TheDevelopers

Developers can contribute to the source code of the toolkit

Updated Yesterday (22 hours ago) by amg...@gmail.com

QuDEx Repository Tools

QuDEx Repository Tools are deployed in an application server, e.g. Apache Tomcat, and rely on a number of existing open-source tools that are in common use in data archives and institutional repositories. These include digital repositories and RDF databases; metadata standards and RDF vocabularies; QuDEx schema, published by the UK Data Archive for describing CAQDAS metadata and qualitative data collections; among others. These tools are entirely developed in Java language and will be released under the Apache 2.0 license. These tools rely on and communicate with two different external systems that need to be installed and configured before installing these tools. They are described as follows:

- Fedora Digital Repository. Across a wide range of disciplines, a key enabler of semantic web adoption more generally is the provision of flexible, configurable digital repositories which are `semantic web ready' by offering interfaces such as OAI-PMH or providing mechanisms for their integration with semantic databases such as triplestores. Following analysis of the functionalities of several repositories, Fedora was selected as the digital repository underlying the developed self-archiving toolkit. Fedora offers many useful features such as a digital object model that allows to store, within an object, metadata annotations using different schemas, data in differing formats and even semantic-ready data in the form of internal RDF/XML data. Additionally, it is closely-coupled with Mulgara RDF database. This approach enhances the management of semantic-ready metadata and data stored in Fedora by aggregating them in the Mulgara triplestore instance.
- Mulgara Semantic Triplestore. This RDF database implements many of the World Wide Web Consortium's (W3C) Semantic Web concepts and it is a database designed to hold metadata in the form of subject-object-predicate statements. Its integration with Fedora repository facilitates access to data and metadata from the digital repository, which can then be combined with data from other sources, query interfaces used to populate web interfaces or lightweight visualisation frameworks like 'Exhibit' from the MIT Simile toolkit. It is possible to pass complex queries to Mulgara using a number of query languages such as SPARQL - query language similar to the ones used to query relational databases - or iTQL - Mulgara's proprietary query language. The results of these can then be used to drive web applications, be presented using the SIMILE Exhibit framework, or be exposed as web services since it provides a REST-like web interface.

qudex-repository-tools

Self-archiving toolkit for the Fedora Repository

SourceCode

Developers: access the source code

Updated Yesterday (18 hours ago) by qudexrep...@gmail.com

The source code of the Toolkit is hosted in a subversion server. Use this command to anonymously check out the latest project source code:

```
# Non-members may check out a read-only working copy anonymously over HTTP.
svn checkout http://qudex-repository-tools.googlecode.com/svn/trunk/ qudex-repository-tools-read-only
```

DevEclipseSetup

Developers: importing the toolkit source code into Eclipse IDE

Updated Today (16 hours ago) by qudexrep...@gmail.com

QuDEx Tools v0.1

The following instructions refer to the set up of Eclipse IDE to import the source code of the applications included in this toolkit so that they can be set up as Java projects within the Eclipse environment.

Prerequisites

The following details of the plugins that need to be installed in Eclipse IDE work for Eclipse Helios (3.6) and Eclipse Indigo (3.7). Most of the Update Sites URLs will work for more recents versions of Eclipse, although some of them are now available through Eclipse Marketplace rather than via update sites.

- **VeloEdit** (optional). This is a plugin that provides an editor for working with velocity web templates.
  - Update Site: http://veloedit.sourceforge.net/updates/
  - Version 1.3.1-final
- Install **SVN** + **Subclipse** (recommended). Subversion plugins to enable working with version control systems such as Subversion.
  - From the Eclipse Market Place. Search for Subclipse.
  - Install Maven plugin
    - Installation through MarketPlace, or
    - Installation from Install Software
      - Update Site: http://m2eclipse.sonatype.org/sites/m2e
    - Maven extra for WTP tools for eclipse
      - Update Site: http://m2eclipse.sonatype.org/sites/m2e-extras
- Install **Spring** Plugins (required). The applications included in the toolkit use the Spring Framework.
  - Installation from the MarketPlace. Search for spring-ide
  - Update Site (AJDT needed first): http://download.eclipse.org/tools/ajdt/35/update/
  - Spring Update site: http://dist.springframework.org/release/IDE
- **Veloeclipse** (optional)
  - Update site: http://veloeclipse.googlecode.com/svn/trunk/update/

