Preservation DataStores: Architecture for Preservation Aware Storage

Michael Factor, Dalit Naor, Simona Rabinovici-Cohen,
Leeat Ramati, Petra Reshef, Julian Satran
IBM Haifa Research Lab
{factor, dalit, simona, leeat, petra, satran}@il.ibm.com

David L. Giaretta
Science and Technology Facilities Council
d.l.giaretta@rl.ac.uk

Abstract

The volumes of digital information are growing continuously and most of today's information is "born digital". Alongside this trend, business, scientific, artistic and cultural needs require much of this information to be kept for decades, centuries or longer. The convergence of these two trends implies the need for storage systems that support very long term preservation for digital information. We describe Preservation DataStores, a novel storage architecture to support digital preservation. It is a layered architecture that builds upon open standards, along with the OAIS, XAM and OSD standards. This new architecture transforms the logical information-object, a basic concept in preservation systems, into a physical storage object. The transformation allows more robust and optimized implementations for preservation aware storage. The architecture of Preservation DataStores is being developed as an infrastructure component of the CASPAR project[*] *and will be tested in the context of this project using scientific, cultural, and artistic data.*

[*]Work partially supported by European Community under the Information Society Technologies (IST) program of the 6th FP for RTD - project CASPAR contract IST-033572. The authors are solely responsible for the content of this paper. It does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of data appearing therein.

1. Introduction

The growth of long-lived digital information, along with new compliance regulations such as HIPAA, Sarbanes-Oxley, OSHA and other federal securities laws and regulations, demand the long-term viability of data. Other types of data must be preserved for the benefit of humankind. Just some examples include earth observation data from the European space agency and cultural heritage data from UNESCO, which must be kept for decades, centuries, or longer. Additionally, the amount of long-lived data is expected to grow as more digital devices generate vast amounts of born-digital data. This increases the need for digital preservation systems to preserve a myriad of information types including scientific, financial, healthcare, artistic, and cultural data—for tens and hundreds of years. Most of this information is reference data; it hardly changes once written. Due to its nature, this kind of data is typically accessed infrequently. Consequently, preservation systems generally utilize near-line and offline storage.

The digital preservation challenge can be divided into "bit preservation" and "logical preservation". Bit preservation is the ability to restore the bits in the presence of storage media degradation or obsolescence, physical destruction by a malicious user, or even environmental catastrophes such as fire and flooding. Logical preservation involves preserving the understandability and usability of the data, despite logical attacks that may occur or unknown future changes that will take place in technologies and users. The data needs to be properly accessed and interpreted in the far future when current technologies for servers, operating systems, data management products and applications may no longer exist. Additionally, logical preservation needs to maintain the provenance of the data, along with its authenticity and integrity, and ensure that only legitimate users will access it.

While the issues surrounding bit preservation are well understood and can be supported by some products, logical preservation is still an open research area. A core standard for digital preservation systems is the Open Archival Information System (OAIS) [1], an ISO standard since 2003 (ISO 14721:2003 OAIS). This standard concentrates on logical preservation and specifies the terms, concepts, and reference models to be used in a system dedicated to preserving digital assets for a dedicated user group that needs to access and understand the information preserved (designated community). OAIS is a high-level reference model, which means it is flexible enough to be used in a wide variety of environments. However, more detailed steps and workflow stages need to be developed for its implementation.

OAIS defines logical preservation as a recursive problem; in addition to storing the raw data, it must also store the separately-born (in time and place) metadata that helps interpret and use the raw data. Moreover, this metadata (representation information) may recursively need additional metadata to help interpret it. The recursion ends when the representation information is non-digital and preserved by the designated community. To further support logical preservation, OAIS defines additional metadata that is associated with the raw data and describes its context, logs its provenance, and ensures its data integrity (fixity).

At the heart of any solution to the preservation problem, resides a storage component, which is the permanent location of the information. Traditional archival storage considers only bit preservation, if it considers preservation issues at all, and generally has functions to insert data into permanent storage, manage a storage hierarchy, refresh media on which archive holdings are stored, perform routine and special error checking, provide disaster recovery capabilities, and retrieve data from the permanent storage.

We have already laid the foundation for the concept of *preservation aware storage* [2], which supports logical preservation in addition to bit preservation. In this paper, we provide an architecture to support this concept. Previously [2], we argued that the traditional archival storage should be enhanced with additional functionalities oriented towards logical preservation. Preservation aware storage encapsulates the raw data with its complex interrelated metadata objects, so they are inseparable during the migration process and future data access. Preservation aware storage also adds more functions to the storage component. It decreases the amount of data transfers between applications and the storage by offloading data intensive functions, such as fixity computations, to the storage. In addition, preservation aware storage simplifies applications by transferring the responsibility of managing the storage-related events, such as provenance events, to the storage itself. Finally, preservation aware storage handles migration internally, including the ability to execute externally-specified logical transformations.

Our main contribution in this paper is the definition of an architecture for preservation aware storage: Preservation DataStores (PDS). PDS is a significant advance over traditional storage, which is oblivious to the needs of logical preservation. In contrast to traditional block or file storage, or even traditional archival systems, PDS materializes the logical concept of a preservation information-object into a physical storage object. It defines a way to ensure the grouping of metadata with data, supports functions such as provenance and fixity that are close to the data, and supports the execution of transformations during physical migrations. PDS is a layered architecture based on open standards and, as such, is compliant with the general design principle of preservation systems that employ open standards where possible. We are developing Preservation DataStores as an infrastructure component of CASPAR, a European Union project that focuses on the preservation of data for very long periods of time [3]. CASPAR is building a framework to support the end-to-end preservation ”lifecycle” for scientific, artistic, and cultural information based on existing and emerging standards, most notably the OAIS model.

In the rest of this paper, we review the concept of preservation aware storage and the requirements a preservation aware storage must address. We then describe the PDS architecture, which aims to address those requirements. PDS builds on a set of standard-based layers (OAIS, XAM and OSD) with generic mappings between the layers. The described mappings have merits beyond preservation and may be used for other types of applications as well. We conclude with a discussion of future work and a summary.

