Research Data Registry

Developer Documentation

This document details the work completed by JK Software for The University of Melbourne, Information Technology Services (ITS) division on the Research Data Registry (RDR) project. Specifically, it outlines the customisations that were performed to the underlying system VIVO in order to fulfill the specific requirements of the client.

Table of Contents

Stakeholders

Entity	Role	Individual Contact Details
Information Technology Services - The University of Melbourne	Client	**Simon Porter (Lead Developer)** email: simon.porter@unimelb.edu.au **Terry Brennan (Project Manager)** email: tbrennan@unimelb.edu.au **Michael Bertie (User Acceptance)** email: mbertie@unimelb.edu.au **Sara Ogston (Assistant Project Manager)** email: sogston@unimelb.edu.au
JK Software	Consultant Developer	**Judd Kirby (Project Manager - External)** email: j.kirby@jksoftware.com.au **Tom Sullivan (Lead Developer)** email: tom@msbit.com.au

Versioning

Number	Changelog
0.1 (Draft)	● Creation of document ● Addition of Table of Contents section ● Addition of Stakeholder section ● Addition of Versioning section ● Addition of Introduction section ● Addition of Requirements section and subsections ● Addition of Design & Implementation section and subsections ● Addition of Acronyms section
0.2 (Draft)	● Added description of autocomplete functionality ● Added further details around the structure and configuration of the *ands, vivo* and *unimelb-rdr* ontologies
0.3 (Draft)	● Updated Java class documentation ● Updated Freemarker template documentation

Introduction

The RDR is a pilot project initiated and managed by the ITS division of UOM. Currently there is no provision in the university for a central repository to house information related to research projects and the data generated from these. The primary goal of this project is to allow the creation and management of this information and to relate it to existing university resources such as:

- Researchers, which may be referred to as a Person
- Contracts and Grants, which are collectively known as Agreements
- Information Resources, which may also be referred to as Publications or Academic Articles

These existing resources are provided through integration with the Find an Expert system, a separate university project.

Once the information is housed within the system, the subsequent goal is to make it locatable and accessible by university personnel.

Requirements

Infrastructure

The system is built upon VIVO, an open source web based technology originally created by Cornell University. Further information can be obtained from the VIVO website. Specific details on the implementation of the infrastructure is available in the *Design & Implementation -> System Architecture* section.

User Interface Appearance

The web based user interface will be visually represented using the university theme as developed by the Marketing and Communications division outlined on their Web Template Guidelines page.

Resources

In order to support the housing and retrieval of the research related information, the following resources are required within the system, and must be creatable and modifiable by the user:

- **Project**
  Identifies a research project conducted within the university

- **Research Data**
  Identifies research data generated within the university, usually attributed to a project.

The specific details of the user interface to allow the creation and modification of these resources is available in the *Design & Implementation -> Forms* section.

Relations

In order to provide context for the resources described above, the system must allow the user to link them to existing resources within the university. This is achieved through integration with the Find an Expert system, which is detailed in the next section. The following relations must be allowable in the system:

- Project to an Agreement
- Research Data to a Researcher
- Research Data to a Project
- Research Data to an Agreement
- Research Data to an Information Resource

Each of these relations should allow for the created resource - either a Project or Research Data - to retrieve the following information from the linked resource where it is relevant:

- Subject Areas
- Custodians
- Custodian Departments

The specific details of the user interface to allow the linking of these resources and their relevant data is available in the *Design & Implementation -> Forms* section.

Integration

In order to support the linking of created resources to existing resources, integration is required with the Find an Expert system. This integration involves the retrieval of all resources in Find an Expert and population of that data in the RDR system. The resources can at that point be located within the RDR system and linked to internally created resources. The implementation of how this integration is achieved is outlined in the *Design & Implementation -> Integration* section.

Design & Implementation

System Architecture

Figure 1. displays the visual representation of the system architecture. As can be seen from the diagram the RDR functional customisations detailed in this document reside within a VIVO system. This is deployed in a Tomcat servlet container running on a JVM. All of these applications are housed within an Ubuntu Server OS, a Linux based distribution. The specific versions of each application are:

- VIVO - 1.4
- Tomcat - 6.0.35
- JVM - Java 1.7.0
- Ubuntu Server - 10.04.03 LTS

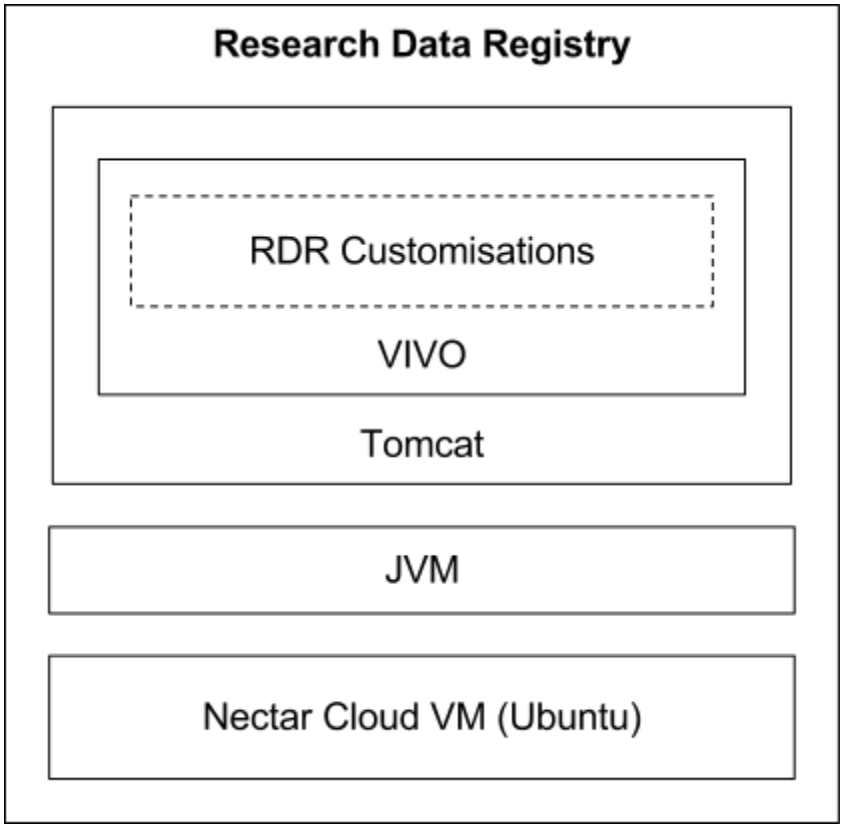

Figure 1.

Integration

Integration with the Find an Expert system consists of two components, an initial and one-time import of all data within the Find an Expert system and then subsequent periodic imports of modified data in the form of deltas. The initial import will be completed before the system goes live. The periodic updates are performed by a script that is executed by a cron task on the system that occurs daily at a low usage time (I.e. 3:00 AM), which performs the following tasks:

1. Fetches the entire list of ADD delta files from Find an Expert
2. Ascertains from it's internal storage which of the files have previously been executed and therefore don't need to be imported
3. Fetches the files to be imported and loads them into the database using the command line tool *vivo-db-tool* that has been provided by UOM
4. Executes a selective recompute of inferences using the command line tool *unimelbAdditionalTriples* that has been provided by UOM

NB: Find an Expert provides both ADD and DEL files which perform the addition and deletion of data respectively. It was decided by UOM that it's unnecessary at this point to use the DEL files, hence they are ignored by the integration scripts.

Forms

Creation of a custom form involves the interaction of at least two items, a Java generator class, a Freemarker template file and optionally a Java preprocessor class. The generator performs SPARQL queries, sets up N3 assertions, indicates which Freemarker template file to use and provides variables and labels for the form output.

The Freemarker template file provides the layout for the resultant form, and also has scripting logic to allow more complicated parsing of the provided variables and labels.

On submission, the optional preprocessor class can be used to perform further required logic.

Generators

The created generators are contained within the following package:

```
edu.cornell.mannlib.vitro.webapp.edit.n3editing.configuration.generators
```

Extending the existing generator class `VivoBaseGenerator`, the abstract class `RdrVivoBaseGenerator` provides an implementation of the required method:

```
EditConfigurationVTwo getEditConfiguration( VitroRequest vreq, HttpSession session);
```

The implementation of this method calls a number of custom methods to populate the

`EditConfigurationVTwo` instance and this is where the subclasses can tailor generation of the form as required. This class requires subclasses to implement the following methods:

Method	Description
`List<String> getN3Optional();`	Return a listing of optional N3 statements that can be matched to create additional assertions on creation of an object.
`List<String> getUrisOnForm();`	Return a listing of all the URIs that are returned on submission of the generated form. These can be referenced within the provided optional N3, and need a counterpart field generated in `getFields()`.
`List<String> getLiteralsOnForm();`	Return a listing of literal strings that are returned on submission of the generated form. These can be referenced within the provided optional N3, and need a counterpart field generated in `getFields()`.
`List<N3ValidatorVTwo> getValidators();`	Returns a list of `N3ValidatorTwo` instances, which represent form level validation rules.
`HashMap<String, Object> getFormSpecificData(VitroRequest vreq);`	Return a map of name to arbitrary objects that can be referenced in the Freemarker template which provides the layout for the theme.
`Log getLog();`	Return the specific `Log` instance for the implementing class.
`String getTemplate();`	Return the name of the Freemarker template file.
`String getForwardUri();`	Return the URI of the object to redirect to on form submission. Generally this is the N3 variable for the object of the newly created relationship, as specified in `getN3Required()` and `getObjectName()`.
`List<String> getN3Required();`	Return a listing of the N3 statements that must be asserted to create the object and/or relationship.
`Map<String, String> getNewResources(VitroRequest vreq);`	Return a map of resource URI names to namespace definition for the new object. Generally this is set to the static variable `DEFAULT_NS_TOKEN` to force the use of the default namespace.
`String getSubjectName();`	Return the variable name of the subject in the

