Collection-Based Persistent Digital Archives - Part 1

Reagan Moore, Chaitan Baru, Arcot Rajasekar, Bertram Ludaescher, Richard Marciano, Michael Wan, Wayne Schroeder, and Amarnath Gupta

San Diego Supercomputer Center

[This is the first of a two-part story. The second part will appear in the April 2000 issue of D-Lib Magazine.]

Abstract

The preservation of digital information for long periods of time is becoming feasible through the integration of archival storage technology from supercomputer centers, data grid technology from the computer science community, information models from the digital library community, and preservation models from the archivist’s community. The supercomputer centers provide the technology needed to store the immense amounts of digital data that are being created, while the digital library community provides the mechanisms to define the context needed to interpret the data. The coordination of these technologies with preservation and management policies defines the infrastructure for a collection-based persistent archive [1]. This paper defines an approach for maintaining digital data for hundreds of years through development of an environment that supports migration of collections onto new software systems.

1. Introduction

Supercomputer centers, digital libraries, and archival storage communities have common persistent archival storage requirements. Each of these communities is building software infrastructure to organize and store large collections of data. An emerging common requirement is the ability to maintain data collections for long periods of time. The challenge is to maintain the ability to discover, access, and display digital objects that are stored within an archive, while the technology used to manage the archive evolves. We have implemented an approach based upon the storage of the digital objects that comprise the collection, augmented with the meta-data attributes needed to dynamically recreate the data collection. This approach builds upon the technology needed to support extensible database schema, which in turn enables the creation of data handling systems that interconnect legacy storage systems.
The long-term storage and access of digital information is a major challenge for federal agencies. The rapid change of technology resulting in obsolescence of storage media, coupled with the very large volumes of data (terabytes to petabytes in size) appears to make the problem intractable. The concern is that when the data storage technology becomes obsolete, the time needed to migrate to new technology may exceed the lifetime of the hardware and software systems that are being used. This is exacerbated by the need to be able to retrieve information from the archived data. The organization of the data into collections must also be preserved in the face of rapidly changing database technology. Thus each collection must be migrated forward in time onto new data management systems, simultaneously with the migration of the individual data objects onto new media. The ultimate goal is to preserve not only the bits associated with the original data, but also the context that permits the data to be interpreted. In this paper we present a scalable architecture for managing media migration, and an information model for managing migration of the structure of the context. The information model includes a logical schema for organizing attributes, a physical characterization for how to load the attributes into the database, and a data dictionary for defining semantics.

We rely on the use of collections to define the context to associate with digital data. The context is defined through the creation of semi-structured representations for both the digital objects and the associated data collection. Each digital object is maintained as a tagged structure that includes either the original bytes of data or persistent links to the object, as well as attributes that have been defined as relevant for the data collection. The collection context is defined through use of both logical and physical representations for organizing the collection attributes. By using infrastructure independent representations, the original context for the archived data can be maintained. A collection-based persistent archive is therefore one in which the organization of the collection is archived simultaneously with the digital objects that comprise the collection [1].

A persistent collection requires the ability to dynamically recreate the collection on new technology. For a solution, we consider the integration of scalable archival storage technology from supercomputer centers, infrastructure independent information models from the digital library community, and preservation models from the archivist’s community. An infrastructure that supports the continuous migration of both the digital objects and the data collections is needed. Scalable archival storage systems are used to ensure that sufficient resources are available for continual migration of digital objects to new media. The software systems that interpret the infrastructure independent representation for the collections are based upon generic digital library systems, and are migrated explicitly to new platforms. In this approach, the original representation of the digital objects and of the collections does not change. The maintenance of the persistent archive is then achieved through application of archivist policies that govern the rate of migration of the objects and the collection instantiation software.

The goal is to preserve digital information for at least 400 years. This paper examines the technical issues that must be addressed and presents a prototype implementation. The paper is organized into sections to provide a description of the persistence issues, and a generic description of the technology. (A description of the creation of a one million message persistent E-mail collection will be discussed in Part 2 in next month's issue of D-Lib Magazine.)
2. Persistence Issues

The preservation of the context to associate with digital objects is the dominant issue for collection-based persistent archives. The context is traditionally defined through specification of attributes that are associated with each digital object. The context is organized through relationships that exist between the attributes, and a description of the preferred organization of the attributes within user interfaces for accessing the data collection. We identify three levels of context that must be preserved:

- **Digital object representation.** Every digital object has attributes that define its structure, physical context, and provenance, and annotations that describe features of interest within the object. Since the set of attributes (such as annotations) will vary across all objects within a collection, a semi-structured representation is needed. Not all digital objects will have the same set of associated attributes.

