Re-thinking Fedora's storage layer: A new high-level interface to remove old assumptions and allow novel use cases*

Aaron Birkland[1] and Asger Askov Blekinge[2]

[1]Cornell University Ithaca, NY USA
[2]State and University Library Aarhus, Denmark

March 1, 2010

Abstract

Traditionally, the pluggable storage interface in Fedora has followed a "low-level" paradigm where objects and datastreams are presented to the storage layer as independent, anonymous blobs of data. This arrangement has proven simple, reliable, and generally flexible. In the past few years however, there has been an increasing need for Fedora to mediate storage in more complex scenarios.

Managing large numbers of big datastreams, multiplexing storage between different devices or cloud storage, and archiving content in a transparent manner are tasks that are difficult to achieve through Fedora currently. One reason for difficulty is that all storage, indexing, and transaction logic is essentially hard-coded in Fedora, and storage plugins are limited in what they can deduce from the anonymous blobs they are given.

To evolve the current Fedora architecture and address existing storage limitations, we present an alternate "high-level" interface in which implementations are aware of the structural semantics of a Fedora object, and operations are performed on a whole-object basis. Such an architecture pushes the storage logic into external plugins, allowing greater flexibility in adapting to the needs of preservation, performance, cloud, or analysis-oriented use cases. In addition to storage concerns, this layer provides an appropriate extensibility point for data-oriented approaches to indexing (asynchronous or synchronous), caching, messaging, policy enforcement, and locking or transaction strategies.

The Fedora committers have agreed that further development of the proposed high-level storage interface is a logical next step in the evolution of Fedora's internal architecture. Besides presenting the current state of design and implementation of the interface, we hope to engage the community at large to solicit feedback for current and future revisions.

*Submitted to the Fedora users group at OR10: The 5th International Conference on Open Repositories. Madrid, Spain. 6-9 July 2010

1 Introduction

There are many ways to manage digital content with the Fedora repository. Fedora objects provide a mostly uniform encapsulation and service binding over content that can be stored and managed in many different ways. Out of the box, Fedora offers the capability to accept datastream content and store/manage it itself in Managed datastreams, as well as referencing content managed elsewhere in External datastreams.

To date, Managed datastreams in Fedora have been implemented in a very simple manner. Object XML and datastreams are serialized as blobs with opaque identifiers, and sent to a low-level storage module in a series of independent operations. This low-level storage module has always been a pluggable component - many implementations have been written for various storage devices.

Unfortunately, this architecture essentially hard-codes several storage assumptions, and makes it difficult to store content in in a dynamic or intelligent manner. For example, if one wanted to store content in different devices based upon some criteria (multiplexing use case), it is not clear where this multiplexing logic should be. The low level storage module is given very little information about the nature of the blobs it is asked to store, and would potentially need to reflect into blob content in order to understand its nature. In addition, storage logic is currently intermixed with tasks such as object validation and maintaining indexes.

Providing a simple high-level object interface beneath the management layer in Fedora allows for a cleaner design that encourages modularity and separation of concerns. Indeed, we conceive HighLevelStorage being implemented as a series of modules adhering to the same (or similar) interface chained together[1] into a pipeline, each performing a very specific task such as lock management, caching, blob serialization, blob storage and multiplexing (e.g. in Akubra[2]), and providing data to synchronous or asynchronous processing units such as an indexer or messaging bus.

2 The `HighlevelStorage` layer

This paper proposes the addition of a high-level storage layer between the object management code and storage. This is an internal, fedora-specific interface useful for developing Fedora storage models, and is not intended to be a generic or portable storage interface such as Akubra.

Perhaps it is best to describe this layer in terms of Fedora architecture and separation of concerns. Below is a description of various proposed or existing components within Fedora, and the tasks that could or should be performed within each layer. We start with the highest-level management layer, and work our way down to the lowest storage layer. Each layer is characterized with a brief descriptive label, and the most relevant internal java interfaces/components are listed.