2. Related work

The storage aspect of digital preservation has been attracting more attention lately [4, 5, 6, 7] and this trend is likely to increase. Storer et al. [7] describe both existing security threats (e.g., integrity, authentication and privacy) and new specific threats (e.g., slow attacks) that arise when storing data for long periods of time. Furthermore, the work examines how existing systems address these concerns to ensure long term survivability. Baker et al. [5] present the key differences between enterprise systems and long term storage systems in terms of requirements and threats. The work provides several architectural solutions that focus on replication across autonomous sites, reduced per-site engineering costs, and the ability to scale over time and different technologies. Baker et al. [4] explore the needs and threats related to the long-term storage of digital information. They describe an extended reliability model and discuss several strategies to reduce data loss. The work suggests a possible system architecture that incorporates the different strategies and aims at balancing the different tradeoffs. All of the studies mentioned above concentrate primarily on bit preservation, suggesting how traditional storage systems can address the new challenges posed by long-term bit preservation but do not address the logical preservation aspects.

The Storage Resource Broker (SRB) [6, 8, 9] is a data grid technology developed by the San Diego Supercomputing Center (SDSC). SRB is middleware built on top of standard file systems, commercial archives, and storage systems. It manages distributed data, enabling the creation of data grids that focus on the sharing of data. SRB was recently extended to directly support archives that focus on the preservation of data from the bit preservation perspective. The SRB stores the data records (content) as files on the storage system repository; a separate database is used to store the metadata (context) related to the electronic record and the data grid is used to maintain the association between the content and context. The data grid technology also includes support for managing and replicating data on remote storage systems, a uniform name space for referencing the data, a federated catalog for managing information about the data, and mechanisms for interfacing to the preferred access method. SRB has been used as a foundation technology in the context of several preservation projects; many of them were commissioned by the National Archives and Records Administration (NARA) and supported by the Library of Congress and NSF. The SRB code was recently extended into a new system called iRODS [10], Intelligent Rule-Oriented Data management System. Its rule engine can be useful for implementing preservation-specific policies.

The e-depot digital archiving system of the National Library of the Netherlands (KB) is described by Oltmans et al. [11]. The main component of e-depot is the Digital Information Archiving System (DIAS) from IBM. DIAS provides an open deposit library solution for storing and retrieving electronic documents and multimedia files. The e-depot library conforms to the OAIS standard and supports physical and logical digital preservation. The paper describes the design of an extension to DIAS called Preservation Manager, which manages and stores the metadata associated with the preserved digital content. The content and metadata are ingested into DIAS, where the content is assigned a unique identifier. However, although the content bit stream is stored in the archive, the metadata associated with it is stored in the KB catalog, which is in a physically separate location.

Like many other systems, both SRB and DIAS keep the metadata in a database that is separate from the content. Our approach stores the metadata along with the raw data managed by the storage itself. This encapsulation keeps the data and the metadata tied together as the bits of the preserved data change (e.g., migrations). The physical co-location is a key characteristic; it allows graceful loss of data and thus reduces the chances of losing the ability to interpret the data.

One of the intriguing approaches to digital preservation is the Universal Virtual Computer (UVC) [12]. It employs the emulation approach and aims to allow digital objects to be retained in their original format. To enable the interpretation of the object in the future, a program that can decode the data and present it in an understandable form is also kept. UVC and the emulation approach complement our approach and can be used as additional representation information in a preservation aware storage.

Recently, we have observed an emerging technology trend where functions such as bit-to-bit data migration, block-level data integrity, and even encryption, are carried out by advanced, intelligent disks and tapes. Some systems, such as the provenance-aware storage system (PASS) [13], track the provenance of data at the storage level rather than managing it in a standalone database. PASS is prototyped at the file system level and demonstrates the advantages of managing provenance at the storage system level.

Long-term security aspects are further explored in the POTSHARDS [14] system, which provides secure long-term storage without encryption. The POTSHARDS system shifts data secrecy from encryption to authentication by using secret splitting across multiple authentication domains. Furthermore, since data retrieval methods cannot rely upon external indexes, approximate pointers between independent archives are utilized to ensure availability.

Several other efforts [15, 16] provide further motivation for solving the general data preservation problem as well as requirements for the overall preservation system. The National Digital Information Infrastructure and Preservation Program (NDIIPP)

[17] is a collaborative initiative run by the Library of Congress. Its goal is to develop a national strategy for digital preservation. NDIIPP develops a rich set of formats appropriate for preservation and provides a high level architecture. The architecture emphasizes the need to support a federation of archives but does not refer to specific storage aspects or functionalities of the system.

3. Preservation aware storage

This section briefly summarizes the concept of preservation aware storage [2]. Preservation aware storage serves as the foundation for the architecture proposed in this paper.

The storage component is an important part of preservation systems. This is the portion of the system that manages the long-term storage and maintenance of digital material. We argue [2] that digital preservation systems will be more robust and have a lower probability for data corruption or loss if they offload preservation related functionality to the storage layer. This offloading enables the storage layer to reduce the amount of data transfers to applications and improve its own data placement in favour of preservation. We use the term *preservation aware storage* to describe such a storage component with a built-in support for preservation. OAIS-based storage is a specific type of preservation aware storage, based on OAIS notions and its functional and information models.

The main OAIS concept relevant to storage is the Archival Information Package (AIP) depicted in Figure 1.

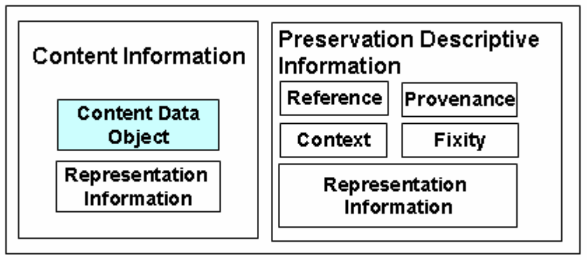
Content Information
Content Data Object
Representation Information
Preservation Descriptive Information
Reference
Provenance
Context
Fixity
Representation Information

Figure 1. OAIS AIP logical structure

An AIP contains zero or one content information parts, which include the raw data and its interpretation, and one or more Preservation Description Information (PDI) parts, which provide additional metadata related to the content logical preservation. More specifically, content information contains the Content Data Object (the raw data) that is the focus of the preservation, plus the Representation Information (RepInfo), which is needed to render the object intelligible to its designated community. This may include information regarding the hardware and software environment needed to view the content data object. Note that the RepInfo is a recursive object and may have additional RepInfo to interpret itself. This recursion ends when facing a RepInfo that is non-digital and will be preserved by the designated community. For example, astronomical data represented in a FITS file is associated with RepInfo that includes a dictionary to describe the FITS keywords. The FITS dictionary is associated with RepInfo that includes the dictionary structure specification. Assuming the dictionary specification is in XML, its RepInfo includes the XML specification, and the latter is associated with a RepInfo that includes the unicode specification. We assume that the unicode specification is preserved by the designated community and thus it doesn't need more RepInfo.

