Design - PREMIS Event Service

Implementation Proposal:

See ticket for follow-on action plan: https://www.pivotaltracker.com/story/show/70580524

The Fedora 4 repository relies heavily on metadata to describe and manage its digital content. There is a large intersection between the types of data Fedora uses in its implementation to manage data and the PREMIS standard for metadata that supports information necessary in the description of preserved digital objects. In fact Fedora currently relies on PREMIS to describe content nodes such as the properties premis:hasContentLocaiton and premis:hasSize.

The Fedora 4 repository also is built based on the intersection of several structural concepts. There is the legacy Fedora construction of **objects** and **datastreams** adhering to a content model definitions. There is the fedora 4 ontological description of abstract **resources** taking the form of **objects**, **datastreams** and **content**. There is the modeshape node type definitions extending a **base** nodetype into **file** and **folder** primary types that can be extended by the addition of **mixins**. The PREMIS 2.2 is structured similarly, defining abstractly **intellectual entities** in the form of objects classified as **representations**, **files**, and **bitstreams**. Given these similarities, Fedora is well situated to borrow many of the concepts defined in PREMIS in particular PREMIS's definition of an event ontology to capture actions taken on objects preserved in the repository.

The kinds of actions that the Fedora repository would preserve can be divided into 2 categories, 1) preserving an object's prior event history, and events generated external to fedora and 2) preserving any events performed on the object by the repository itself. In either case the PREMIS OWL ontology provides much of the vocabulary necessary for these implementations:

> "This OWL ontology allows one to provide a Linked Data-friendly, PREMIS-endorsed serialization of the PREMIS Data Dictionary version 2.2. This can be leveraged to have a Linked Data-friendly data management function for a preservation repository, allowing for SPARQL querying. It integrates PREMIS information with other Linked Data compliant datasets, especially format registries, which are now referenced from the PREMIS ontology (for instance, the Unified Digital Format Registry [4] and PRONOM [5]). Thus information can be more easily interconnected, especially between different repository databases. The OWL design of PREMIS should NOT be considered as a replacement for the XML Schema: the two of them should rather be considered complementary. Work to align the PREMIS ontology with the PROV ontology [6] is being considered."

The first case of adding previous or external events to an object could be implemented via a REST service, either leveraging the current REST services for adding properties, or creating a new REST endpoint specific for events. In the second case, where processing is triggered by repository events, the google eventing machinery subscribing to repository events could be utilized, either synchronously perhaps using the auditing module, or asynchronously by subscribing to a message queue through an extension of the fcrepo-jms-indexer-pluggable module. In some cases the eventing model may not capture with specific enough detail all the repository actions that one would wish to preserve in which case is may be necessary to couple a method explicitly with the fedora action or define and generate a new set of events for these.

As far as retrieval, in any of these cases, the event descriptions could be retrieved directly via a creating a REST endpoint that retrieves the events of a node, or by using SPARQL queries against an external triplestore set up in conjunction with the core repository, or by using SQL against a relational databaes populated by a "SQL indexer group" created as an extension within the fcrepo-jms-indexer-pluggable. Events could be described by any subject /predicate/object combination, but the PREMIS ontology already provides much of the linked data necessary. Here is an example use of the PREMIS vocabulary:

1) A fedora resource with an RDF type indicating that it is an object that maintains event information, similar to the current implementation where an RDF type is used to indicated a resource is indexable or describable by dublin core. Unfortunately there is no unique PREMIS URI lends itself totally to this description. One that may work is the #hasEvent:

<object> <rdf:type> <http://id.loc.gov/ontologies/premis.html#hasEvent> .

However it seems that this URIs primary usage should be as a predicate joining objects to their events:
<object> <#hasEvent> <object/event1> .
<object <#hasEvent> <object/event2> .

It might possible for the #hasEvent to be used as both the object of an rdf:type predicate and the predicate pointing to an objects actual event. Otherwise a fedora ontology predicate could be created for the object of the rdf type.

> *I think in the PREMIS ontology the premis:Object is the container of premis:Events, and its relationship to the owl:Thing it describes is indicated by http://id.loc.gov/ontologies/premis.html#hasRelatedObject - Benjamin Armintor*

2) The PREMIS vocabulary can also be used within the event child objects by creating he child nodes and using the #hasEvent predicate to point to them. Inside the event nodes PREMIS predicates could be added, at a minimum:
http://id.loc.gov/ontologies/premis.html#hasEventDateTime
http://id.loc.gov/ontologies/premis.html#hasAgent
http://id.loc.gov/ontologies/premis.html#hasEventType
http://id.loc.gov/ontologies/premis.html#hasEventOutcomeInformation

The #hasEventDateTime, and #hasEventType should be self-expanatory. The #hasAgent would be the repository, or an external source of the events, such as a metadata generator or image transformer. The #hasEventOutcomeInformation could run the gamut from a simple literal, XML, or details expressed in a sub-graph form under on #hasEventOutcomeDetail predicate.

3) A REST endpoint would then be needed for the CRUD of events on an object, something that takes a payload of a triples to create the event properties and subnodes with PUT/POST/PATCH calls, and returns the events and properties for GET calls. These event properties could also be ingested into a triplestore allowing for selective retrieval via a SPARQL query, or a relational database and retrieved via a SQL query.

Complications:

1) The JCR events (NODE_ADDED,PROPERTY_CHANGED, etc) don't necessarily map 1:1 to all the PREMIS or other event types that one may wish to track. So the Fedora implementation of some PREMIS events may not be able to rely on the existing event machinery. In these exceptional cases, either the PREMIS may have to be coupled in the code to where the event actually takes place or a new set of Fedora events will have to be generated for the events that aren't explicitly mapped to JCR events or have the necessary information.