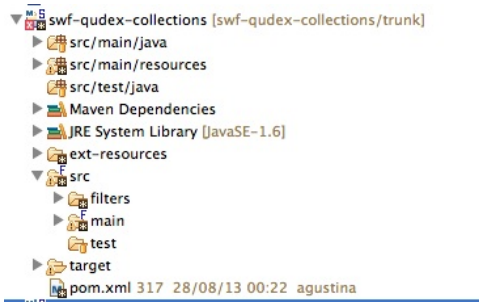
Collections Manager - Eclipse/Maven Project Description

The project has been created using Eclipse IDE (tested using Eclipse 3.5 and 3.6. It should work in previous versions. Eclipse IDE available from: http://www.eclipse.org/) and Maven. Its type is Dynamic Web Application with Spring nature (using Spring Tools plug-in) and Maven management (using the M2Eclipse plug-in).

Project Folder Structure

The code of the project is structured as follows:

- src/main/java. Java code implementations of the classes that implement the web application's functionalities.
- src/main/resources. Contains all the resource files needed by the web application, e.g. configuration files, installer files, etc.
- src/filters. Contains the application properties file (filter-prod.properties).
- src/main/webapp. Directory of the web application files: HTML files, javascript code and Spring configuration files.



Caption: Project structure in Eclipse IDE

Template Builder - Eclipse/Maven Project Description

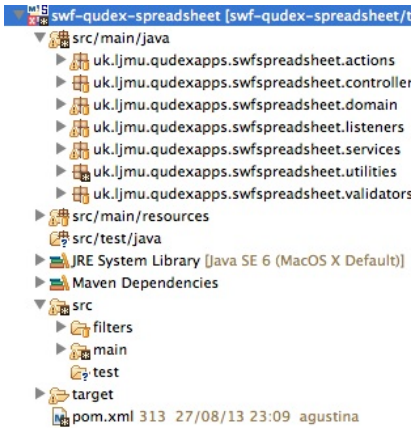
The project has been created using Eclipse IDE (tested using Eclipse 3.5 and 3.6. It should work in previous versions. Eclipse IDE available from: http://www.eclipse.org/) and Maven. Its type is Dynamic Web Application with Spring nature (using Spring Tools plug-in) and Maven management (using the M2Eclipse plug-in).

Project Folder Structure

The code of the project is structured as follows:

- src/main/java. Java code implementations of the classes that implement the web application's functionalities.
- src/main/resources. Contains all the resource files needed by the web application, e.g. configuration files, installer files, etc.

- src/filters. Contains the application properties file (filter-prod.properties).
- src/main/webapp. Directory of the web application files: HTML files, javascript code and Spring configuration files.



Caption: Project structure in Eclipse IDE

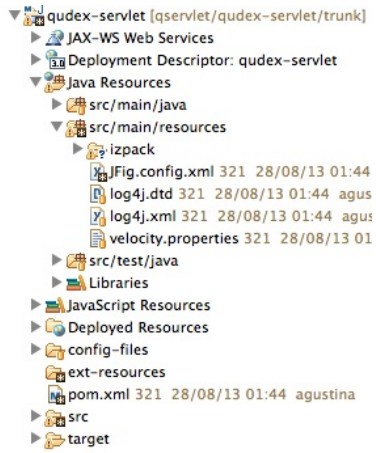
Collections Explorer - Eclipse/Maven Project Description

The project has been created using Eclipse IDE (tested using Eclipse 3.5 and 3.6. It should work in previous versions. Eclipse IDE available from: http://www.eclipse.org/) and Maven. Its type is Dynamic Web Application with Spring nature (using Spring Tools plug-in) and Maven management (using the M2Eclipse plug-in).

Project Folder Structure

The code of the project is structured as follows:

- src/main/java. Java code implementations of the classes that implement the web application's functionalities.
- src/main/resources. Contains all the resource files needed by the web application, e.g. configuration files, installer files, etc.
- src/filters. Contains the application properties file (filter-prod.properties).
- src/main/webapp. Directory of the web application files: HTML files, javascript code and Spring configuration files.