	relationship, to be used in N3 statements.
`String getPredicateName();`	Return the variable name of the predicate in the relationship, to be used in N3 statements.
`String getObjectName();`	Return the variable name of the object in the relationship, to be used in N3 statements.
`List<FieldVTwo> getFields();`	Return a listing of the `FieldVTwo` objects representing fields that will exist on the form, or will be provided as inputs used at submission time. For the purposes of this project, a convenience class `CustomFieldVTwo` has been created in the same package to allow easy creation of fields via an overloaded constructor.

Directly implementing this class are the following abstract subclasses:

1. `AddProjectToThingGenerator`
2. `AddResearchDataToThingGenerator`
3. `AddPublicationToThingGenerator`
4. `AddOrCreateGenerator`

The first 3 of these classes respectively provide the basic functionality for relationships where the object is of type:

1. http://vivoweb.org/ontology/core#Project
2. http://vivoweb.org/ontology/core#InformationResource
3. http://purl.org/ands/ontologies/vivo/ResearchData

The last provides a generic way to create custom forms which allow adding an existing item via an autocomplete field or forwarding off to a different custom form to allow addition of a new item.

Where appropriate, these classes infer possible relationships to other existing items based on existing relationships between the subject and these existing items. All these classes provide implementation of the following required methods:

- `List<String> getN3Optional();`
- `HashMap<String, Object> getFormSpecificData(VitroRequest vreq);`
- `List<String> getLiteralsOnForm();`
- `List<String> getUrisOnForm();`
- `List<FieldVTwo> getFields();`
- `Map<String, String> getNewResources(VitroRequest vreq);`
- `List<N3ValidatorVTwo> getValidators();`
- `String getForwardUri();`

These classes may require subclasses to implement some methods, these are outlined as below.

AddProjectToThingGenerator

Method	Description
`Map<String, String> getInheritedRolesLabelAndUri(String subjectUri);`	Returns a map of URI to label for roles that are considered appropriate for consideration for assignment to the newly created Project.

AddResearchDataToThingGenerator

Method	Description
`Map<String, String> getInheritedCustodians(String subjectUri);`	Returns a map of URI to label for custodians that are considered appropriate for consideration for assignment to the newly created ResearchData.
`Map<String, String> getInheritedCustodianDepartments(String subjectUri);`	Returns a map of URI to label for custodian department that are considered appropriate for consideration for assignment to the newly created ResearchData.

In addition to this, the `AddResearchDataToThingGenerator` class also sets up display and creation of custom data properties for the ResearchData object.

AddPublicationToThingGenerator

This class sets up the autocomplete processing for adding relationship between an existing subject and an existing `vivo:InformationResource` object. As such, this class does not require implementation of any additional methods. The bulk of the custom aspect of this class is in the implementation of `getFormSpecificData(VitroRequest vreq)`; this provides the details for hooking into the existing autocomplete functionality of VIVO.

AddOrCreateGenerator

This class is primarily useful in cases where creation of a new instance of an item requires a custom form which allows entry of some data type properties for setting at creation time. Unlike the other classes listed, implementation of this class requires creation of a zero argument constructor which then calls the overloaded constructor in this class with the following arguments:

Argument	Purpose
`String acType`	The fully qualified URI for the type to be used for the autocomplete field
`String typeName`	The user friendly name for the type to be used for the autocomplete field.
`String objectUriVar`	The name of the variable used to hold the URI of the selected object. Should be the same as the object name used in N3 statements within the class.
`Class editForm`	The form to which to forward if the user nominates to create a new item.

Preprocessors

The created preprocessors are contained within the following package:

```
edu.cornell.mannlib.vitro.webapp.edit.n3editing.configuration.preprocessors
```

`SetEntityReturnPreprocessor` extends `BaseEditSubmissionPreprocessorVTwo` and implements the required method:

```
void preprocess(MultiValueEditSubmission editSubmission);
```

This method is called during the processing of the values submitted by the form, and allows the instantiating generator to set a URI to which to forward. The URI is passed to the instance as part of an overloaded constructor. Whether or not to continue to the (possibly newly created) object of the new relationship is based on a variable set during form submission. This is used to allow having the “Add and Continue”, and “Add and Return” buttons on the relationship forms.