The PDI is further divided into four sections: reference (globally unique and persistent identifiers), provenance (chain of custody, the history and the origin of the content information custody), context (relationships of the content information to its environment), and fixity (a demonstration that the particular content information has not been altered in an undocumented manner).

The following list is a refinement of our previous work [2] and includes the major requirements and desired features of an OAIS-based preservation aware storage:

- In the storage, encapsulate and physically co-locate the raw data and its complex interrelated metadata objects, such as representation information, provenance, and fixity. This ensures that the metadata needed for interpretation is not separated from the raw data and thus never lost (if the raw data survives).
- Include the representation information of metadata (e.g., representation information of fixity and provenance) so the metadata can be interpreted when accessed in the future.
- Utilize the locality property and execute data intensive functions such as fixity (i.e., data integrity) computations within the storage component.
- Handle some of the provenance events internally. The applications on top of the preservation aware storage should be free of managing events that can be handled internally in the storage. Moreover, this enables richer types of provenance events and the inclusion of events related to the migration between physical medium and the transformation of representations.
- Support the loading and execution of external transformations during the migration process

and facilitate on demand triggering of these transformations.

- Support media migration, as opposed to system migration. In media migration, performing migration from one system to another can be done by physically detaching the media from one system and attaching it to the new system.
- Maintain referential integrity including updating all the links during the migration process so they remain valid in the new system. This requires an awareness of certain metadata fields that represent links, both internally to the system and externally.
- Ensure *readability* of the data by a different system in the future. This is done by developing and supporting global self-describing media independent formats.
- Support a *graceful loss* of data. Some portions of the data are likely to be lost or become corrupted over time. If some data is lost, a good preservation system must minimize the economic effect of this data loss and prevent cases where data that is still intact in the system cannot be read or interpreted.

From among these requirements, we focused on supporting logical preservation; however, as should be clear, there is often an interdependency and interaction between logical and bit preservation (e.g., the interactions between migration to new media and transformations of formats).

4. Preservation DataStores: preservation aware storage architecture

Preservation DataStores (PDS) are OAIS-based preservation aware storage that focus on supporting logical preservation. They are aware of the structure of an archival information package (AIP) and offload OAIS derived generic functions such as representation information inclusion, provenance tracking, fixity computation, and migration support to the storage layer. They provide strong encapsulation of large quantities of metadata with the data at the storage level and enable easy migration of the preserved data across storage devices.

4.1. Architecture overview

The PDS architecture includes a stack of three layers based on OAIS, XAM and OSD, respectively. Figure 2 depicts the general architecture of PDS. The top layer is an OAIS-based preservation engine, which provides preservation functionality for heterogeneous data and applications. It includes efficient generation and placement of AIPs along with support for migration and data transformations performed within the storage. The second layer includes an eXtensible Access Method (XAM) [18] library. XAM is an emerging storage standard intended for reference information that provides a logical abstraction for data containers. The bottom layer includes an Object-based Storage Device (OSD) [19, 20], an advanced type of storage implementing an object-based interface that has a built-in access control mechanism.

In OAIS, AIPs are the information objects that are passed to and from the storage component. Our PDS supports both a direct API as well as a web services API. We chose a standards-based web services API since web services are platform independent and support clients built within different environments; this is an important feature in preservation environments.

Our proposed PDS is constructed of two processes. The higher level process includes the upper two layers and is responsible for implementing OAIS functions, providing the interface to the outside world, and incorporating XAM functionality. The lower level process includes the bottom layer and provides the storage in the form of the object store.

The higher level process is composed of a stack of the following components:

- **Preservation Engine** – provides the external interface and the OAIS specific preservation semantics.
- **XAM Library –** generates logical container objects of data and metadata under a common globally unique name.
- **XAM to OSD** – a bridge that maps the logical XAM objects to physical OSD.

Since the higher level process needs to support HTTP, web services, and security, to interface with clients of the PDS, the box may utilize an application server. Offloading an application server to the storage box supports the new paradigm of moving the application to the data instead of the traditional paradigm of moving the data to the application.

