Fedora

The open-source Flexible Extensible Digitial Object and Repository Architecture was developed at Cornell and the University of Virginia.

There has not been much work on best practices associated with an object repository or with Fedora. I have only found one bare-bones list. However, Johns Hopkins, UVA, and MIT have begun a large-scale evaluation of repository systems to uncover best practices and make recommendations for future development. For current status, see their project pages.

Note: This page and its children describe the implementation of Fedora in the IU Digital Library Program. Fedora is only one piece of the larger repository system. For more details about the entire repository system, start with the Repository Architecture page.

Fedora documentation

- Latest version is Fedora 2.2.1, see Fedora 2.2 for details
- Documentation for the 2.2.1 release
- Basic tutorials
- Documents from the Fedora user community

Fedora vocabulary

General

Fedora object – Think of this as an "object-oriented" object, rather than a single digitized item. It may contain an arbitrary number (and type) of media files.

Disseminator – Think "add-on methods". A named set of additional methods that are available for a Fedora object. A disseminator binds the object to a particular behavior definition and behavior mechanism. It specifies which datastreams are sent to the behavior mechanism to satisfy the user's request.

Datastream – Think "member". The data fields/elements/files that make up the Fedora object.

Behavior – Think "method".

Behavior definition – Think "interface". Lists one or more methods and their arguments, so a client knows how to interact with an object. Each disseminator subscribes to a behavior definition, which defines how the disseminator works.

Behavior mechanism – Think "implementing class". The implementation of a behavior definition. But the mechanism doesn't really implement the functionality; it is more a pointer to a web service that implements the functionality.

Datastreams

Managed datastream (M) – Fedora stores and manages the content bytestream. Disadvantage is that it is painful to access objects outside of Fedora; advantage is that you don't have to When a change is made to a managed datastream, a completely new copy of it is placed in the repository, so you always have access to both copies.

External datastream (E) – Fedora stores a reference (URL) to the content. Advantage is that you still control the content directly; disadvantage is that you have to track all the versions of an object. The user will see a Fedora URL in their browser's address bar. When a change is made to an external datastream, the repository doesn't notice. Changes to the administrative metadata are logged in the object, but probably don't matter that much.

Redirected datastream (R) – Very similar to External. Fedora stores a reference (URL) to the content, but will not mediate access to content. Any requests that Fedora receives for this datastream result in a redirect. This is useful in cases where the datastream is really some sort of streaming media that cannot be piped through Fedora, or the datastream is an HTML document with relative hyperlinks to the server on which is is normally hosted. Users will see the redirected URL in their browser's address bar.

XML datastream (X) – Fedora stores a name-spaced block of XML content within the Fedora digital object XML file, rather than storing the object in the normal datastore. When a change is made to an XML datastream, a completely new copy of the datastream is stored in the Fedora object, so you always have access to both copies.

Note that the fedora.fcfg file has an option "doMediateDatastreams" that purports to force all referenced (both R and E?) datastreams to go through Fedora before being acted on by a behavior mechanism. This provides an additional level of control/security, since the mechanism cannot act on the datastream outside of Fedora. This variable does not appear to affect any user-visible interactions.

Content models

Content model – A collection of disseminators that is applied to a given type of object. For example, all images may share an "image" content model, there may be separate content models for documentary photographs and art images, or there may be a collection-specific content model to provide some specific functionality.

Atomistic content model – A content model in which each digitized piece of an object has its own Fedora object (and Fedora pid). For a book, each page would be a separate Fedora object, and there would be a single "book" object that referenced all of the pages. This model means you have more objects, but it greatly enhances the "discovery" of objects in multiple contexts. All of the content models being developed at UVa are using this type of atomistic content model.

Compound content model – A content model in which each bibliographic record translates into a single Fedora object, with a separate datastream for each digitized component. This is similar to the way METS collects all of the representations of an object together. It can make searching more difficult than an atomistic content model. For example, it may not be simple to implement search over a collection of books so that the results actually match pages. This model results in far fewer objects than an atomistic content model.

Central Fedora web services/apps

The Fedora documentation doesn't have any place that collects all of Fedora's web-accessible features into a single list. This list isn't exhaustive either, but it's a good start:

Search system – http://hostname:port/fedora/search
Basic object access – http://hostname:port/fedora/get/PID
Object history – http://hostname:port/fedora/getObjectHistory/PID
Repository info – http://hostname:port/fedora/describe
Repository reports – http://hostname:port/fedora/report
Resource Index search – http://hostname:port/fedora/risearch
Backend security config – http://hostname:port/fedora/management/backendSecurity
OAI provider – http://hostname:port/fedora/oai
PID generation – http://hostname:port/fedora/mgmt/getNextPID
Saxon – http://hostname:port-number/saxon/SaxonServlet?source=xml-source&style=xsl-stylesheet&clear-stylesheet-cache=yes
XML-FO to PDF converter – http://hostname:port/fop/FOPServlet?source=xml-source
Image processing – http://hostname:port-number/imagemanip/ImageManipulation?url=url-of-image parameters

For information on applications external to Fedora, see Fedora Tools.

Other Fedora Implementations

The future of Fedora

Future plans include:

1. web-based submission & workflow tools
2. preservation system
3. Web IR (Institutional Repository) client
4. Cooperation with DSpace, aDORe

Determining where to implement functionality

Some repository functionality should be implemented "directly" in Fedora, through XSL transforms, or services that are only called by Fedora disseminators. We want everything in Fedora to be portable, and that often constrains what we can do. Other functionality should be implemented by the external infrastructure applications (ingest, delivery, cataloging).

A good rule of thumb is: Users will interact almost exclusively with the infrastructure applications. Fedora lives only to serve these applications. Fedora should only implment something if it is needed by multiple applications, or if it would make the writing of a single application much easier/simpler.

Keep in mind that these are the primary actions users want from our collections, and we should optimize for them (in this order):

1. open collection initial page
2. view individual item (from the PURL)
3. search
4. catalog/update item
5. ingest items
6. request archival-quality copy