- **Data collection representation.** The collection also has an implied organization, which is typically a subset of the attributes associated with the digital objects. A schema is used to support relational queries of the attributes or meta-data. It is possible to reorganize a collection into multiple tables to improve access by building new indexes, and in the more general case, by adding attributes. The schema used to organize the collection attributes can be different from the set of attributes associated with a digital object within the collection.

- **Presentation representation.** The user interface to the collection can present an organization of the collection attributes that is tuned to meet the needs of a particular community. Researchers may need access to all of the meta-data attributes, while students are interested in a subset. The structure used to define the user interface again can be different from the schema used for the collection organization. Each of these presentations represents a different view of the collection. Re-creation of the original view of a collection is a typical archival requirement.

Digital objects are used to encapsulate each data set. Collections are used to organize the context for the digital objects. Presentation interfaces are the structure through which collection interactions are defined. The challenge is to preserve all three levels of context for each collection.

2.1 Managing Context

Management of the collection context is made difficult by the rapid change of technology. Software systems used to manage collections are changing on three to five-year time scales. It is possible to make a copy of a database through a vendor specific dump or backup routine. The copy can then be written into an archive for long term storage. This approach fails when the database is retrieved from storage, as the database software may no longer exist. The archivist is then faced with migration of the data collection onto a new database system. Since this can happen for every data collection, the archivist will have to continually transform the entire archive. A better approach is needed.

An infrastructure independent representation is required for the collection that can be
maintained for the life of the collection. If possible, a common information model should be used to reference the attributes associated with the digital objects, the collection organization, and the presentation interface. An emerging standard for a uniform data exchange model is the eXtended Markup Language (XML) [2]. XML is the predominant instance of a semi-structured information model (i.e., labeled, ordered trees) and provides a representation for tagging data. Data could be relational data, object oriented data, schemas, procedures, etc. We define a collection as an XML view on the original tagged data using an information model. A particular example of an information model is the XML Document Type Definition (DTD) which provides a description for the allowed nesting structure of XML elements. Richer information models are emerging such as XSchema [3] (which provides data types, inheritance, and more powerful linking mechanisms) and XMI [4] (which provides models for multiple levels of data abstraction).

We shall reference the next generation information model as an Open Schema Definition (OSD). The OSD contains the collection schema and the presentation definition. For example, an XSL style sheet can be used for the presentation component of an OSD. XSL is the eXtensible Style sheet Language [5] and supports transformation of XML documents and formatting of the output for presentation. For the prototype we used XML DTDs and XSL style sheets together as the OSD.

It is possible to provide multiple presentational views of a collection. For our prototype we use multiple XSL style sheets for a specific collection to accommodate different user interfaces. The use of OSDs gives us freedom of choice for assembling a collection from tagged data objects, and for presenting the derived collection to multiple user communities.

Although XML DTDs were originally applied to documents only, they are now being applied to arbitrary digital objects, including the collections themselves. More generally, OSDs can be used to define the structure of digital objects, specify inheritance properties of digital objects, and define the collection organization and user interface structure.

While XML DTDs provide a tagged structure for organizing information, the semantic meaning of the tags is arbitrary, and depends upon the collection. A data dictionary is needed for each collection to define the semantics. A persistent collection therefore needs the following components of an OSD to completely define the collection context:

- Data dictionary for collection semantics,
- Digital object structure,
- Collection structure,
- User interface structure.

### 2.2 Managing Persistence

Persistence is achieved by providing the ability to dynamically reconstruct a data collection on new technology. While the software tools that do the reconstruction have to be ported to each new hardware platform or database, the collection can remain in its infrastructure independent format within an archive. The choice of the appropriate standard for the information model is vital for minimizing the support requirements for a...
collection-based persistent archive. The goal is to store the digital objects comprising the collection and the collection context in an archive a single time. This is possible if any changes to the standard information model are added as a superset to the prior information model. The knowledge required to manipulate a prior version of the information model can then be encapsulated in the software system that is used to reconstruct the collection. With this caveat, the persistent collection never needs to be modified, and can be held as infrastructure independent bit-files in an archive.