Caption: Project structure in Eclipse IDE

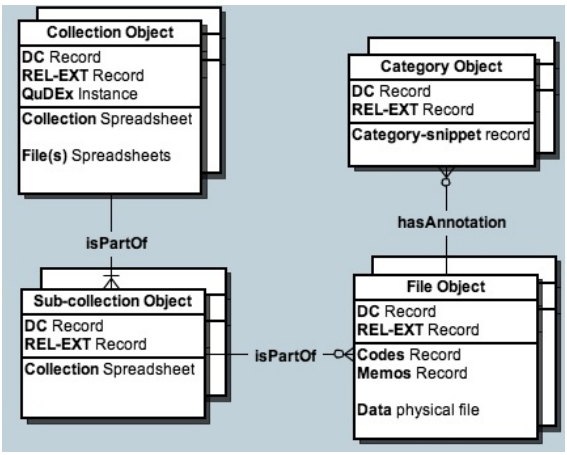
TheDataModel

Toolkit's data model

Updated May 31, 2013 by amg...@gmail.com

Toolkit's Data Model

The data model that underpins the self-archiving toolkit includes support, firstly for managing basic analytical units such as categories and codes, that can be attached to any data file included in the collection, and secondly it enables to express how the different elements within a collection relate to each other by using a set of common relationships, while at the same time, maintaining descriptive documentation at the study level, similarly to data archives. This model is based on a combination of the Fedora object model to represent the types of objects within a collection, and the QuDEx qualitative schema, to represent the relationships between the different materials within a qualitative collection. The model includes three basic types of objects, which are generic enough to represent, on the one hand, the structure of the collection and on the other hand, the types of qualitative materials that can be included.



The digital repository object model is designed based on a "compound digital object" that enables to aggregate one or more content items into the same digital object. These items can be of any format and can either be stored locally in the repository, or stored externally and then referenced by the digital object. This design principle has facilitated the design of the proposed data model, which required to incorporate a combination of data documentation content items, using a variety of metadata standards, along with their associated qualitative materials. The data model is composed by three different types of digital objects, which are linked to each other by using semantic relationships that are defined in an ontology expressed in RDF format: Fedora relationships ontology. While the relationships expressed in this ontology are used internally as a mechanism for linking the digital objects, they are generic enough and expressed in a standard way - conforming to linked data standards - so that it present the advantage of enabling re-usability and dissemination of the digital objects, and their relationships.

The first of these objects, 'collection object', is the core object of the model and it represents the qualitative collection as a whole (the study). Following the principles of the fedora compound digital object, it incorporates the following content items:

- Dublin Core record. This is a documentation content item, in other words, it contains a metadata record describing a qualitative material. This metadata record contains those terms from the Dublin Core standard that are of relevance to describe the study represented in a particular collection. The contents of this record are indexed automatically, in the form of triples, into the RDF database (Mulgara) so that they are accessible to the collection visualisation tools.
- REL-EXT record. This is a special documentation record that contains the relationships information, that is, it expresses all the objects with which this collection is related, and the type of relationship. It also holds the metadata are associated with the collection, which are included in metadata standards other than Dublin Core. The contents of this record are also indexed into the RDF database.
- QuDEx instance record. This content item is particularly important since it contains a representation of the whole collection, including all the files, analytical elements and relationships, expressed in a QuDEx XML instance. This is highly important in terms of data sharing and re-usability, since the description of the files included in the collection, along with their relationships and attached analytical elements, could be imported into other applications that support the QuDEx schema.
- Collection spreadsheet content item. This is a content item that contains the original spreadsheet used to generate a collection in the digital repository. Storing the original spreadsheets used to generate is useful in terms of providing a backup mechanism for the sets of spreadsheets used to create a collection in the repository.
- A number of File spreadsheet content items. Similarly to the collection spreadsheet content item, the collection object contains a list including all the file spreadsheets used to aggregate the qualitative materials to the collection.

