How to Model a Book

It is important to note that there is no one way, nor right way to model resources in Fedora. Different approaches will be more appropriate in different circumstances, and they all come with trade-offs.

This guide represents a starting point for thinking about modeling content in Fedora.

Scenario

We want to store a book in Fedora:

- Each page of the book is a single PDF
- The book's thumbnail is a single JPEG
- A METS description is a single XML document

Structure

Fedora resources can be structured physically or logically. Generally speaking, the repository will be more performant if resources are structured logically, given the increasing performance impact with the increasing number of *direct* children of a shared parent resource. In other words, if a parent resource ("book") has a thousand pages, and each of those pages were structured within Fedora as a direct child of "book", that would probably impose negligible performance impact. However, if "book" had 10,000 children pages, then the creation of subsequent pages would likely be noticeably slower.

On the other hand, certain repository operations (e.g. authorization, nested move, nested export) are able to act over a tree of resources. For example, authorization policies can be defined to apply to all sub-resources within a tree of resources to the point until a descendant within the hierarchy overwrites the ancestor's policy.

Therefore, in some situations where the repository size is expected to be small and nested resources would be useful, it is possible that creating structural hierarchies could be advantageous.

REST Examples

Identifiers

The topic of identifiers overlaps somewhat with that of "Structure". In Fedora, you can have any number of identifiers stored as properties on a resource; however, the identifier by which a resource is retrieved is the location path at which the resource is stored within the repository. Therefore, if a resource is stored in the repository at the location: "<host>/<context>/rest/book/page0", then the external identifier of that resources is the same: "<host>/<context>/rest/book/page0".

That is both good and bad, from the perspectives of readable URLs as well as performance. Having an identifier that is semantically meaningful, clearly describing something about the nature of the resource (e.g. this resource is the first page of a specific book), can be nice at first. But invariably, the resource will either have to be relocated within the repository, or migrated to a different system, or in one way or another need to have the semantics that were embedded in its identifier changed over time.

The performance considerations (specifically for the case of adding new resources to the same parent) have been described above.

Generally speaking, within Fedora, opaque identifiers that hold no semantics should be favored over semantically meaningful identifiers. There are justifiable reasons for having structural hierarchies of resources in your repository, but having meaningful identifiers is likely not one of them. Fortunately, when creating a Fedora resource, if no user-provided identifier is included in the request, Fedora provides a default (pairtree path elements terminated with a UUID) identifier that is designed to ensure a balanced tree of resources.

REST Examples

Properties

Resources (both containers and binaries) can be adorned with properties. A property is effectively a name/value pair. All resources have a set of system properties that are managed by Fedora and not editable by users. Fedora also allows a resource to effectively have an unlimited number of user-defined properties. The "name" of the property in the name/value pair can be any term coming from any namespaced vocabulary (except, of course, from the Fedora system property vocabulary). The "value" in the name/value pair can be a URI or a literal.

The result of these name/value pairs on resources is that when a request is made on a resource (i.e. HTTP /GET) the response that is returned is a set of RDF triples (subject - predicate - object) that further describe the requested resource. The "subject" of those triples is the resource being requested. The "predicates" of those triples are namespaced terms created as the "name" in the property's name/value pair. The "objects" of those triples are the URIs or literals created as the "values" in the property's name/value pair.

REST Examples

Relationships

As may be expected, relationships between resources within, as well as external to the repository are defined like any other resource property, presumably with a URL as the "value" of the property which references the other resource. In the case where the resources within the repository are physically structured, Fedora adds properties that note the containment of the child resources (http://www.w3.org/ns/ldp#contains and http://fedora/info/definitions/v4/repository#hasParent). As a slightly more advanced use, the user can define additional default relationship properties that will be applied to parent and child resources.

REST Examples

Summary

Pulling all of these considerations together, the simplest approach to the "book" scenario would likely be to create all resources without user-provided identifiers. The "book" would be a container and the pages, thumbnail, and METS files would all be binaries. Relationships between the resources would be established (#hasPage, #nextPage, #isThumbNailOf, etc). To facilitate more robust search, the metadata packed into the METS file could be extracted out into properties. Likewise, using property terms that are defined in commonly used, linked data vocabularies will aid in cross-repository, cross-institutional normalization.

REST Examples

Resources

- Glossary
- RESTful HTTP API