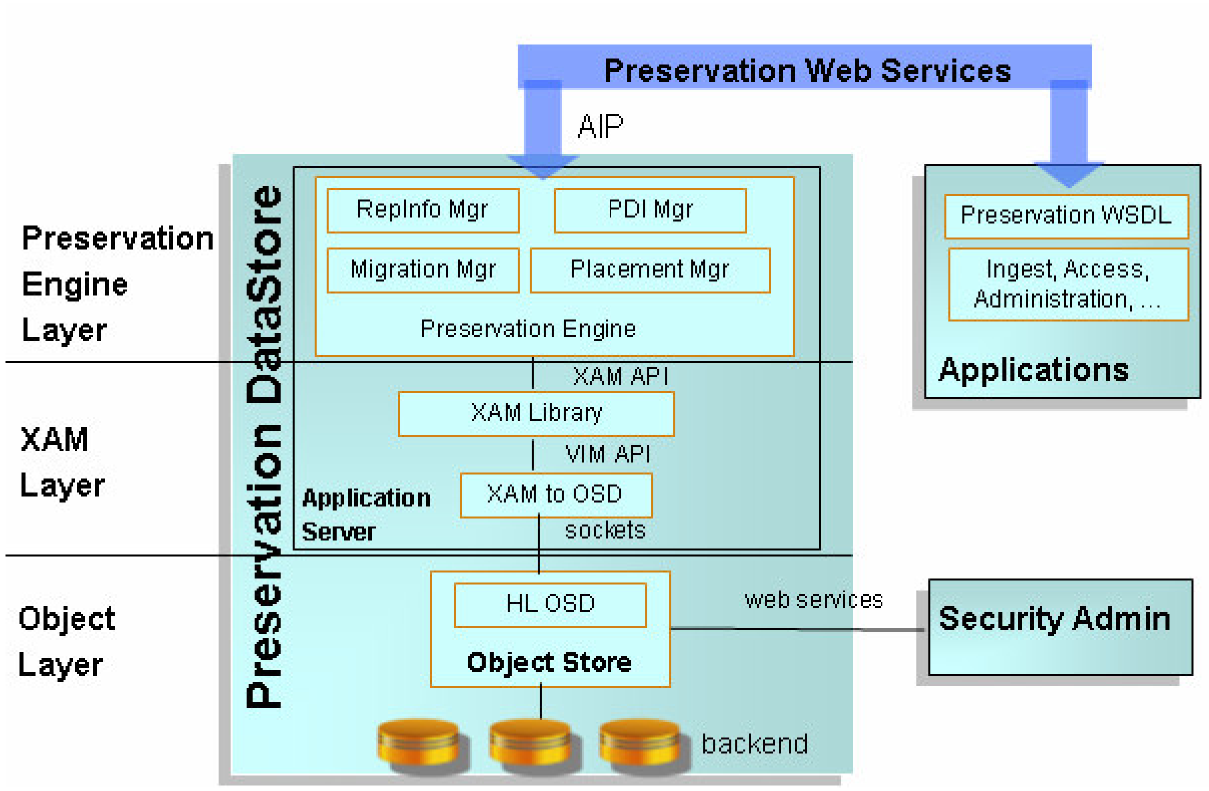

Figure 2. Preservation DataStores architecture

The second process in the PDS box serves as the object layer and includes an OSD component. The OSD process requires periodical communication with the security admin, an external component used to obtain security credentials that enforce access control. In the proposed architecture, the OSD calls a third-party security admin via an API or web services.

The object-layer can also be materialized using a standard file system as the underlying storage instead of an OSD. The chosen architecture builds on an OSD as its object-layer since an OSD is naturally designed to support an object store layer [21]. It handles and manages the space allocation of an object and it associates attributes at the storage level in an optimal manner. This allows optimizations such as placing some key OAIS attributes (e.g., a link to the representation information) close to the data in a persistent manner. Furthermore, in cases where the actual disks are network attached, an OSD provides object-level secure access control to networked disks.

4.2. Preservation Engine layer

The preservation engine component provides the external API, implements the OAIS abstractions and provides the preservation characteristics. In this section, we describe the unique features supported by the preservation engine layer and elaborate on the value of realizing them in the storage. A policy manager, either internal or external will drive the application of these features.

In addition to these specific features, a major role of the preservation engine layer is to perform a mapping between the different layers of abstraction. At the top, the preservation engine layer uses OAIS concepts to communicate with its clients. In turn, the preservation engine builds upon XAM underneath to implement its function. Thus it must map between the OAIS and XAM levels of abstraction. We describe this mapping in detail in Section 4.5.1.

4.2.1. Managing availability/data loss. One key responsibility of the preservation aware storage is to manage the availability of the data entrusted to the storage. Traditional storage systems manage data availability (e.g., RAID, remote mirroring etc.) at the block level. Instead, PDS must manage availability at a *higher level,* specifically at the level of an AIP. A basic assumption is that some data will be lost over time; thus a basic requirement is that the degree of information loss should be proportional to the number of bits lost. This is difficult to achieve with systems that provide availability at the block level.

Not all entities stored by the PDS are of equal importance. For example, some content data objects may have very high inherent value and some metadata (e.g., representation information for a common format) may be the key to interpreting large amounts of data. We expect that a policy manager will direct the PDS regarding the inherent importance of the content data objects while the PDS itself can determine the importance of the metadata it manages. Propagating this knowledge to the PDS allows the storage system to employ techniques such as grouping related objects and creating copies of objects, both within a medium and across media.

For composite objects such as AIPs, if any sub-component is lost, access to the entire composite may be lost. This is of particular importance when dealing with data stored using offline media (e.g., tapes), which can be physically moved between locations. To reduce the risk of data loss, the AIP sub-objects are placed together physically. The preservation engine layer aggregates the AIPs into clusters, such that each cluster is self-contained, namely AIPs that reference each other are put on the same cluster. Then, each cluster is placed on the same media unit (e.g., tape volume) to maintain physical co-location. The media unit can sometimes be a logical unit (e.g., three adjacent tape volumes), but we avoid situations where the data is completely distributed. In addition to this co-location of related data, the PDS may also make physical copies of information that is deemed of high importance.

The alternative to having the PDS manage the availability of AIPs would be to have the client of the storage manage the physical organization of the data. Such a breaking of abstraction would greatly increase the complexity of the client and would be a step backwards since in most applications the application is not aware of the physical data placement.

4.2.2. AIP transformations. Another key aspect of long term digital preservation is the need to migrate data objects being preserved. This may be triggered by changes such as media decay, obsolescence of hardware or software, or a change in the copyright or external environment (e.g., organization). Some types of migration entail simply copying a set of bits from one medium to another to ensure the bits are preserved. It is natural for this type of migration to be handled by the storage, since the storage is aware of the media characteristics and can monitor the health of the media. It also saves bandwidth because the data does not need to be read by a client and written by the client to a different media.

Another type of migration is transformation, which is the focus of logical preservation. Transformations involve changing the bits of an existing data object (e.g., transforming data in a format that is about to become obsolete into the replacing format). While the storage should not be aware of the semantics of the transformation, it is natural for the storage to apply the transformations. This can be done at the same time as data is migrated for bit preservation, saving accesses to the physical medium. Alternatively, it can be done on demand; prior to returning an AIP to the client, the format can be transformed into one the client recognizes. The storage should support transformations since this is a data intensive function and it can improve overall system performance by performing the computation for the transformation close to the storage. It also minimizes the risk of data loss involved in massive data transfers for transformation purposes.