The second of the objects included in the model is the 'sub-collection object'. This object is quite similar to the collection object in that it holds documentation/relationship records and a collection spreadsheet content item. However, it is used with collection organisation purposes so that the qualitative materials can be grouped by different criteria and then be attached to the sub-collection object. The third type of object is the file object, which is used to represent and store any type of qualitative material. It includes similar documentation records to the ones for a collection or sub-collection object, although the metadata terms used to describe this type of object differ (for more detailed information about the metadata vocabularies used to describe collections and files see the table below). Additionally, it can have analytical documentation attached to it. This information is included in special content item records, codes and memos, which are represented as XML snippets expressing what codes are associated with a particular file object, and more importantly, a list of all the files that are also associated with the given code. These codes and memos are analytical annotations of a given file, similar to the ones used within CAQDAS packages. Lastly, the fourth type of object is the 'category object', which is an analytical element that can be attached to files objects within a collection. The important records associated with this type of object are the RELS-EXT documentation record, and the `category-snippet' content item. The first of these includes important relational information which describes all the files within the collection to which a given category object is attached and it also describes the source(s) for a given category. Analytical categories, like the ones used in CAQDAS packages, are normally hierarchical, therefore relationships defining the provenance are needed. The second, is a content item record that holds the information about a given category object as a QuDEx XML snippet.

Type	Description	Schema	Metadata Terms
Collection	Two different types of documentation are provided for collection and sub-collection objects, which use a combination of metadata terms from multiple schemas. The first type of documentation is related with the methodology of the study, and uses a reduced set from the DDI schema (version 2.1); geographical information, from WGS84 lat/long vocabulary. The second type of documentation is general description of the collection by using mostly Dublin Core terms.	DC	description; language; coverage; rights; contributor; type; creator; identifier; subject; date; publisher and title.
		QuDEx	updatePid and deletePid (used for deleting and updating collections.)
		DDI	samplingProcedure; universel; timeMethod; dataCollector and collectionMode
		RDF	type (collection or file)
		GEO	latLong and location
File	Two different types of documentation are provided for file objects. The first one is a more general description of the qualitative material physical file, including information such as file format, date of creation, author and title, etc; whereas the second type is analytical information attached to the file such as memos or annotations; codes within the file or categories.	DC	description; language; subject; format; title; coverage; type; creator and source
		QuDEx	labelCode; labelCategory; category; labelMemo; memo; isOriginal and code
		RDF	type (collection or file)
		GEO	latLong and location

7

UMLDiagrams

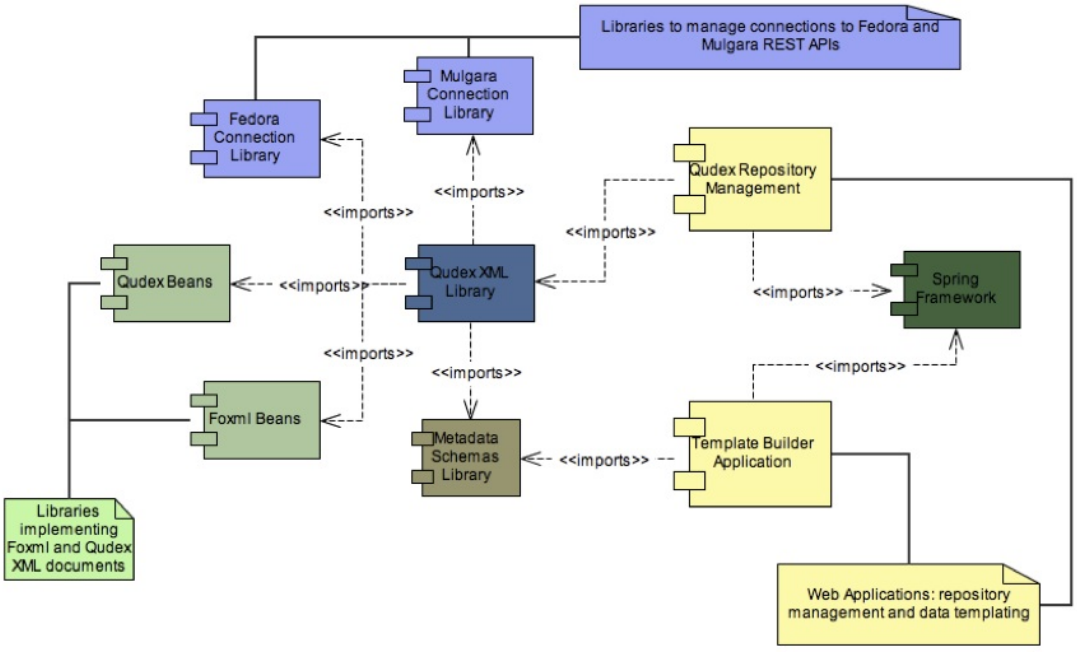
Toolkit applications UML Process diagrams