The re-creation or instantiation of the data collection is done with a software program that uses the schema descriptions that define the digital object and collection structure to generate the collection. The goal is to build a generic program that works with any schema description. This will reduce the effort required to support dynamic reconstruction of a persistent data collection to the maintenance of a single software system.

Maintaining persistent digital objects requires the ability to migrate data to new media. The reasons for continuing to refresh the media on which the collection is maintained are:

- Avoid loss of data because of the finite lifetime and resulting degradation of the media.

- Minimize storage costs. New media typically store at least twice as much data as the prior version, usually at the same cost per cartridge. Thus migration to new media results in the need for half as many cartridges, decreased floor space, and decreased operating costs for managing the cartridges. Note that for this scenario, the media costs for a continued migration will remain bounded, and will be less than twice the original media cost. The dominant cost to support a continued migration onto new media is the operational support needed to handle the media.

- Maximize the ability to handle exponentially increasing data growth. Many data collections are doubling in size in time periods shorter than a year. This means the effort to read the entire collection for migration to new media will be less than the effort to store the new data that is being collected within that year. Migration to higher density media that has a faster read/write rate is the only way to guarantee the archived data is accessible. The governing metric for a collection is the total time required to re-read the entire collection. Unless the re-read time remains bounded, a persistent archive will become unmanageable.

To facilitate migration and access, supercomputer centers keep all data in tape robots. For currently available tape (cartridges holding 20 GB to 50 GB of data), a single tape robot is able to store 120 terabytes to 300 terabytes of uncompressed data. By year 2003, a single tape robot is expected to hold 6000 terabytes, using 1-terabyte capacity cartridges. The storage of petabytes (thousands of terabytes) of data is now feasible. The capacity of archives will not be a limiting factor.

Given that the collection context and the digital objects can be migrated to new media, the remaining system that must be migrated is the archival storage system itself. The software that controls the tape archive is composed of databases to store the storage location and name of each data set, logging systems to track the completion of
transactions, and bitfile movers for accessing the storage peripherals. Of these components, the most critical resource is the database or nameserver directory that is used to manage the names and locations of the data sets. At the San Diego Supercomputer Center, the migration of the nameserver directory to a new system has been done twice, from the DataTree archival storage system to the UniTree archival storage system, and from UniTree to the IBM High Performance Storage System [6]. Each migration required the read of the old directory, and the ingestion of each data set into the new system. Although the number of files increased from 4 million to 7 million between the two migrations, the time required for the migration decreased from 4 days to 1 day. This reflects advances in vendor supplied systems for managing the name space. Based on this experience, it is possible to migrate to new archival storage systems, without loss of data.

One advantage of archival storage systems is their ability to manage the data movement independently from the use of the data. Each time the archival storage system was upgraded, the new version of the archive was built with a driver that allowed tapes to be read from the old system. Thus migration of data between the archival storage systems could be combined with migration onto new media, minimizing the number of times a tape had to be read.

The creation of a persistent collection can be viewed as the design of a system that supports the independent migration of each internal hardware and software component to new technology. Management of the migration process then becomes one of the major tasks for the archivist.

2.3 Managing Scalability

A persistent archive can be expected to increase in size through either addition of new collections, or extensions to existing collections. Hence the architecture must be scalable, supporting growth in the total amount of archived data, the number of archived data sets, the number of digital objects, the number of collections, and the number of accesses per day. These requirements are similar to the demands that are placed on supercomputer center archival storage systems. We propose a scalable solution that uses supercomputer technology, based on the use of parallel applications running on parallel computers.

A scalable system is built by identifying both the capabilities that are best provided by each component, and the constraints that are implicit within each technology. Interfaces are then constructed between the components to match the data flow through the architecture to the available capabilities. Archival storage systems are used to manage the storage media and the migration to new media. Database management systems are used to manage the collections. Web servers are used to manage access to the system.

