Object Creation Thread

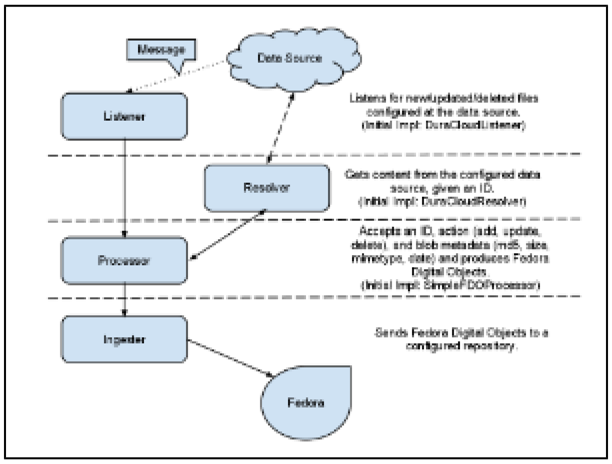
Return to parent thread

Object Creation Thread

When new content is uploaded to DfR (via DuraCloud), it is uploaded as a bytestream having a limited amount of metadata in itself. Also, DuraCloud operates with semantics appropriate to a bytestream, a desirable characteristic since this creates a logical separation of concerns between storage-like containers and containers that perform other functions. In DfR, a key container/service family is the Fedora Repository. It is able to support a richer set of semantics for marking up and relating bytestreams to other content. To do this, however, Fedora Digital Objects, using the FOXML schema, must be created encapsulating the uploaded bytestreams (stored in DuraCloud). This thread contains discussions of the process.

Design

- Sketch of four major components from Chris/Dan brainstorming call on Jan 18th. The source of this document is here.


Message
Data Source
Listener
Listens for new/updated/deleted files configured at the data source. (Initial Impl: DuraCloudListener)
Resolver
Gets content from the configured data source, given an ID. (Initial Impl: DuraCloudResolver)
Processor
Accepts an ID, action (add, update, delete), and blob metadata (md5, size, mimetype, date) and produces Fedora Digital Objects. (Initial Impl: SimpleFDOProcessor)
Ingester
Sends Fedora Digital Objects to a configured repository.
Fedora

Likely the discussion threads follow the four major components and their relationship. They seem pretty fundamental and follow good separations of concern.

A key separation of concerns was identified for the basic design. That is Fedora will be used to support rich representations of content and their relationships. It will "refer" to individual resources (usually stored content) in the form of bytestream (also called blob) containers. This the current relationship between Fedora and Akubra (and LLStore). It will also be the relationship between Fedora and DuraCloud. By using this approach, Fedora can work in similar ways with a variety of content sources, pretty much anything that can deliver a representation from a resource as a stream.

The primary purpose of the Object Creation service is to construct Fedora Digital Objects (FDO). Since the term "object" is very overloaded we will be more specific in our terminology wherever there is the potential for confusion. An FDO is primarily a small body of XML, conforming to the FOXML schema, that contains the information needed to obtain one or more resource bytestreams, versioning information, its relationship to other FDOs, one or more identifiers and some useful metadata. Some of the resource bytestreams, represented by DataStreams in the FOXML schema, may be considered to be "owned" by Fedora. This is a policy arrangement since the actual storage is delegated to another system, for now we will call "resource server". Since the FDO is an XML document, it too can be stored as a bytestream.

Fedora is a resource server, as well as DuraCloud and Akubra. The form of the bytestream supplied by the resource server is called a "representation". Resource servers provide differing abilities to create representations. DuraCloud and Akubra can provide simple representations that are generally one-to-one with stored bytestreams (storage semantics). The relationship between Fedora and other resource servers is the key integration in this design, especially ones with simple storage semantics, enable enormous flexibility, scalability, evolvability and durability. Fedora can be rebuilt entirely from the resources.

In the drawing, new bytestreams are uploaded directly into a resource server supporting storage semantics (e.g. DuraCloud). The "storage" resource server publishes a notification that new Storage Digital Objects (SDO) have been received and are available. The Object Creation Server provides a "listener" for the notification. When a notification is received by the listener, it invokes the "processor" component to create the FDO. The listener may also use a polling mechanism to locate new SDOs but this is usually much less efficient than using notification. A key relationship still to be explored is the information that may be passed along with the notification. Any information passed with the notification, particularly some resolvable identifier for the resource, makes the process simpler and easier. Additional information, best provided by the storage resource server, may also be of interest (e.g. checksum). However, we cannot assume every storage service provider can efficiently supply the same information (having the same semantics). We may be able to implement greater efficiencies if we know more about the storage service provider (see Akubra Hints).

The processor component creates one or more Fedora Digital Object(s) using any information it has at its disposal, including information in the content, from configuration information or from third party sources. It is imagined (eventually) as a customizable pipeline that invokes a set of reusable microservices, each of which contributes its capability to the flow. We will start simply for this project but over time, the capabilities of the processor (indeed the whole Object Creations Service pattern) can be extended to the limits of technology. Initially, we will create one simple FDO per SDO. Additional information can be added to the simple FDO using post processes or curatorial applications. Later, we can improve the richness of the FDO and even handle groups of SDOs in a batch. The first concrete implementation will be modular but likely a simple set of composite services and component API calls. However, the greatest use of off-the-shelf FOSS components is planned starting with bytestream characterization.

Some sort of "operating environment" may be used for implementing the processor component, for the moment using "orchestration" as a working term and "service container" for where the code operates. An earlier version of similar notions for processing items for Fedora ingest is described in Fedora Service Framework Simple Queue Services. However, for a first implementation, orchestration is likely reduced to its simplest form since we need to concentrate on focus on the major components and their relationships. Keeping it in mind will be useful since this will eventually make it easier to add or customize capabilities for creating FDOs and later for loading search/discovery mechanisms and tools for researcher's to use in a cloud close to the data. But we need to walk before we run. When paired with an operating environment have a dynamic service loading capability like the DuraCloud Service Container, we will have a compelling ability to provision and use customized functionality.

One important source of information is the SDO (at least the representations of it that are provided by the storage resource server). To create even the simplest FDO we must have an identifier for the resource or resources related to the FDO that can be provided by the storage resource server. We also need to be able to resolve the identifier in order to get the representation. This introduces the function of the "resolver" component in the diagram. This design support the use of both public and private resolvers. An important public resolver system is the World Wide Web. The Web's architecture permits private resolvers and many other resource server operate on schemes that only permit private resolution. A computer's file system is part of a kind of private resolver. The first concrete implementation will be a DuraCloud resolver using its API. The design permits any number of concrete implementations over time so support DfR evolvability goals.

Lastly, we need to ingest the FDO into Fedora. The "ingester" component provides this function and handles (or reports) the error conditions that may result from problems in the ingest process.

Related Materials

- DCS Data Model
- DCS Feature Extraction - A framework used by the Data Conservancy to process objects. Has built-in support for parallelism and error reporting.
- Fedora Service Framework Simple Queue Services
- The Tiny Filtering Framework - A small library providing source/sink/filter abstractions for working with sequences of Java objects.
- Fedora DTO Libraries - Java libraries to aid in creating, reading, and writing Fedora objects.
- Active MQ (REST binding)
- Amazon Simple Queue Services
- Stanford Library (Hydra) - Work-Do
- Mule
- Apache Camel (A good intro article)
- Apache Service Mix
- Spring (http:www.springsource.org)
- MIT Libraries - Cloud Task Replica
- JBoss jBPM
- JBoss Drools
- Pronom Droid
- Harvard FITS

Return to parent thread