal Institute of Technology in Zurich (ETH), Switzerland.

Heinrich Schneider *Credit Suisse, Oq8, CH-8070 Zurich, Switzerland (electronic mail: Internet: heinrich.schneider@cs.csh.com; Inter-enterprise: CHSKARF9 at IBMMAIL).* Mr. Schneider is responsible for the architecture of middleware software at Credit Suisse and is consultant for application development projects. Since he joined Credit Suisse in 1982, he has spent his time working on different projects, covering a broad spectrum of information technology issues. He is experienced in different areas, among them system programming, system-/application-analysis and design, office automation, application architecture issues, data and function modeling, and project management. He was project leader for the development of the enterprise-wide Business Object Management System at Credit Suisse. His current professional interests include application design based on messaging technology and object-oriented development combined with the client/server model. Recently he worked for several months in the Santa Teresa Laboratory of IBM. Prior to joining Credit Suisse, he graduated from the Swiss Federal Institute of Technology in Zurich (ETH) and received an M.Sc. in physics. He is currently working toward an M.B.A. from the City University Zurich.

Heinrich A. A. Streckeisen *IBM Switzerland, Hohlstrasse 600, CH-8048 Zurich, Switzerland (electronic mail: Internet: CHIBMLLK@IBMMAIL.COM; Inter-enterprise: CHIBMLLK at IBMMAIL).* Mr. Streckeisen is a senior systems engineer and works as a solution development specialist in the Field Business Unit "Banking and Insurance" of IBM Switzerland. He joined IBM in 1964, first working with customers in the manufacturing, distribution, and insurance industries. Since 1986 he has worked in the ongoing BOMS project of Credit Suisse and is the lead designer from the IBM side, concentrating on overall design and CICS as well as DB2 implementation aspects.

Reprint Order No. G321-5541.