4.2.3. AIP identifier generation. An AIP must have a persistent globally unique identifier. In PDS, this identifier is composed of a logical ID, a copy number and a version number. The logical ID is the same identifier for all the copies and versions of the same AIP, and can be used to find the AIP in the preservation system. It can be constructed using a standard for persistent identifiers such as Digital Object Identifier (DOI). The copy number identifies the various copies of the same bit-wise AIPs while the version number identifies the various versions originating from the same AIP (i.e., transformations create new versions of AIPs).

It is important for the AIP ID generation to be supported by the preservation storage for two reasons. First, the preservation storage is a centralized resource used by all preservation clients and thus is a natural location for ensuring the uniqueness of IDs. Second, as described above, the preservation storage will at times create new AIPs, either through transformations or through copying. Since it is creating new AIPs based upon existing AIPs, it is natural for the generation of the AIP ID to be handled by the storage.

4.2.4. Storlet container. Digital preservation systems need to periodically perform data validation and data transformation procedures. To perform these data-intensive tasks, traditional systems move the data across the network to the application side. Once the procedures complete, the system moves the data back to the storage component. Similar to the ABACUS system [22] and active disks [23], we propose to better utilize the locality property and perform these data intensive procedures within the storage. Therefore, the preservation engine includes a module container that can embed and execute restricted modules with pre-defined interfaces. We use the term ‘storlets’ (like applets in applications and servlets in servers) to describe these deployable restricted modules. Storlets, such as transformation modules and fixity computation modules, can be deployed in PDS and later on executed either periodically or when explicitly triggered.

4.2.5. Manage preservation specific metadata. OAIS specifies a set of metadata that is needed for AIPs. By having the management, and in some cases the computation, of this metadata handled by the storage component, we can offload work from the client, avoid transferring data back and forth between clients and the PDS, and ensure a consistent and correct implementation

One important set of metadata is the Preservation Description Information (PDI). The PDI includes the *fixity* (an integrity check value). Computing the fixity within the PDS prevents unnecessary data transfers and ensures a consistent implementation for all objects. In addition, the PDS will be executing data transformations, which change the fixity value; hence, we need to know how to calculate fixity. The fixity algorithm may be updated by the user (using the storlet container) or chosen when there are several options.

Provenance is a second class of metadata in the PDI. Provenance events may be triggered internally (e.g., replication, transformation, etc.), making it natural to manage the provenance data within the storage. Other events, such as a change of ownership, are triggered from outside the archival storage and an API is called to inform the preservation engine of the event. If the AIP was already moved to offline media, the preservation engine records the provenance events that occur between migrations and is responsible for updating the preservation data on the next migration. If the AIP is online, the updates may take place upon occurrence.

Another important class of preservation-specific metadata is Representation Information (RepInfo), which is the metadata needed to interpret the content data that is being preserved. The RepInfo is by itself an AIP, which can be referenced by multiple other AIPs. The RepInfo includes references to additional RepInfos needed to interpret it. This forms a RepInfo network where some of its links may refer to RepInfos stored in external registries. To ensure the availability of the data it manages, the PDS validates the external RepInfo links and optionally copies them from the external registries to the PDS to maintain physical co-location (as described above). Multiple AIPs with the same RepInfo should be clustered together on the same medium. This allows the system to share the RepInfo among content objects with the same representation and manage the availability of the RepInfo by creating the appropriate number of copies. Since the storage layer is aware of the mapping of data to physical media, it is natural to have the management of RepInfo offloaded to the storage.

The Preservation Description Information (PDI) has a special kind of RepInfo. This is the representation information for the metadata generated by the preservation engine itself and thus should be managed by the storage component.

The PDS also ensures the referential integrity of the links in the PDI and RepInfo elements. This includes links to external registries, URLs, and AIP IDs. Each of these references needs to be validated. This validation should take place on ingest, access, and each time the AIP is accepted into the system (during ingestion, transformation etc.). This validation of referential integrity is essential in ensuring the availability of the data.

4.3. XAM layer

While PDS exposes high level complex logical objects such as AIP, existing storage systems (object, file or block) expose much lower-level interfaces and do not provide the necessary abstractions. Our architecture uses a middle-layer of abstraction to mediate between the lower storage system and the preservation engine. We chose to base this mid-layer on XAM [18]. This section describes XAM and why it was chosen as the mid-layer abstraction in the PDS architecture.

A XAM storage system contains one or more *XSystems*, where each XSystem is a logical container of *XSet* records. An XSet, which is the basic artifact in XAM, is a data structure that packages multiple pieces of *XSet fields* (data and metadata), all bundled together for access under a common globally unique external name, called an *XUID*. There are two types of XSet fields: *property* fields that usually contain metadata and *XStream* fields that contain unbounded byte streams of a valid MIME-type. Each XSet field has a fixed set of attributes that are manipulated via get/set methods.

The XAM architecture allows applications to use the XAM API to store and retrieve information in a vendor-independent and location-independent manner. The top software module in the XAM architecture is the XAM library, which exposes the XAM API. This module interacts with the application and provides a view of the underlying storage through the entities of the XAM world. Under the XAM library resides the Vendor Interface Module (VIM), which acts as a bridge between the XAM standard APIs and the vendor storage systems.

