Overview

Good progress has been made in establishing a prototype data repository for demonstration purposes on development. In the expectation that we will launch a production data repository in the not-too-distant future, these notes outline areas for architectural enhancements in order to support a production repository. The major areas of focus are in support of the compound object, specifically using Fedora relationships, and revising the compound object structure from how it was originally conceived for the prototype. Although these notes might also apply to other types of objects (e.g. ETDs with supplementary files), the focus here is on data and what needs to be put in place for an RUcore release that supports data.

Compound Object Structure

The proposed compound object structure is represented in Figure 1 below.

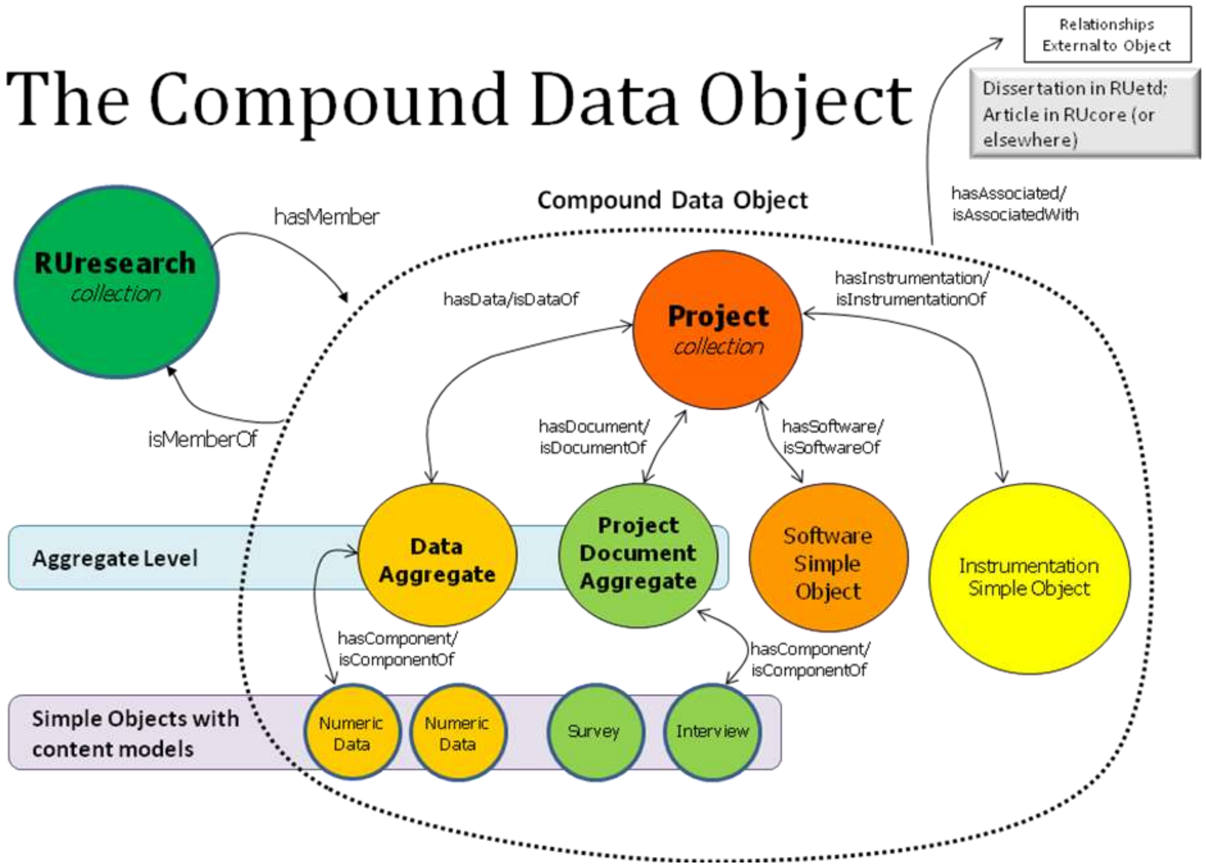
The Compound Data Object
Relationships External to Object
Dissertation in RUetd; Article in RUcore (or elsewhere)
hasAssociated/ isAssociatedWith
Compound Data Object
RUresearch collection
hasMember
isMemberOf
hasData/isDataOf
Project collection
hasInstrumentation/ isInstrumentationOf
hasDocument/ isDocumentOf
hasSoftware/ isSoftwareOf
Aggregate Level
Data Aggregate
Project Document Aggregate
Software Simple Object
Instrumentation Simple Object
hasComponent/ isComponentOf
hasComponent/ isComponentOf
Simple Objects with content models
Numeric Data
Numeric Data
Survey
Interview

Figure 1 – Compound Object for Data

It should be noted that the current prototype in development uses a two-level collection structure and the MODS relatedItem field to model compound objects. This approach is recognized as being overly complex and not providing the flexibility that is needed for data. Our knowledge and capabilities regarding the use of relationships has grown significantly in the past few months, given the work with annotations and much informal discussion.