Updated Yesterday (23 hours ago) by amg...@gmail.com

Introduction

The QuDEx Repository Toolkit is entirely developed in Java and can be installed and executed in multiple platforms. The toolkit has been tested primarily in Mac OS X (10.6.X and higher) and Windows (XP and 7). They should also work in Unix and Linux based systems. \newline

The toolkit is composed of a number of applications that implement the core functionalities, including the repository management logic and web interfaces. These applications rely on multiple independent Java libraries implementing separate groups of functionalities. The first set of libraries implements the functionalities related to connection and communication management with the digital repository (Fedora) and the RDF database (Mulgara). The second set implements all the classes required to manage both QuDEx and FOXML (more information on Fedora's FOXML available here) documents. Lastly, the third set of libraries implements additional metadata schemas management (DDI2, Dublin Core schemas, among others). The following Figure shows the groups of libraries (colour-coded) which comprise the toolkit.



► Sign in to add a comment

activityTemplateBuilder

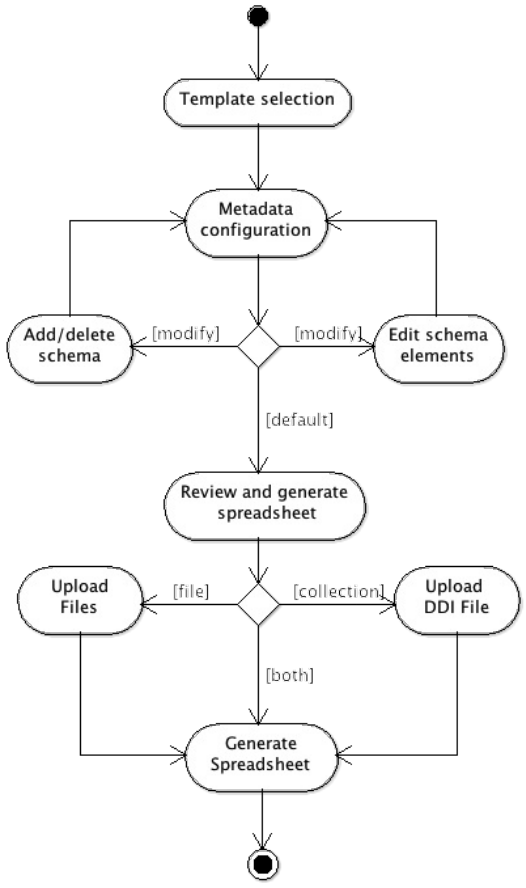
Activity diagram Template creation process

Updated Jun 3, 2013 by amg...@gmail.com

Template creation process

The template builder application uses Spring Webflow to design and implement all the interaction flows, i.e., user web interfaces are designed by using flows depending on the activities being performed. The application comprises the following flows (from Spring webflow perspective):

- Metadata Flow. This flow represents the process of choosing the different metadata schemas that will be available to include in the spreadsheet template.
  - Metadata-elements Flow. This is a sub-flow of Metadata-flow and it is used to select, add or delete elements from a particular schema and included in the template spreadsheet.
- Upload Flow. This flow manages the physical files uploads.
- Generate Flow. This flow manages the process of generating the spreadsheet templates and including any physical attached files to the spreadsheet template.



(This activity diagram represents the process of creating a template for either a collection spreadsheet or a files spreadsheet.)