2) The PREMIS event types cover some but not all of the events one may wish to track. For example, a copied file maybe expressed by http://id.loc.gov/vocabulary/preservation/eventType/rep.html, a checksum maybe expessed by http://id.loc.gov/vocabulary/preservation/eventType/mes.html, but there is no eventType for a versioning. Again, the PREMIS vocabulary may have to be extended to accomodate all of the desired events.

3) If the number of PREMIS events becomes unwieldly, it might make sense to have a PREMIS container object added as a layer in between the object and its events for better incapsulation.

Resources:
http://www.loc.gov/standards/premis/ontology-announcement.html
https://wiki.duraspace.org/display/FF/Fedora+Repository+Home
https://wiki.duraspace.org/display/FF/Properties+CRUD
https://wiki.duraspace.org/display/FF/RESTful+HTTP+API
https://github.com/futures/fcrepo4/blob/master/fcrepo-audit/src/main/java/org/fcrepo/audit/LogbackAuditor.java
https://github.com/futures/fcrepo4/blob/master/fcrepo-jms/src/main/java/org/fcrepo/jms/observer/JMSTopicPublisher.java
http://id.loc.gov/vocabulary/preservation/eventType.html

A New Eventing System

The current eventing system accumulates the javax.jcr.observation.Event events in a Iterator, applies a pluggable filter to the objects in the iterator, and publishes them using the google eventbus as FedoraEvents that can then be retrieved by any method implementing the Subscribe annotation. The jcr events are as follows:
NODE_ADDED
NODE_MOVED
NODE_REMOVED
PERSIST
PROPERTY_ADDED
PROPERTY_CHANGED
PROPERTY_REMOVED

These event types are artifacts of the jcr implemention underneath fedora, and don't reflect the event types in fedora that would ideally be best to expose. To provide for the types of events we'd like to publish and preserve, once solution would be to systematically inventory the fedora services (found in fcrepo-kernel) and create and publish FedoraEvents with event types that represent these service methods. A non-exhaustive list includes:

NodeServiceImpl.findOrCreateObject
NodeServiceImpl.getObject
NodeServiceImpl.deleteObject
NodeServiceImpl.copyObject
NodeServiceImpl.moveObject
DatastreamServiceImpl.createDatastream
DatastreamServiceImpl.getDatastream
DatastreamServiceImpl.getFixity
DatastreamServiceImpl.<note -there is no removeDatastream>
LockServiceImpl.acquireLock
LockServiceImpl.getLock
LockServiceImpl.releaseLock
ObjectServiceImpl.createObject
ObjectServiceImpl.getObject
TransactionServiceImpl.beginTransaction
TransactionServiceImpl.commit
TransactionServiceImpl.rollback
VersionServiceImpl.createVersion
VersionServiceImpl.revertToVersion
VersionServiceImpl.removeVersion
FedoraResourceImpl.updatePropertiesDataset
FedoraResourceImpl.getPropertiesDataset
FedoraResourceImpl<getTriples,getHierarchyTriples,getVersionTriples,replaceProperties?>
LowLevelStorageService<transformLowLevelCacheEntriesgetLowLevelCacheEntriesFromStore?>

The PREMIS service would then subscribe to events with event types representing these methods, to create an PREMIS record reflecting these event types. The methods which generate these events also have application in the task of implementing a XACML authorization system as policies can be built with finer grained action targets that what is currently possible with modeshape actions.

PREMIS structure and retrieval

fedora structure:
-resource (object/datastream node)
  -premis (a container node)
    -event1 (hasEventType,hasEventDateTime,hasAgent,hasEventOutcomeInformation)
    -event2 (hasEventType,hasEventDateTime,hasAgent,hasEventOutcomeInformation)
    -event3 (hasEventType,hasEventDateTime,hasAgent,hasEventOutcomeInformation)

triples:
<http://localhost:8080/rest/object> <rdf:type> <http://fedora.info/definitions/v4/rest-api#premisResource> .
<http://localhost:8080/rest/object> <http://id.loc.gov/ontologies/premis.html#hasEvent> <http://localhost:8080/rest/object/premis/event1> .
<http://localhost:8080/rest/object> <http://id.loc.gov/ontologies/premis.html#hasEvent> <http://localhost:8080/rest/object/premis/event2> .
<http://localhost:8080/rest/object> <http://id.loc.gov/ontologies/premis.html#hasEvent> <http://localhost:8080/rest/object/premis/event3> .
<http://localhost:8080/rest/object/premis/event1> <http://id.loc.gov/ontologies/premis.html#hasEventType> "node added" .
<http://localhost:8080/rest/object/premis/event1> <http://id.loc.gov/ontologies/premis.html#hasEventDateTime> "2007-03-01T13:00:00Z" .
<http://localhost:8080/rest/object/premis/event1> <http://id.loc.gov/ontologies/premis.html#hasAgent> "fcrepo4 repository" .
<http://localhost:8080/rest/object/premis/event1> <http://id.loc.gov/ontologies/premis.html#hasEventOutcomeInformation> "node added info" .

retrieval:
1) use the fedora structure of object/datastreams with premis nodes and properties in a service that serializes them into json that can re returned from a restful call. This might be implemented at the resource level (each object and datastream), or an aggregation (on an object that also pulls in it's datastream PREMIS info)
2) use the triples such that sparql queries can be created to return events and their properties in a flexible way