We chose to use the XAM emerging standard in PDS since it adequately addresses many of the special needs of a preservation aware storage. Choosing a standard rather than inventing an internal layer is in line with the basic design principle of presentation systems; open standards are more likely to be robust, system-independent, last longer, and support interoperability. The preservation needs addressed by XAM are as follows:

- In preservation aware storage, the metadata is crucial for interpreting the data retrieved and therefore it is at the same importance level as the data. XAM enables bundling large amounts of data and metadata in an XSet and raises data and metadata to the same level by representing both as XSet fields. Note that the OAIS defines the AIP as a working unit in the storage that encapsulates data and metadata. XSet may serve as a "building block" for AIP complex structures.
- Each XSet is associated with a globally unique ID, which enables the preserved data to be location independent. Furthermore, XAM has a built-in import/export service for XSets. This in turn allows the preserved data to be efficiently managed and migrated from one system to another over time, facilitating transparent media and technology refresh cycles in long-term archives. The technology refresh may be done by replacing the VIM without any change to the application.
- XAM is specialized for reference data that is written once and primarily referenced later. As mentioned above, each XSet field has a binding attribute. If this attribute is set and the value of the field is changed, the existing XSet remains unchanged (with the same XUID) and a new XSet containing the modified content is generated. This mechanism provides an easy way to generate new versions of the content while keeping the old version of the content intact. This property is important for preservation systems, since preservation environments mostly accumulate data and do not overwrite it.
- XSets are a single transactional unit that can either be committed as a whole or abandoned. This is important, since operations on the preserved data (e.g., ingest), are expected to be comprised of many sub-operations on the data and its associated metadata. For large amounts of data and metadata, performing operations such as ingest as a single transaction can potentially be implemented efficiently using the transaction support of XAM.
- XAM has a built-in import/export service based on a standardized self-describing format for XSets. We leverage this service to create a self-describing, self-contained format (SD-SCDF) for AIPs written to a portable medium. This allows media containing preserved data to be shared across time and space. The self-describing property allows the AIPs to be extracted from the medium and the self-contained property ensures that the necessary information is available to interpret the data.
- XAM provides integrated basic storage management capabilities such as placement management. These capabilities may facilitate supporting different management policies in the preservation aware storage.
- XAM is an emerging standard. Standards should be used in preservation aware storage in order to facilitate interoperability.

4.4. Object layer

As described in the previous section, the XAM standard provides us with a high-level storage abstraction needed for the preservation engine. The XAM library interacts with the supplied underlying storage system via the VIM API. The VIM maps XAM entities such as XSystems, XSets, XSet fields and their attributes to the vendor storage system entities.

The selection of the storage system is driven by its compliance with XAM and, in addition, must also support the special needs of a preservation aware storage. While we intend to architecturally support both file systems and object stores via the cross-VIM portability of XAM, we believe that Object Stores (OSD) are better suited to the special needs of preservation aware storage:

- In an OSD, metadata is an integral part of the object and is managed, stored persistently, and recovered with the object's data. The fact that the importance of the metadata penetrates all the way to the object layer is an advantage for the preservation aware storage.
- The OSD security model provides security per object. This enables the use of networked storage while supporting fine-grained secure access control to the storage. In a preservation system, which stores vast amounts of data, the ability to store it on a scalable storage network is essential.
- OSD enables us to include application hints that will denote OAIS “hot” attributes such as the attribute that links to a RepInfo object. The OSD will use these hints to improve storage performance and robustness.
- OSD is an industry standard. Standards are preferred in preservation aware storage in order to facilitate interoperability.

- OSD provides a natural and adequate mapping to XAM.

4.5. Mapping between layers

A key point in the PDS architecture is the use of standards in each of the layers. In the Preservation Engine layer, we exploit the OAIS reference model as the top level standard. We implement OAIS concepts utilizing the XAM emerging standard, which in turn is mapped to the OSD standard at the object layer. As a result, the complex structures of the OAIS implemented in the high-level Preservation Engine layer are gradually mapped to the simpler structures of the object layer. In this section, we describe the representation of OAIS objects in the Preservation Engine layer and propose their mapping to the underlying layers. More specifically, we describe the representation and mapping of an AIP.

4.5.1. AIP representation in Preservation Engine.

We define a hierarchical representation of an AIP. The basic building-block of an AIP is the *Information Object,* which consists of a single instance of content data, the raw data being preserved, and RepInfo to interpret the data. In the PDS architecture, the RepInfo is also represented by an AIP. This is done for two main reasons. First, being a shared resource in the archival storage, the RepInfo element has to be an independent unit identified by a unique ID. Second, RepInfo objects consist of content data and RepInfo (forming the recursive RepInfo network), and also need to be preserved. By representing RepInfo with an AIP, we can utilize an existing mechanism to preserve the RepInfo, while keeping the design simple. Consequently, each Information Object contains embedded data for its content and an AIP reference for its RepInfo.

As depicted in Figure 3, an AIP contains elements of two types: *Content Information* and *PDI*, both inherit from the Information Object. Content Information is a simple Information Object. A PDI object is a more complex Information Object, containing four sub-elements (*Provenance*, *Fixity*, *Reference* and *Context*). OAIS does not specify whether these sub-elements are themselves Information Objects. In the PDS system, we decided that the PDI sub-elements should inherit from Information Objects because they contain data and RepInfo to interpret that data. For example, provenance data is kept in a certain format; the description of this format serves as the RepInfo of the provenance data. In other words, by representing the elements of the PDI themselves as Information Objects, we enable the PDS to use the same mechanism to preserve user data and its own metadata.

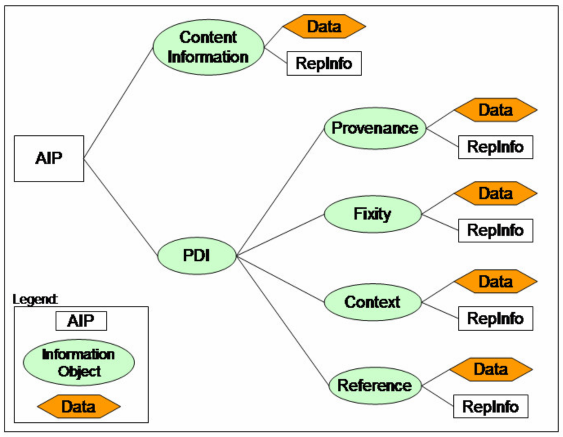

Figure 3: Representation of AIP in Preservation Engine layer. AIP consists of Information Objects that contain data and RepInfo, which is also an AIP.

4.5.2. Preservation Engine Mapping to XAM.

From a high-level perspective, an AIP is naturally mapped to an XSet. The AIP contains pieces of data and metadata bundled together under a unique ID. Recall that the XSet offers a unique ID (XUID) and may contain pieces of data and metadata stored as its XSet fields.

Looking at the AIP content, an AIP is constructed of Information Objects. Within an Information Object, the content data (an opaque byte stream) is naturally mapped to an XStream. The RepInfo reference is mapped to a property of type XSet reference (XUID), since each RepInfo is represented by a separate AIP and therefore mapped to a separate XSet.