In preparing to move the prototype to production, an alternate approach for modeling compound objects is suggested as illustrated in Figure 1. In contrast to the existing prototype on devleopment, the compound object structure can be greatly simplified by having only one collection object (the view object) as part of the compound object. The second collection level – the aggregate level – that provided a grouping mechanism for data, documents, etc. is not needed and overly complicates the structure. The four main components of the compound object – data, project documents, software and instrumentation – can be modeled using Fedora relationships and will be used in the portal to display the tabs for navigation within the compound object. In addition, the compound object can have many additional ontological relationships and need not be strictly hierarchical. For example, one might create a relationship between a data and software object to enable a user to navigate directly from the data to the software. For initial implementation on production, the recommendation is to use the relationships that have been established in the prototype, i.e. data, software, documents, and instrumentation. These relationships are illustrated in Figure 1.

Object Aggregation

It should be noted that there can be many simple objects of a specific type in the compound object. For example, Figure 1 indicates that there are two data objects and two document objects. It is possible that there might be many data objects (e.g. >50) in a single compound object. An example could be state census data. A user might want to download all state census data or select individual state census data.

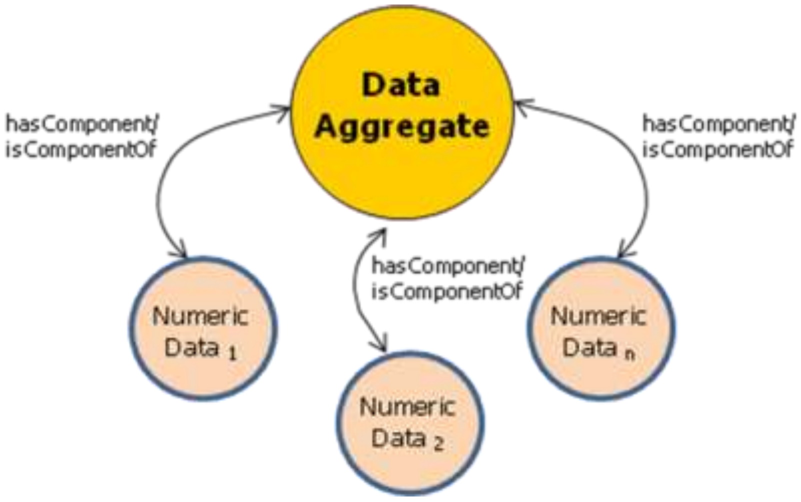
Data Aggregate
hasComponent/ isComponentOf
hasComponent/ isComponentOf
hasComponent/ isComponentOf
Numeric Data 1
Numeric Data 2
Numeric Data n

Figure2– Data Object Aggregation Example

As we get more experience with different types of data, we may want to introduce additional relationships to manage this complexity. For example, there might be a single object with the hasData relationship and which encapsulates the many component data objects in a tar or zip file. The user could download this single object and uncompress it to get access to all the data objects. If it is likely that the user will want to view and select a few objects or may not want to download a single, very large aggregation, then we might consider introducing another level under the data object with a different relationship (e.g. hasComponent) which would allow the user to download only a few of the data objects.

Questions to consider are:

- What should be delivered in the aggregation and how is it physically structured?
- How and when is the aggregation created?
- What file format and file standard should the aggregation be?

Possible solutions:

- Aggregation should include presentation datastreams of related objects and exported metadata about the individual objects and the data aggregate. Also, a manifest file about the aggregate to serve as a "map" of the aggregation.
- Aggregation creation options include:
  - Creating the aggregation and storing it as a datastream with the aggregation object. If any of the related components are updated, then the aggregation object datastream needs updating. Process can be manual or automatic.
  - Creating the aggregation at time of request
- Format possibilities include ZIP and TAR. Others should be considered along with the standards version. The data curator can provide further guidance on this issue.

