Google Summer of Code 2009 Submission Enhancements

<?xml version="1.0" encoding="utf-8"?>
<html>

Submission Enhancements Project

Details

- Student: Gaurav Kejriwal
- Mentor: Claudia Juergen,ARD Prasad
- Backup mentor: Stuart Lewis,Mark Diggory
- SCM Location for Project (prototype in sandbox): https://scm.dspace.org/svn/repo/sandbox/gsoc/2009/kejriwal/

Project Aims

1. Configure input-forms.xml via the user interfaces
2. Configure item-submission.xml via the user interfaces
3. The ability to allow the submitter to choose which input form to use (this facility to be turned on or off for particular collections)

Project Plan

Currently if the Dspace admin wants to change the looks of submission forms,or alter the submission workflow he has to change the desired xml files(input-forms.xml and item-submission.xml) . Also,there is no scope for item type based submission currently.We can solve this by developing UIs for the admin so that he can choose the options which he wants to be displayed to the user on the basis of item type.Also,we want to develop a UI so that the admin can decide the workflow steps.

Of all configuration DSpace has, input-forms.xml and item-submission.xml are the ones which end-users would continuosly want to be changing because administrators would want different submission forms for different collections or they may want to alter their collection's workflow.The logic to do so is simple enough that they could do it but being xml file in the server gets in the middle of that.

There is no scope of item type based submission right now. DSpace currently allows for the display of custom submission forms based upon the collection handle.But there is no feature to customize the forms based on the item types. For example we might need different metadata for different kind of item types like resolution for image,duration for audio and video files.

It would be very nice to have a 'Edit Input Forms' page with a similar layout and function to the 'Edit Metadata Registry' page, ie. Type the name of your form element, choose the input type (text,combo,checkbox, etc) from a dropdown list, set required yes/no, assign controlled vocabularies if necessary.
For the edit-input forms page we can have a link just like we have for metadata schema registry which will first link to the already present input-forms for that item. Here I can give an option to the Collection Administrator to add a new form or simply edit an existing one by clicking on the link of that form.
If one wants to edit an existing form a page will open displaying the existing elements and attributes of that submission form and he can add more elements or edit the existing elements/attributes.
Also additionally I can give the administrators an option to upload their desired form in the input-forms.xml format if they find the process of filling out the 'Edit Input Form' tedious.
For solving the problem of modification of submission workflow we can have a 'Edit Workflow Page' where admin can select the list of steps and their order to be taken for that collection through a UI. Here,also we can give an option for admin to upload the desired workflow in xml format which will ultimately be updated in the database.
Tasks:
We'll have to manage the input-forms.xml and item-submissions.xml either through database or we can just create a new updated input-forms.xml after the admin completes the process of choosing the options to be put in the input-forms.

Details

Present scenario
The dspace/config/item-submission.xml contains the submission configurations for both the DSpace JSP user interface (JSPUI) or the DSpace XML user interface (XMLUI or Manakin).
The Structure of item-submission.xml presently is as follows

```
<item-submission>
<!-- Where submission processes are mapped to specific Collections -->
  <submission-map>
    <name-map collection-handle="default" submission-name="traditional" /> ...
  </submission-map>

  <!-- Where "steps" which are used across many        submission processes can be defined in a
       single place. They can then be referred to by ID later. -->
  <step-definitions>
    <step id="collection">
      <processing-class>org.dspace.submit.step.SelectCollectionStep</processing-class>
      <workflow-editable>false</workflow-editable>
    </step>
    ...
  </step-definitions>

  <!-- Where actual submission processes are defined and given names. Each
       <submission-process> has        many <step> nodes which are in the order that
       the steps should be in.-->
  <submission-definitions>
    <submission-process name="traditional">
        ...
        <!-- Step definitions appear here! -->
    </submission-process>
    ...
  </submission-definitions>
</item-submission>
```

By default, this file contains the "traditional" Item Submission Process for DSpace, which consists of the following Steps (in this order):

Select Collection -> Initial Questions -> Describe -> Upload -> Verify -> License -> Complete

If one would like to customize the steps used or the ordering of the steps, he can do so within the <submission-definition> section of the item-submission.xml.

One may also specify different Submission Processes for different DSpace Collections. This can be done in the <submission-map> section.
The ordering of the <step> tags within a <submission-process> definition directly corresponds to the order in which those steps will appear!