Operational logic (`Management`)

- Translate API operations into specific change sets for an object
- API-oriented messages and policy enforcement
- Manage timestamps and propagate writes to `DOManager`
- Generate new audit entries

Object-level accounting (`DOManager`)

- Manage datastream versioning
- Add audit records to object
- Set properties or object state
- Internal registries
- RELS-EXT manipulation (add/purge relationship)

Data and Storage logic (`HighlevelStorage`)

- Assemble data for multiplexing decisions
- Grouping of components into file archives
- Transactions and atomicity, and locking strategies
- Serializing
- Indexing and caching
- Data-oriented messages and policy enforcement
- Sending data to storage implementation

Storage impl. (Akubra or non-blob storage)

- Connect to storage implementations
- Store bytes, manage failures
- Present storage device as a transactional resource

In order to perform functions such as multiplexing decisions, serialization, locking, etc, this module necessarily has to be aware of the logical structure of fedora objects, and have an awareness of operations in at least a whole-object scope. Thus, the interface it presents to the object logic components must not be blob-based.

```
interface ILowlevelStorage {

 void addObject(
        String objectKey,
        InputStream content);

 void replaceObject(
        String objectKey,
        InputStream content);

 void removeObject(
        String objectKey);

 InputStream retrieveObject(
        String objectKey);

 void addDatastream(
        String dsKey,
        InputStream content);

 void replaceDatastream(
        String dsKey,
        InputStream content);

 void removeDatastream(
        String dsKey);

 InputStream retrieveDatastream(
        String objectKey)

}
```

Figure 1: Existing low level storage interface (abridged)

Figure 1 summarizes the existing low-level interface, and figure 2 contains proposed high-level interfaces. As we can see, the proposed HighlevelStorage interface is oriented towards pid identifiers and whole-object representations while the ILowlevelStorage is oriented toward opaque identifiers and InputStream blobs

```
interface HighlevelStorage {

 Result add(
        DigitalObject object);

 Result update(
        DigitalObject oldVers,
        DigitalObject newVers);

 Result remove(
        DigitalObject oldVers);
```

```
 DigitalObject read(
        PID pid);

}
```

Figure 2: Proposed high level interface. May be further split into `Writable` (upper) and `Readable` (lower)

For the high level interface, `DigitalObject` is a logical representation of the fedora object that also serves as an access point to datastream content. `Result` is some data structure that may contain relevant information relevant to the current operation. The intent is to return information such as an operation ID that may be used to determine status, coordinate workflows or interact external processes that may result from the given Fedora operation.

Please note that while the core concepts are unlikely to change, we expect the continual evolution of the high level interface as presented in figure 2. As an example, there is discussion[3] in the Fedora community related to object versioning. The idea has been put forth that versioning is a storage-layer concern and should not be part of the fedora object model. If this discussion progresses and the community supports the principle of storage-initiated versioning, then the HighLevelStorage interface may need to be extended accordingly.

Likewise, for development of add-on modules such as indexing or messaging, it may make sense to further split `HighLevelStorage` into `Writable` and `Readable` as indicated in figure 2. The motivation for this split is that some functionality, such as maintaining an index, should not be be concerned with retrieving object representations.

3 Conclusion

A high-level storage interface in Fedora would be an enabler for a number of use cases that are currently difficult to achieve in Fedora, such as

- Multiplexing, particularly when combined with the Akubra blob store.
- Storing entire object and all datastreams in self-contained file archives such as bag-it[4] or AtomZip[5].
- Adopting non-blob storage paradigms such as an XML database, column-oriented stores such as HBase[6] and Vertica[7].
- Implementing lock-free storage strategies allowing multiple fedora servers to operate against the same data concurrently.
- Integrating a caching layer for objects or datastreams.

Certainly, Fedora is not going to be able to solve these needs overnight. As envisioned, the new storage layer interface and architecture will primarily be an enabler, with well-defined extensibility points and the ability for modules to be assembled together like building blocks. The default configuration shipped with Fedora will likely be as simple as possible, with complexity available to those who need to adapt to complex or challenging scenarios.

As it stands now, many driving use cases have been mentioned by members of the Fedora community as being desirable features. These have been kept in mind while drafting initial proposals for the new storage architecture. That being said, involvement of the fedora community at large is essential in order to assure that Fedora continues to evolve in a way that allows its users to adapt to changes in technology. We intend to present the concepts, progress, results, and open questions relating to the high-level storage effort in the hopes of eliciting community input and seeding novel ideas or applications.

References

[1] Initial sketch of potential module chaining configuration is illustrated in http://prezi.com/kjpdt6zhcir6

[2] Akubra blob store, http://fedora-commons.org/confluence/x/LAF2

[3] Discussion on storage-oriented versioning originated at London committer meeting, 23-24 February 2010. http://fedora-commons.org/confluence/x/vB7S

[4] Bag-it specification available at http://www.digitalpreservation.gov /library/resources/tools/docs/bagitspec.pdf

[5] Atom and AtomZip serializations described at http://fedora-commons.org/confluence/x/fgBI

[6] HBase is part of the Apache Hadoop project http://hadoop.apache.org/hbase/

[7] Vertica is a proprietary column-oriented database http://www.vertica.com/enterprise-data-warehouse