Archival storage systems excel at storing large amounts of data on tape, but at the cost of relatively slow access times. The time to retrieve a tape from within a tape silo, mount the tape into a tape drive, and ready the tape for reading is on the order of 15-20 seconds for current tape silos. The time required to spin the tape forward to the position of the desired file is on the order of 1-2 minutes. The total time can be doubled if the tape drive is already in use. Thus the access time to data on tape can be 2-4 minutes. To overcome this high latency, data is transferred in large blocks, such that the time it takes
to transfer the data set over a communication channel is comparable to the access latency time. For current tape peripherals which read at rates from 10 MB/sec to 15 MB/sec, the average data set size in an archive should be on the order of 500 MB to 1 GB. Since digital objects can be of arbitrary size, containers are used to aggregate digital objects before storage into the archive.

The second constraint that must be managed for archives is the minimization of the number of data sets that are seen by the archive. Current archival storage nameservers are able to manage on the order of 10 - 40 million data sets. If each data set size is on the order of 500 MB, the archive can manage about 10 petabytes of data (10,000 TBs, or 10 million GBs). Archival storage systems provide a scalable solution only if containers are used to aggregate digital objects into large data sets. The total number of digital objects that can be managed is on the order of 40 billion, if one thousand digital objects are aggregated into each container.

Databases excel at supporting large numbers of records. Note that the Transaction Processing Council D benchmark [7] measures performance of relational databases on decision support queries for database sizes ranging from 1 gigabyte up to 3 terabytes and from 6 million to 18 billion rows. Each row can represent a separate digital object. With object relational database systems, a binary large object or BLOB can be associated with each row. The BLOBs can reside either internally within the database, or within an external file system. In the latter case, handles are used to point to the location of BLOB. The use of handles makes it feasible to aggregate digital objects within containers. Multiple types of container technology are available for aggregating digital objects. Aggregation can be done at the file level, using utilities such as the TAR program, at the database level through database tablespaces, or at an intermediate data handling level through use of software controlled caches. The database maintains the information needed to describe each object, as well as the location of the object within a container and the location of the container within the storage system. A data handling system is used to support database access to archival storage.

Queries are done across the attributes stored within each record. The time needed to respond to a query is optimized by constructing indexes across the database tables. This can reduce the time needed to do a query by a factor of a thousand, at the cost of the storage space for the index, and the time spent in assembling the index. Persistent collections may be maintained on disk to support interactive access, or they may be stored in the archive, and rebuilt on disk when a need arises. If the collection is reassembled from out of the archive, the dominant time needed for the process may be the time spent creating a new index. Since archival storage space is cheap, it may be preferable to keep both infrastructure independent and infrastructure dependent representations of a collection. The time needed to load a pre-indexed database snapshot is a small fraction of the time that it would take to reassemble and index a collection. The database snapshot, of course, assumes that the database software technology is still available for interpreting the database snapshot. For data collections that are frequently accessed, the database snapshot may be worth maintaining.

The presentation of information for frequently accessed collections requires Web servers to handle the user load. Servers function well for data sets that are stored on local disk. In order to access data that reside within an archive, a data handling system is needed to transfer data from the archive to the Web server. Otherwise the size of the accessible collection may be limited to the size of the Web server disk cache. Web...
servers are available that distribute their load across multiple CPUs of a parallel computer, with parallel servers managing over 10 million accesses per day.

Web servers provide a variety of user interfaces to support queries and information discovery. The preservation of the user interface requires a way to capture an infrastructure independent representation for the query construction and information presentation. Web servers are available that retrieve information from databases for presentation. What is needed is the software that provides the ability to reconstruct the original view of the collection, based upon a description of the collection attributes. Such technology is demonstrated as part of the collection instantiation process in the SDSC persistent archive prototype.

2.4 Managing Heterogeneity of Data Resources

A persistent archive is inherently composed of heterogeneous resources. As technology evolves, both old and new versions of the software and hardware infrastructure will be present at the same time. An issue that must be managed is the ability to access data that is present on multiple storage systems, each with possibly different access protocols. A variant of this requirement is the ability to access data within an archive from a database that may expect data to reside on a local disk file system. Data handling systems provide the ability to interconnect archives with databases and with Web servers. Thus the more general form of the persistent archive architecture uses a data handling system to tie each component together. At the San Diego Supercomputer Center, a particular implementation of a data handling system has been developed, called the Storage Resource Broker (SRB) [8].