The challenge lies in mapping the Information Object itself. How do we tie together its contained objects to enable a sound mapping of the AIP inner-structure while using XAM correctly? The AIP has a hierarchical and complex inner-structure, while XAM is oriented towards a flat structure and aims to contain many XSet fields in a single XSet.

One option is to map each Information Object to an XSet. In this option, the AIP is mapped to an XSet that may contain two XSet references – one for the Content Information and one for the PDI. The PDI XSet, in turn, contains four XSet references (Provenance, Fixity, Reference and Context). Although true to the AIP structure, this mapping enforces the creation of many sparse XSets and thus does not follow the XAM spirit. Namely, it does not exploit the advanced encapsulation capability of XAM while still paying the overhead of maintaining complex abstractions.

We chose to implement an alternative option in the PDS. We avoid mapping the Information Objects and place their contained objects directly under the AIP XSet as XSet fields. In this manner, the AIP hierarchical structure is completely flattened. A naming scheme maintains the grouping of XSet fields to Information Objects (see Figure 4). This mapping better exploits the XAM object structure (taking advantage of the fact that an XSet is a container of multiple fields of multiple types). Moreover, mapping an AIP to a single XSet enables transactional operations on AIPs. Operations on AIPs (e.g., ingest AIP) are complex and composed of many sub-operations. Having the AIP as a single transactional unit guarantees executing each complex operation on an AIP as a single transaction.

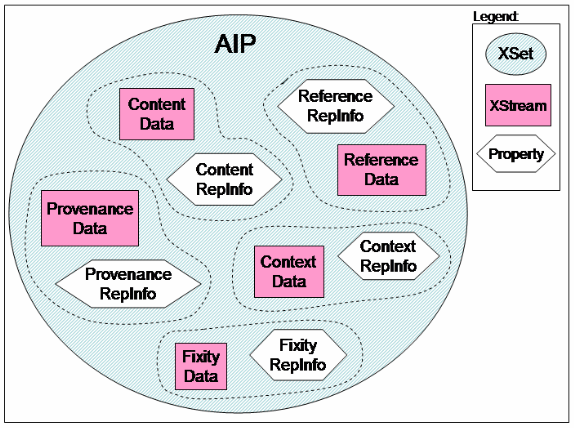

Figure 4: Flat mapping of AIP to XSet. AIP maps to XSet. Content Data and RepInfo objects map to XSet fields. Their grouping to Information Objects (illustrated by the broken lines) is kept by a naming scheme.

Above, we suggested that each content data naturally maps to an XStream, assuming that the content data is an opaque byte stream. This assumption is correct for the Content Information data. However, the PDI sub-objects' data have a complex inner-structure. Fixity and provenance data consist of a series of events, whereas context data contains different pieces of data and references. If each such data is mapped to a single XStream, we will incur additional complexity in parsing these XStreams on each access. Nevertheless, such an implementation would keep the mapping simple.

An alternative is to map each such data to a group of XSet fields, parsing the data at the XAM level. This means each piece of data or reference (e.g., each provenance event) is mapped to a separate XSet field. The mapping is more complicated, but enables easy search and access to these pieces of data and reference.

We also looked into which XAM attribute values should be assigned to each XSet field, in particular the use of the "binding" vs. "non-binding" qualifier of XSet fields. Recall that by setting a binding attribute to a field, XAM implies that any edit/addition/removal of this field results in the automatic creation of a new XSet. Hence, the binding qualifier determines the conditions and/or events under which a new AIP will be generated.

Since any change to the Content Information will result in the creation of a new version of the AIP while keeping the original AIP intact, we map the Content Information's content data and RepInfo as binding objects. The RepInfo of the PDI sub-objects are also set to binding. For example, a change to the fixity RepInfo may be an update of the fixity algorithm and should cause the creation of a new version of the AIP.

On the other hand, the handling of PDI sub-objects' data is more complex. For example, we believe that unlike external provenance events (e.g., a change of ownership) that should cause the creation of a new AIP, the internal generation of provenance events should not. Such an internal event may occur during media migration, where a provenance event is added by the Preservation Engine to document the migration. Therefore, if provenance data is mapped to a single XStream, it should map as non-binding. When external events are added, the creation of the new XSet must be handled by the Preservation Engine layer since it will not be automatically handled by XAM. If each provenance event is mapped to a separate XStream, external events map as binding while internal events map as non-binding.

One benefit of the use of XAM as described above is in supporting a self-describing, self-contained data format (SD-SCDF) for AIPs written to a portable medium. A cluster of AIPs is mapped to a XAM object (XSet) with properties that include references to the various AIPs in the cluster. By that, we take advantage of the XAM export format and propose that one way to implement SD-SCDF will be to use XML-binary Optimized Packaging (XOP). The SD-SCDF will include the references and offsets to the various AIPs of the cluster, followed by an attachment that includes a serialization of each AIP. This allows parallel access to the AIPs if desired. The serialized AIP will follow the mapping to XAM objects and include a description of the compact parts of the AIP (e.g. references to the various RepInfos), followed by a table of contents attachment to list the offsets to each of the streams. The table of contents is followed by a MIME attachment for each one of the streams (e.g., content data object stream, reference data stream, provenance stream, context stream, fixity stream).

Note, the SD-SCDF format has by its own RepInfo that needs to be preserved, such as the specifications of XOP, MIME, XAM, etc.

4.5.3. XAM mapping to OSD. There is an ongoing effort in the OSD working group in SNIA to propose a mapping of the XAM standard to OSD. To date, the work that has been done offers a mapping of XAM objects and their attributes to OSD objects and attributes. Other issues, such as security, have not yet been addressed.

The XAM to OSD mapping to be used in the PDS system is based on the SNIA proposal and is generic; namely, it has no preservation specific characteristics. It aims to provide a natural and efficient backend support to XAM.

5. Discussion and future work

The Preservation DataStores architecture proposed in this paper is currently under prototype implementation. The implementation will be tested and validated within the CASPAR project with large heterogeneous data sets provided by our CASPAR partners. This includes earth observation data from the European Space Agency and cultural data from UNESCO and artistic data. These testbeds are of diverse formats including text, binary, imaging, videos from diverse domains including scientific, cultural and artistic domains.