There are obvious pros and cons of this approach and the issues relate to reducing complexity for both the user and the project manager. For R5.2 the immediate need for this functionality is low. During 5.2.1 development it is recommended that a data aggregation specification be written. In the interim, between R5.2 and R5.2.1, the proposed solution is to provide a request mechanism to end users who wish to have data aggregations. The request will be manually filled at the time of request.

Collection Structure

In Figure 1, a single collection object is represented – the project collection. Given the existing RUcore architecture, we will need one data repository collection object (at least) for management and support of functions like statistics reports. This collection object has a relationship to view objects (i.e. *hasMember/isMemberOf)* and the MODS relatedItem field is also used in the view object to show collection membership. For indexing purposes, each of the simple resource objects are members of their respective view objects (i.e. each will have a MODS relatedItem indicating membership in the view object collection). As the data repository matures, we may want to consider introducing subject specific data collections (e.g. biology, chemistry, etc).

Ontologies for the Compound Object

Each simple object is related to the view object by one of the relationships: *hasData/isDataOf, hasDocument/isDocumentOf, hasSoftware/isSoftwareOf* and *hasInstrumentation/isInstrumentationOf.* Definitions of these relationships will have to be developed in more detail. In addition, we may want to develop other network relationships between simple objects of different types.

For a future release, we should consider how we want to link to related objects outside of RUcore. Relationship to "external objects" can be might using the *hasAssociated/isAssociatedWith* ontologies. Fedora RELS supports these "external" non-literal relationships and this has been built into the Relationship API for R5.2.

Implementation in R5.2

The objective of this section is to briefly describe what we will need to do in R5.2 to provide basic support for data.

Metadata Descriptive events might be present that correspond to a particular relationship expressed in RELS. It must be noted that this loose coupling between these metadata events and RELS statement can exist and need to be addressed in the user interface. All non-collection resources will need to be cataloged using specific mods:classification terms to aid in discovery. The following list of controlled vocabularies is proposed; data, document, instrumentation, and software. These mods:classification entries will use the term authority="RUresearch" and have no edition information.

API for Management of Relationships An API is under development that can be used by the analytic, dlr/EDIT and (in a future release) by WMS. This API will provide the capability to add and delete relationships based on characteristics that are expressed in a related table. Onotologies are registered with the API before they can be created. When registering ontologies, characteristics about the ontology are defined and these characteristics are used when indexing, displaying or purging objects. More about the Relationship API can be found by reading the following specification: http://rucore.libraries.rutgers.edu/collab/ref/spc_sawg_r5_2_rels.pdf

Indexing and dlr/EDIT The results from a search will only show the view object in the results list, however metadata in all of the simple objects in the compound object should be indexed (similar to what is being done for the analytic). As a post-R5.2 addition, we should consider highlighting the objects where the search hit actually occurred. As was done for the analytic, we will need to create a content model for the view object. Although the view object is a collection, the current data proposal calls for a zip datastream to be an optional, single "presentation" datastream for the view object. This zip file encapsulates all of the simple objects and allows the user to easily download the complete compound object. In order to standardize relationship processing, dlr/EDIT (and WMS) should use the API under development.

Workflow Management For R5.2, there should be no need to make any changes to WMS, i.e. the collection object and resource objects in Figure 1 can be ingested with the current release of WMS. For the relatively few compound objects that will be ingested, dlr/EDIT will be used to create and manage the relationships.

The Data Portal The data portal must "understand" the basic compound object architecture as shown in Figure 1. Changes would have to be made in the prototype data portal to support navigation via tabs to the various sections of the compound object (i.e. data, documents, etc). As we move to the three major thematic portals, it is assumed that the data collection in its entirety can be assigned to the research portal.

Citing Objects (Showfed) All of the simple objects in the compound object will be assigned a persistent ID. As a result, any of the objects could be cited (e.g. in a journal article). So, for example, someone who used a specific dataset and the related software may decide to cite these two objects. What is important in this scenario is that the reader of article can use the PID to go directly to the dataset. However, this person will likely want to navigate to other parts of the object. Architecturally, our PID displays will need to find the related links and display the tabs as discussed in the results display. There are pending requests to revise showfed to provide the same type of results display as is done in the main search interface. Investigation is underway to determine whether we can make these changes in R5.2.