The SRB supports the protocol conversion needed for an application to access data within either a database, file system, or archive. The heterogeneous nature of the data storage systems is hidden by the uniform access API provided by the SRB. This makes it possible for any component of the architecture to be modified, whether archive, or database, or Web server. The SRB Server uses a different driver for each type of storage resource. The information for which driver to use for access to a particular data set is maintained in the associated Meta-data Catalog (MCAT) [9-10]. The MCAT system is a database containing information about each data set that is stored in the data storage systems. New versions of a storage system are accessed by a new driver written for the SRB. Thus the application is able to use a persistent interface, even while the storage technology changes over time.

3. Implementation Strategy

A collection-based persistent archive can be assembled using a scalable architecture. The scalable architecture relies upon parallel hardware and software technology that is commercially available. The persistent archive requires the integration of three separate components: archival storage, collection management, and access servers through the use of a data handling system. The result is a system that can be modified to build upon new technology on an incremental basis. For a persistent archive to work within this migration environment, the data context must be maintained in an information independent representation. The technology to instantiate the collection will have to be migrated forward in time, along with the data handling system. The collection can be kept as bit-files within the archive, while the supporting hardware and software systems
3.1 General Architecture

The implementation of a prototype persistent archive at SDSC is based upon use of commercially available software systems, augmented by application level software developed at the San Diego Supercomputer Center. The general architecture software components are listed below, followed by the particular software system used for the prototype:

- Archival storage system - IBM High Performance Storage System (HPSS) [6]
- Data handling system - SDSC Storage Resource Broker (SRB) [8]
- Object relational database - Oracle version 7.3, IBM DB2 Universal Database
- Collection management software - SDSC Meta-data Catalog (MCAT) [9, 10]
- Collection instantiation software - SDSC scripts
- Collection ingestion software - SDSC scripts
- Semi-structured data model - eXtended Markup Language - Document Type Definition [2]
- Relational data model - ANSI SQL Data Definition Language [11]
- DTD manipulation software - UCSD XML Matching and Structuring language (XMAS) [12]
- Web server - Apache Web server
- Presentation system - Web Browser such as Internet Explorer version 5.

The hardware components are:

- Archival storage system - IBM SP 8-node, 32-processor parallel computer, 180 TB of tape storage, three Storage Technology tape robots, and 1.6 TB of RAID disk cache
- Data management system - Sun Enterprise 4-processor parallel computer
- Data ingestion platform - SGI workstation
- Network interconnect - Ethernet, FDDI, and HiPPI

Each of these systems is scalable, and can be implemented using parallel computing technology. The efficiency of the archival storage system is critically dependent upon the use of containers for aggregating data before storage. Three different mechanisms...
have been tried at SDSC:

- Unix utilities. The TAR utility can be used to aggregate files. For container sizes of 100 MB, the additional disk space required is minimal. The disadvantages are that the container must be read from the archive and unpacked before data sets are accessed.

- Database tablespace. At SDSC, a prototype version of the DB2 UDB parallel object-relational database has been used to support large data collections. The prototype database stores the digital objects internally within tablespaces. The tablespaces can be stored within the HPSS archival storage system, and retrieved to a disk cache on demand. This effectively increases the database storage capacity to the size of the archive, while simultaneously aggregating digital objects into containers before storage in the archive.

- Data handling software cache. The SDSC Storage Resource Broker supports containers. Digital objects that are written into an archive through the SRB are aggregated into a container on a disk cache. When the container is full, the SRB writes the container into the archive. When data is referenced, the container is retrieved from the archive and the data set is read directly out of the container by the SRB.