Freemarker

The Freemarker templates for the custom forms are located at:

```
<vivo-root>/productMods/templates/freemarker/edit/forms
```

The relationship level Freemarker templates are:

- `addProjectToAgreement.ftl`
- `addResearchDataToAgreement.ftl`
- `addResearchDataToProject.ftl`
- `addResearchDataToPublication.ftl`
- `addResearchDataToResearcher.ftl`
- `addOrCreateResearchData.ftl`

Each of these files include other Freemarker template fragments, via the `<#include .../>` directive. Common to all the templates is the `displayErrors.ftl` fragment which is used to display validation errors if the form submission fails.

Each of the templates that add ResearchData to an existing object include the `addResearchDataForm.ftl` fragment, which provides the display of the data fields for the ResearchData, and the `setupResearchDataInferences.ftl` fragment which sets up the listings of the inferred options for ResearchData items. In turn, `addResearchDataForm.ftl` includes the template fragment `displayResearchDataInferences.ftl` which displays the inferred options in the correct position, and the template fragment `setupChildOrParent.ftl` which sets up the checkbox for continuing to the object or to the subject after object creation.

Similarly, `addProjectToAgreement.ftl` includes `setupProjectInferences.ftl`, `displayProjectInferences.ftl` and `setupChildOrParent.ftl`, all of which provide the same functionality as their ResearchData counterparts.

In addition to these basic Freemarker templates, the generic template `addOrCreateResearchData.ftl` is used in cases where the workflow needs to allow the user to decide between assigning an existing item or creating a new one. This includes the `addExistingForm.ftl` fragment to simplify the first case.

Ontologies

As this project would require creation of additional object / datatype properties, a new ontology was created to house these properties. This ontology has been called `Unimelb RDR` and has a URI of `<https://rdr.unimelb.edu.au/config/>`.

To provide the linking between objects, the following object properties were created:

- `unimelb-rdr:agreementHasResearchDataOutput/unimelb-rdr:isResearchDataForAgreement`
- `unimelb-rdr:projectHasResearchDataOutput/unimelb-rdr:isResearchDataForProject`
- `unimelb-rdr:fundingVehicleFor/unimelb-rdr:hasFundingVehicle`

To allow more specific information to be added to the Research Data objects created by the user, the following datatype properties were created:

- `unimelb-rdr:digitalLocation`
- `unimelb-rdr:nonDigitalLocation`
- `unimelb-rdr:retentionPeriod`
- `unimelb-rdr:accessibility`
- `unimelb-rdr:dataManagementPlanId`

In addition to these property definitions, further configuration was added to the following pre-existing relationships to define the custom entry form that should be used for the relationship:

- `ands:isCollectorOf`
- `ands:relatedInformationResource`

- ands:relatedResearchData
- vivo:hasInvestigatorRole
- vivo:hasPrincipalInvestigatorRole

In order to ensure persistence of these assertions, they were stored within the filesystem to be picked up as follows. As part of the structure of VIVO, each file in the following paths:

- <vivo-root>/productMods/WEB-INF/filegraph/abox
- <vivo-root>/productMods/WEB-INF/filegraph/tbox

is considered for integration into the ontology of the running system. Each file is a listing of N3 assertions that can provide data structure and configuration for the VIVO system itself. For this project a number of additional files have been added to the existing VIVO definitions.

The following have been added to keep in step with the *Find An Expert* system:

- seo08.n3
- seo.n3
- rfcd.n3
- for.n3

The following add the definitions that are part of the ANDS requirements:

- ands.n3

The remaining custom assertion file contains the previously mentioned assertions:

- rdr-unimelb.n3

It is this final file in which modifications should be performed.

NB: Recompute of inferences when loading new or modified N3 assertions on a full data set can take a large amount of time, up to 36 hours has been witnessed on the current server. This needs to be taken into account when making modifications to the N3 files

Acronyms

- ANDS - Australia National Data Service
- ITS - Information Technology Services
- JVM - Java Virtual Machine
- N3 - Notation 3
- OS - Operating System
- OWL - Web Ontology Language
- UOM - The University of Melbourne
- URI - Uniform Resource Identifier
- VM - Virtual Machine
- RDF - Resource Description Framework
- RDR - Research Data Registry
- SPARQL - SPARQL Protocol And RDF Query Language (recursive acronym)
- XML - Extensible Markup Language