One of the difficult issues in preservation systems is its validation methods. How do you validate today that your system actually preserves data for a long time when the future is unpredictable? A primary objective of the CASPAR project is to design such validation methods; these methods will also be utilized for PDS validation.

Another issue for future exploration is the use of PDS in a federated environment that replicates data across multiple independent storage systems to support bit preservation. In addition, the federated environment maintains indexes to facilitate fast access to the data. Those indexes can be in databases for structured data or can be inverted lists type of indexes for unstructured data. Some open issues in this area include how to generate a federated index for the data, how to preserve the index, how to update the search methods as they evolve in the future, should the index be co-located with the data or managed separately from the data, and so forth.

6. Conclusions

There is an increasing need for preservation systems that can preserve myriad types of information for tens and hundreds of years. Such systems will be more robust and have a lower probability of data corruption or loss if they utilize a storage component with built-in support for preservation.

In this paper we describe Preservation DataStores, a novel storage architecture for OAIS-based preservation aware storage. The architecture is currently under development as an infrastructure component of the CASPAR project. It is a stacked architecture composed of three layers: the preservation layer, the compound object layer and the stored object layer. The architecture is based on three open standards, the OAIS, XAM and OSD, which are linked via generic mappings.

This architecture transforms the logical concept of an information object, which is a basic block in preservation systems, into a physical storage object. This allows the storage component to “understand” OAIS structures and functions. This knowledge, when propagated down to the storage layer, can lead to optimized and more robust implementations.

A key design point in this architecture is the choice of open standards. Choosing standards rather than inventing components is in line with the basic design principle of preservation systems in general; open standards are more likely to be robust, system-independent, last longer, and support interoperability.

Finally, this paper introduces the idea of marrying the OAIS with the XAM model. As such, this is a demonstration of a first application of the new and emerging XAM model.

References

[1] ISO 14721:2003, Blue Book. Issue 1. CCSDS, 650.0-B-1: Reference Model for an Open Archival, Information System (OAIS), 2002.

[2] M. Factor, D. Naor, S. Rabinovici-Cohen, L. Ramati, P. Reshef, and J. Satran. "The Need for Preservation Aware Storage - A Position Paper". *ACM SIGOPS Operating Systems Review*, Special Issue on File and Storage Systems, Volume 41, Issue 1 (January 2007), pages 19-23.

[3] CASPAR - Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval. See http://www.casparpreserves.eu/.

[4] M. Baker, K. Keeton, and S. Martin. “Why traditional storage systems don’t help us save stuff forever”. Technical Report 2005-120, HP Laboratories Palo Alto, June 2005.

[5] M. Baker, M. Shah, D. Rosenthal, M. Roussopoulos, P. Maniatis, TJ Giuli, and P. Bungale. “A fresh look at the

reliability of long-term digital storage." In *Proc. European Systems Conference (EuroSys),* April 2006.
[6] Reagan W. Moore and Richard Marciano. "Building preservation environments". In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries,* pages 424–424, New York, NY, USA, 2005. ACM Press.
[7] Mark W. Storer, Kevin M. Greenan, and Ethan L. Miller. "Long-term threats to secure archives". In *Proceedings of the 2nd International Workshop on Storage Security and Survivability (StorageSS 2006)*, Alexandria, VA, pages 9–16, October 2006.
[8] Reagan W. Moore, Joseph F. JaJa, and Robert Chadduck. "Mitigating risk of data loss in preservation environments". In *MSST '05: 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'05)*, pages 39–48, Washington, DC, USA, 2005. IEEE Computer Society.
[9] SRB – Storage Resource Broker. See http://www.sdsc.edu/srb/index.php/Main_Page
[10] iRODS - Intelligent Rule-Oriented Data management System. See http://irods.sdsc.edu/index.php/Main_Page
[11] Erik Oltmans, Raymond J. van Diessen, Hilde van Wijngaarden. "Preservation Functionality in a Digital Archive". *Digital Libraries, 2004 ACM/IEEE Joint Conference on (JCDL'04*), 2004, pp. 279-286.
[12] Lorie, R, "A Project on Preservation of Digital Data". RLG Digi News, 5,3 http://www.rlg.org/preserv/diginews/diginews5-3.html#feature2
[13] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. "Provenance-aware storage systems". In *Proceedings of the 2006 USENIX Annual Technical Conference*, June 2006.
[14] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, Kaladhar Voruganti, "POTSHARDS: Secure Long-Term Storage Without Encryption," In the *Proceedings of the 2007 USENIX Technical Conference*, June 2007.
[15] Andres Rodriguez. "Preserving the Last Copy: Building a Long-Term Digital Archive, a white paper". Archivas, Inc., 2004.
[16] Michael Peterson. "The Coming Archive Crisis, a white paper". SNIA Data Management Forum, Nov. 2006.
[17] The National Digital Information Infrastructure and Preservation Program - The Library of Congress.http://www.digitalpreservation.gov/.
[18] SNIA - Networking Industry Association, Data Management Group, XAM (Extensible *Access Method).* See http://www.snia-dmf.org/xam/
[19] SNIA - Storage Networking Industry Association. OSD: Object Based Storage Devices Technical Work Group.
[20] International Committee for Information Technology Standards (formerly NCITS), SCSI Object-Based Storage Device Commands (OSD). Document Number: ANSI/INCITS 400-2004, technical editor: R.O. Weber, December 2004.
[21] Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, and Julian Satran. "Object storage: The future building block for storage systems - a position paper". In *Local to Global Data Interoperability - Challenges and Technologies*, Sardinia Italy, pages 119–123, June 2005.
[22] Khalil Amiri, David Petrou, Gregory R. Ganger, and Garth A. Gibson. "Dynamic Function Placement for Data-intensive Cluster Computing". In *Proc. of 2000 USENIX Annual Technical Conference*, San Diego, June 2000.
[23] Erik Riedel, Christos Faloutsos, Garth A. Gibson, and David Nagle, "Active Disks for Large-Scale Data Processing". In *IEEE Computer*, June 2001.