### 3.1.1 Archive

The core of the architecture is the archival storage system, as it ultimately determines the total capacity, data ingestion rate, and data migration support for the persistent archive. The High Performance Storage System (HPSS) is supported by a parallel computer, the IBM SP. HPSS at SDSC currently stores over 14 million files, with an aggregate size of 140 TB. Data movement rates have been achieved that exceed 1 TB of data storage per day. The system sustains 16,000 file operations per day. The HPSS system is accessed over high-speed networks through a High Performance Gateway Node (HPGN). The HPGN supports multiple types of network access, including a 100 MB/sec HiPPI network, 100 Mb/sec FDDI, and Ethernet. The HPGN is directly connected to the nodes of the SP on which the HPSS software system runs through the Trail Blazer 3 switch. The HPSS central control services run on one of the four-processor SP nodes, while the bitfile movers that read/write data off of disk and tape are distributed across seven of the SP nodes. By interconnecting the external networks through the HPGN onto the SP switch, all of the mover nodes can be used in parallel, sustaining high data throughput. By having disk and tape drives connected to each of the mover nodes, data can be migrated in parallel to tape. Measured data movement rates from the nodes to the HPGN are 90 MB/s for file sizes on the order of 10 MB.

The HPSS archive includes multiple backup systems for preserving the nameserver directory, including mirroring of the directory on disk, backup of snapshots of the directory onto tape, transaction logging of all changes to the directory, and reconciliation of the transaction logs with the directory snapshots on a daily basis. To handle disasters, copies of the critical data sets are maintained in a second HPSS archival storage system located within another city. A description of the backup systems is given in [14]. The attention paid to nameserver directory backup is of critical importance. If the nameserver directory is lost, it will not be possible to name the files stored in the archive.
The HPSS archive is scalable, through the addition of more nodes, disk, and tape drives. The system has recently been upgraded to a capacity of 360 GB of uncompressed data through the acquisition of tape drives that write 20 GBs of data per cartridge. The system supports data compression. For the scientific data sets stored at SDSC, the average compression ratio is a factor of 1.5, implying the total capacity of the system is 500 TB.

### 3.1.2 Data Handling System

The data handling system provides the ability to connect heterogeneous systems together. We provide a detailed description of the SDSC data handling system to illustrate the software infrastructure needed to provide location and protocol transparency. The data handling infrastructure developed at SDSC has two components: the SDSC Storage Resource Broker (SRB) [8] that provides federation and access to distributed and diverse storage resources in a heterogeneous computing environment, and the Meta-data Catalog (MCAT) [9] that holds systemic and application or domain-dependent meta-data about the resources and data sets (and users) that are being brokered by the SRB. The SRB-MCAT system provides the following capabilities:

- uniform APIs for access to heterogeneous file systems, databases, and archival storage,
- protocol-transparency and location-transparency when accessing distributed systems,
- uniform persistent name space abstraction [24] over the file systems that are being brokered,
- collection-based access to remote data sets, thus supporting information discovery based on domain and system-dependent meta-information stored along with (or extracted from) the stored files,
- facilities for replication, copying or moving files across heterogeneous systems, performing resource-level operations (proxy operations) on data before delivery to the client, and
- an integrated encryption and authentication system that can range from no security to fully encrypted and fully authenticated data transfer including security against man-in-the-middle security intrusions [15, 16].

The SDSC Storage Resource Broker (SRB) is middleware that provides distributed clients with uniform access to diverse storage resources in a heterogeneous computing environment. Storage systems handled by the current release of the SDSC SRB include the UNIX file system, archival storage systems such as UniTree, ADSM and HPSS, and database Large Objects managed by various DBMSs including DB2, Oracle, and Illustra. Currently, the system runs on supercomputers such as the CRAY C90, CRAY T3E and IBM SP, on workstations such as Sun, SGI, and Compaq platforms, and on Windows NT. The SRB API presents clients with a logical view of data sets stored in the SRB. Similar to the file name in the file system paradigm, each data set stored in SRB has a logical name, which may be used as a handle for data operation. Unlike the
file system where the physical location of a file is implied in its path name through its
mount point, the physical location of a data set in the SRB environment is logically
mapped to the data sets. Therefore, data sets belonging to the same collection may
physically reside in different storage systems. A client does not need to remember the
physical mapping of a data set. It is stored as meta-data associated with the data set in
the MCAT catalog. Data sets in the SRB are grouped into a logical (hierarchical)
structure called *collections*. The collection provides an abstraction for:

- placing similar objects (possibly, physically distributed) under one collection
  (e.g., image collections of a museum) and

- placing all dissimilar objects that have a common connection under one
  abstraction (e.g., all the text paragraphs, images, figures, and tables of a
document).