Metadata Flow

This is the initial flow of the web application. It is composed by different view states, depending on the stage within the process and it calls an additional sub-flow (Metadata-elements) to perform operations in the metadata elements. Its end state calls a new flow (Generate Flow) to continue with the template generation process. It presents the following views:

- Select type of spreadsheet. Initially the user has to select the type of template there are going to build: collection/sub-collection template or files template
- Select metadata schemas. Depending on the previous selection different metadata schemas are available. From this view, one can select any of the schemas in case the elements present need to be modified.
- Add Metadata elements. This view displays all the elements available for a given metadata schema. The user can select/deselect any of the elements.

Generate Spreadsheet Flow

This flow describes the final process of generating a spreadsheet template. It is reused in either of the templates scenarios (file and collection templates). Depending on the type of spreadsheet template only certain views/states are present although there are shared views between the two scenarios. The views that are dependent on the type of spreadsheet being generated are \enquote{File upload} and \enquote{Upload DDI XML}

and they are represented in the diagram below by the fork element (black bifurcation).

Upload Flow

This flow represents the process of uploading ZIP files associated with a spreadsheet template. This flow is only present if the user is creating a file template since this type is the only type that accepts descriptions of physical files associated with each file being described in the template. It represents a sequential process of uploading ZIP files one at a time.

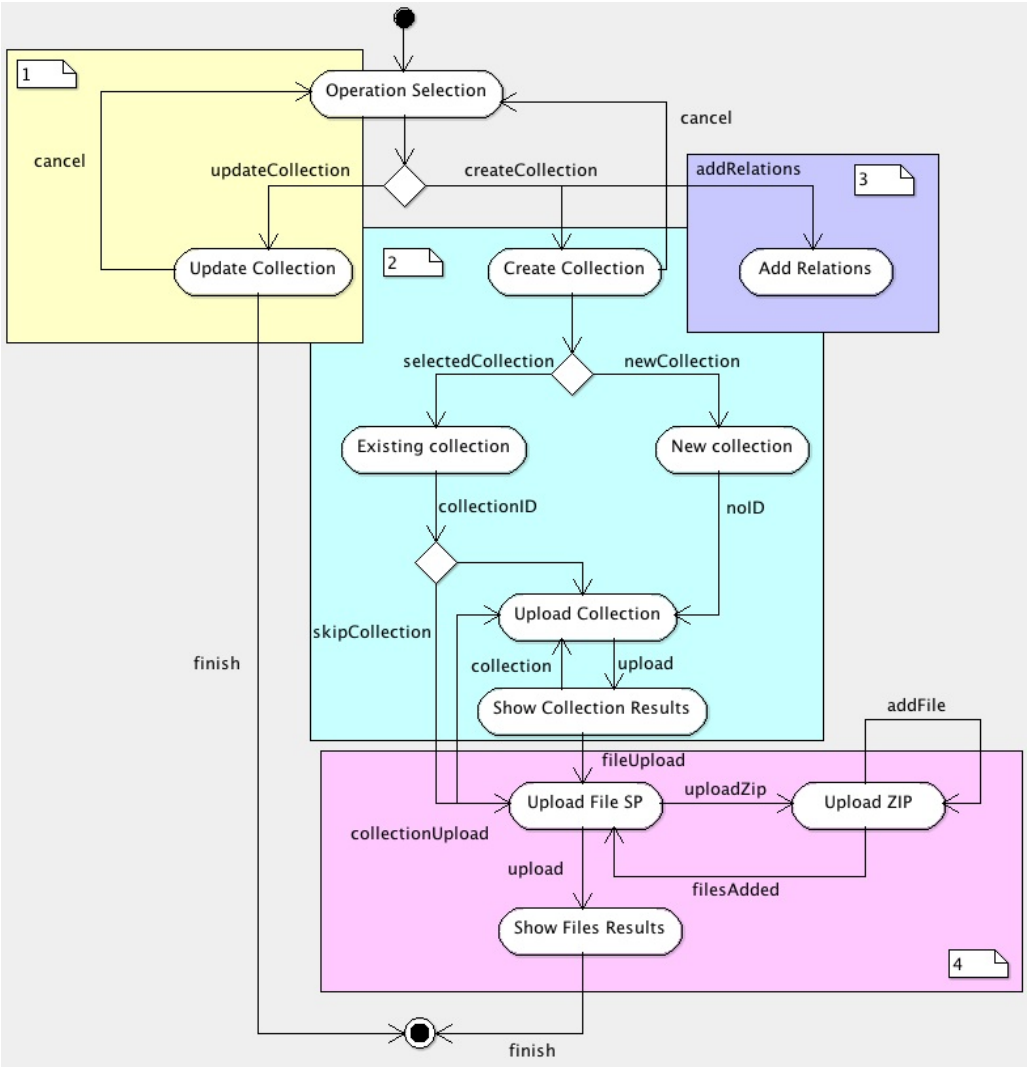
activityColManager

UML general activity diagram for the Collection manager tool.

