AIC Use Cases: API Extension Architecture

1. Extra-repository access control

Issue

We want to apply access controls (ideally WebACs) not only to repository objects, but also to its indexes.

E.g. access to resource http://myrepo.edu/private/res1 is controlled by access policies in the Fedora repo but these policies are not honored in a triplestore or Solr index where metadata can be accessed by anyone.

We want policies to be stored in one place (Fedora) and re-used for both repo and index access.

Proposed solution

Enable single-point access to both repo resources and indexes, e.g.

http://myrepo.edu/fcr:search/triplestoreIndex01/ (where the query string can be either appended as GET parameter or POST payload)

Restrict direct access of clients to indexes via firewall rules to avoid bypassing security.

2. Content models

Issue

We want to be able to CRUD complex resources with a single or a minimal set of requests, without performing complex tasks on the client side (and rewriting the same implementation for each client).

E.g. User sends metadata and binary files via a multipart form POST request to http://myrepo/fcr:model/myns:Image/ ; multiple resources and relationships between them are created according to a content model configuration for the “myns:Image” resource type.

For the “myns:Image” model the service configuration would define:

- GET:
    - return a representation of the resource and related resources according to a “default” transform program defined for myns:Image
    - This may include complex networks involving SPARQL or Solr queries on the indexes.
- POST:
    - Create a new UID from an external UID minter service;
    - create a pcdm:Object resource and populate it with metadata provided by a JSON object in the “metadata” POST field;
    - add the new generated UID;
    - assign the resource the “myns:Image” rdf type;
    - create pcdm:Object resource and assign it a “myns:Instance” rdf type;
    - create a “myns:hasOriginalInstance” relationship between the myns:Image and the myns:Instance resources;
    - Create a LDP-NR resource from the “original” bitstream provided by POST
    - Create a “pcdm:hasFile” relationship between the myns:Instance and the LDP-NR
    - Repeat this for other possible binaries provided by the request
    - Optionally wrap the whole operation in a transaction and roll back if a step fails.

Also, we want to establish a RDFS-like type and sub-type hierarchy which can be reflected in the indexes.

E.g.

- I define myns:Image as a subclass of myns:Asset;
- I define myns:hasOriginalInstance as a sub-property of myns:hasInstance;
- using the resource created above as an example, I should be able to search for all myns:Asset resources and find the myns:Image I created;
- similarly, if I search for all the myns:hasInstance relationships for the image above, I should be able to discover the resource related by myns: hasOriginalInstance.

Proposed solution

Create a configurable service that defines content models and the actions associated to the various HTTP methods that can be performed on each content model.

3. Data and structural validation

Issue

We want to enforce input validation outside of individual client systems. This is related to #2.

This validation may include constraints for property domain and range, cardinality, uniqueness, etc.

Range validation should include both data types for literal properties and class constraints for in-repo resource properties.

E.g.

- restrict the “myns:created” property to xsd:dateTime;
- restrict the “myns:hasInstance” property to resources of type “myns:Instance” or its subtypes (structural validation);
- make myns:uid mandatory.

Proposed solution

I think that the RDFS/OWL syntax lends itself as a framework for a configuration file that defines all these validation rules.

We can define which subset of RDFS/OWL the service supports and enforces.

E.g. the example validation rules above would be expressed as OWL statements:

```
[Prefix declarations]


myns:created rdfs:range xsd:dateTime .

myns:hasInstance rdfs:range myns:Instance .

_:r1 rdf:type owl:Restriction ;

  owl:onProperty myns:uid ;

  owl:cardinality "1"^^xsd:nonNegativeInteger .
```

This syntax could also be used for the content model configuration in #2.

A service would be written to parse this syntax and translate it into validation actions.