The SRB supports data replication in two ways. One can replicate an object during
object creation or modification. To enable this, SRB and MCAT allow the creation of
logical storage resources (LSR) which are a grouping of two or more resources. When
an application creates or writes a data set to these logical resources, the operations are
performed on each of the grouped resources. The result of using a LSR is that a copy of
the data is created in each of the physical resources belonging to the logical resource. It
is possible to specify that the write operation is successful if $k$ of the $n$ copies are
created. The user can modify all the copies of the data by writing to the data set with a
"write all." The SRB provides an off-line replication facility to replicate an existing data
set. This operation can also be used for synchronization purposes. When accessing
replicated objects, SRB will open the first available replica of the object as given by a
list from MCAT. The SRB also provides authentication and encryption facilities \[15,\]
\[16\], access control list and ticket-based access \[17\], and auditing capabilities to give a
feature-rich environment for sharing distributed data collections among users and
groups of users.

The design of the SRB server is based on the traditional network connected client/server
model but has the additional capability of federation. Once a connection from a client is
established and authenticated, a SRB agent is created that brokers all the operations for
that connection. A client application can have more than one connection to a SRB server
and to as many servers as required. The federation of SRBs implies that a client
connects to any SRB server while accessing a resource that is brokered by another
server. An inter-SRB communication protocol supports the federation operation. The
SRB communicates with MCAT to obtain meta-information about the data set, which it
then uses for accessing the data set.

### 3.1.3 Collection Management

A characterization of a relational database requires a description of both the logical
organization of attributes (the schema), and a description of the physical organization of
attributes into tables. For the persistent archive prototype we used XML DTDs to
describe the logical organization. The physical organization of relational databases was
expressed using the Data Definition Language, DDL \[11\]. A combination of the schema
and physical organization can be used to define how queries can be decomposed across
the multiple tables that are used to hold the meta-data attributes. It is possible to
generate arbitrary mappings between a DTD semi-structured representation, and a DDL
relational representation of a collection. A preferred correspondence between the two representations must be defined if a relational database is used to assemble the collection. XML-based databases are becoming available that remove the need to describe the physical layout. Examples are Excelon [18] (an XML variant of ObjectStore) and Ariel [19] (an XML version of O2). By using an XML-based database, it is possible to avoid the need to map between semi-structured and relational organizations of the database attributes. This minimizes the amount of information needed to characterize a collection, and makes the re-creation of the database easier.

A detailed description of the SDSC MCAT system is provided to illustrate the complexity of the information management software needed to describe and manage collection level meta-data. The SDSC MCAT is a relational database catalog that provides a repository of meta information about digital objects. Digital object attributes are separated into two classes of information within the MCAT:

- System-level meta-data that provides operational information. These include information about resources (e.g., archival systems, database systems, etc., and their capabilities, protocols, etc.) and data objects (e.g., their formats or types, replication information, location, collection information, etc.).

- Application-dependent meta-data that provides information specific to particular data sets and their collections (e.g., Dublin Core [20, 21] values for text objects).

Both of these types of meta-data are extensible, i.e., one can add and/or remove attributes. Internally, MCAT keeps schema-level meta-data about all of the attributes that are defined. The schema-level attributes are used to define the context for a collection and enable the instantiation of the collection on new technology. The attributes include definition of:

- Logical Structure: When a set of meta-data is registered with MCAT, one needs to identify a logical structure in which the rest of the meta-data will be organized. The logical structure should not be confused with database schema and are more general than that. For example, we have implemented the Dublin Core database schema [20] to organize attributes about digitized text. The attributes defined in the logical structure that is associated with the Dublin Core schema contains information about the subject, constraints, and presentation formats that are needed to display the schema along with information about its use and ownership.

- Attribute Clusters: An attribute cluster is a set of attribute names that are logically interconnected and that have a one-to-one mapping among them. One can view them as a (single or a set of) normalized table(s) in a database context. For example, in the Dublin Core, publisher, name, address, and contact information form a cluster. Contributor name and contributor type form a second cluster; title and its type form yet another cluster, and so on. Similarly in our system-level MCAT core meta-data, we have one cluster for each data replica containing the type, location, and size of the data objects. This aids the implementation of relational joins across the meta-data tables, since each replica has only one value for these properties and these properties provide the physical characteristics of the object. For each cluster, MCAT keeps information about any constraints and comments that can be searched when using the attribute, along with information about use-privileges and grant-of-use-privileges for the cluster. For each attribute,
MCAT keeps more than 20 different types of information including its physical, logical and input and output characteristics [9].