Updated Jun 3, 2013 by amg...@gmail.com

Collection Manager Activity diagram

The Collection manager application uses Spring Webflow framework. This application implements a main flow that manages the process of creating/updating collections into the Fedora repository, as well as adding relationships between files within a collection. As part of the collection creation, the user can upload compressed files that include the resources associated with a collection being created. This operation is performed by an additional sub-flow. All the scenarios are managed by one flow due to the transactional nature of the application. The application presents a linear process for collection creation to avoid the user performing operations in the wrong order. Therefore, only one main flow is implemented which presents with sequences of user views and operations to perform. For simplicity, the flow is described in terms of groups of operations. Each group presents a set of view-states and action-states that enable to perform the different application activities.



Collection update (section 1 in the diagram)

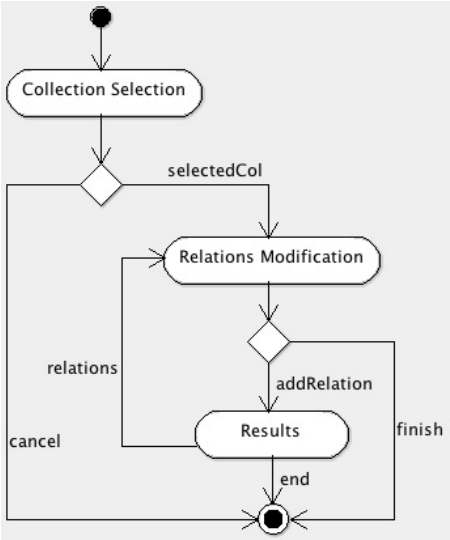
This group of operations enable to perform metadata modifications over existing collection objects as well as deleting collections from the repository. This part of the flow presents a single view that enables the user to upload update collection spreadsheets. This type of spreadsheet is a collection spreadsheet, which presents specific qudex fields to perform update/delete operations over existing collections/sub-collections (see section 1 in the activity diagram below). Once all the update operations have been performed then the process (and the flow) terminates.\newline

Collection creation/modification (section 2 in the diagram)

This group of operations within the flow enables either to perform a QuDEx collection creation (uploads of collections/sub-collections and associated files or documents) or to continue with a previous collection creation process (this refers only to collections previously stored that do not present an associated QuDEx instance document). In the latter case, the user could only associate files to the collection that he wants to update or could add new sub-collections associated with the selected one and after that include files in the collection (see section 2 in the diagram below).\newline

Adding relationships to existing collection sub-flow (section 3 in the diagram)

This sub-flow represents the process of adding relationships between files that belong to an existing collection. Once the user enters the flow the next step is to select the collection to which the new relationships are going to be added. The next step would be to select the resource that is the source of the relationship, then the type of relationship is selected and finally the target resource is selected. This process can be repeated as many times as relations to add. Once the user finishes the process, then the application returns to the initial state for selecting the different operations.



Files upload (section 4 in the diagram)

This is part of the collection creation/modification processes. Once the collection structure has been created, one can associate files or documents objects to it. This group of operations enable to upload both files spreadsheets templates and compressed files with the associated data resources (see section 4 of the diagram below). The upload of compressed files is optional and can be performed more that once. To perform the latter, the main flow calls the Upload sub-flow that manages the upload and operations over compressed resources files.

Upload Sub-flow

This flow represents the process of uploading ZIP files associated with a spreadsheet template. This flow is only presented if the user is uploading a files spreadsheet since this type is the only type that accepts physical files associated with each file being described in the template. It represents a sequential process of uploading compressed files, one at a time.