- **Token Attributes**: Token attributes have a specific function (compared to other attributes); they capture some simple semantic information about the domain of discourse. One can also use the token attribute to capture semantic translation between discipline domains (e.g., common names vs. scientific names) and also capture hierarchical and equivalence relationships in the domain of discourse. Given the development of semantic standards within a discipline, one can use the token attribute as a bridge between two schemas and provide semantic interoperability.

- **Linkages**: Linkages provide a means for inter-operating within and between schema. One can define four types of linkages:

  1. attribute-to-attribute,
  2. cluster-to-attribute,
  3. cluster-to-cluster, and
  4. cluster-to-token.

Each of the linkages can be from one-to-many, many-to-one, or many-to-many. The linkage information is used to generate joins dynamically based on the user’s chosen set of attributes. The join algorithm uses Steiner Tree generation of SQL commands from a directed acyclic graph; the DAG is a mapping of clusters and the linkages between them. The linkage information is used to perform federated query operations across schemas. The DAG is also used to figure out the notion of an allowed query by disallowing queries that span disjointed graphs.

MCAT provides APIs for creating, modifying and deleting the above structures. MCAT provides an interface protocol for applications such as Web servers. The protocol uses a data structure for the information interchange which is called MAPS -- Meta-data Attribute Presentation Structure. The data structure, which also has a wire-format for communication and a data format for computation, provides an extensible model for communicating meta-data information. A mapping is being developed to translate from the MAPS structure to the Z39.50 format [22]. Internal to MCAT, the schema for storing meta-data (may possibly) differ from MAPS, and hence mappings between the internal format and MAPS are needed for every type of implementation of the MCAT. Note that it is possible to store the meta-data in databases, flat files, or LDAP directories [23]. MAPS provides a uniform structure for communicating between MCAT servers and user applications.

The MAPS structure defines a query format, an update format and an answer format. The MAPS query format is used by MCAT in generating joins across attributes based on the schema, cluster and linkages discussed above. Depending upon the internal catalog type (e.g., DB2 database, Oracle database, or LDAP) a lower-level target query is generated. Moreover, if the query spans several database resources, a distributed query plan is generated.

The MCAT system supports the publication of schemata associated with data collections, schema extension through the addition or deletion of new attributes, and the
dynamic generation of the SQL that corresponds to joins across combinations of attributes. GUIs have been created that allow a user to specify a query by selecting the desired attributes. The MCAT system then dynamically constructs the SQL needed to process the query. By adding routines to access the schema-level meta-data from an archive, it is possible to build a collection-based persistent archive. As technology evolves and the software infrastructure is replaced, the MCAT system can support the migration of the collection to the new technology. Effectively, the collection is completely represented by the set of digital objects stored within the archive, the schema that contains the digital object meta-data, and the schema-level meta-data that allows the collection to be instantiated from scratch.

To Be Continued:

The first part of this article has concentrated on a description of the persistence issues, and a generic description of the scalable technology for managing media and context migration. The second part of the article will describe the creation of a one million message persistent E-mail collection. It will discuss the four major components of a persistent archive system: support for ingestion, archival storage, information discovery, and presentation of the collection. The technology to support each of these processes is still rapidly evolving, and opportunities for further research are identified.

References


[19] Eric N. Hanson, "The Design and Implementation of the Ariel Active Database Rule System", IEEE Transactions on Knowledge and Data Engineering , Vol. 8, No. 1, February 1996


[24] Persistent namespace abstraction, the Handle System <http://www.handle.net/>.

Copyright © Reagan Moore, Chaitan Baru, Arco Rajasekar, Bertram Ludascher, Richard Marciano, Michael Wan, Wayne Schroeder, and Amarnath Gupta

Top | Contents
D-Lib Magazine Access Terms and Conditions

**DOI**: 10.1045/march2000-moore-pt1