For example, the following defines a Submission Process where the License step directly precedes the Initial Questions step.

```
<submission-process>
  <!--Step 1 will be to Sign off on the License-->
  <step>
    <heading>submit.progressbar.license</heading>
    <processing-class>org.dspace.submit.step.LicenseStep</processing-class
    <jspui-binding>org.dspace.app.webui.submit.step.JSPLicenseStep</jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.LicenseStenseStep</xmlui-binding>
    <workflow-editable>false</workflow-editable>
  </step>

  <!--Step 2 will be to Ask Initial Questions-->
  <step>
    <heading>submit.progressbar.initial-questions</heading>
    <processing-class>org.dspace.submit.step.InitialQuestionsStep</processing-class>
    <jspui-binding>org.dspace.app.webui.submit.step.JSPInitialQuestionsSteonsStep</jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.InitialQutialQuestionsStep</xmlui-binding>
    <workflow-editable>true</workflow-editable>
  </step>

  ...[other steps]...

</submission-process>
```

The same <step> definition is used by both the DSpace JSP user interface (JSPUI) an the DSpace XML user interface (XMLUI or Manakin). Therefore,we can notice each <step> definition contains information specific to each of these two interfaces.

The structure of the <step> Definition is as follows:

```
<step>
  <heading>submit.progressbar.describe</heading>
  <processing-class>org.dspace.submit.step.DescribeStep</processing-class>
  <jspui-binding>org.dspace.app.webui.submit.step.JSPDescribeStep</jspui-binding>
  <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.DescribeScribeStep</xmlui-binding>
  <workflow-editable>true</workflow-editable>
</step>
```

Current management of submission forms:
All of the custom metadata-entry forms for a DSpace instance are controlled by a single XML file, input-forms.xml, in the config subdirectory under the DSpace home.
The form-map maps collection handles to forms. DSpace does not require that a collection's name be unique, even within a community .DSpace does however insure that each collection's handle is unique.Form-map provides the means to associate a unique collection name with a form. The form-map also provides the special handle "default"(which is never a collection), here mapped to "traditional". Any collection which does not appear in this map will be associated with the mapping for handle "default".
The XML configuration file has a single top-level element, input-forms, which contains three elements in a specific order. The outline is as follows:

```
<input-forms>

  <--  Map of Collections to Form Sets -->
  <form-map>
    <name-map collection-handle="default" form-name="traditional" />
    ...
  </form-map>

  <--  Form Set Definitions -->
  <form-definitions>
    <form name="traditional">
    ...
  </form-definitions>

  <--  Name/Value Pairs used within Multiple Choice Widgets
        -->
  <form-value-pairs>
    <value-pairs value-pairs-name="common_iso_languages" dc-term="language_iso">
    ...
  </form-value-pairs>
</input-forms>
```

Each name-map element within form-map associates a collection with the name of a form set. Its collection-handle attribute is the Handle of the collection, and its form-name attribute is the form set name, which must match the name attribute of a form element.

For example, the following fragment shows how the collection with handle "12345.6789/42" is attached to the "TechRpt" form set:

```
  <form-map>
    <name-map collection-handle=" 12345.6789/42" form-name=" TechRpt\" />
    ...
  </form-map>

  <form-definitions>
    <form name="TechRept">
    ...
  </form-definitions>
```

Goals

We have to make UI so that the admin can select the type of form to be displayed for different item types for a particular collection.Also we have to make a UI so that the user can select the workflow steps,order,working class and processing class.
We have to add one step in the submission process which asks the user about the item type if it is enabled for that collection.

Implementation

We can either remove the notion of the xml files and move through their db management or we can just produce a new updated xml file once the admin completes the process in UI without disturbing the core Dspace database.

Moving through the db approach

I'll achieve the database management of forms and workflow by moving in a parallel framework to that of metadata schema registry.
The database management of input forms and configurable submission workflow will be somewhat similar.
I'll be creating separate db tables for each purpose like I'll create a new db table for input form with the following columns(item_type, page, schema, element, qualifier,repeatable, label, input-type, hint, required) and another for collection with columns like (Community|collection,steps,order).
For doing this in JSPUI interface I'll have to create a JSP file in which I can give input fields like text boxes for element,qualifier,label,hint,item_type, required,page,schema and I can give dropdown menus for repeatable and input-type.These values will be updated in the corresponding fields of the input-forms table.I will create another jsp which will list the existing forms and give an input box for entering the "item-type" in case the admin wants to add a new form.
I can write servlets with code to extract the values entered by the admin in these forms which can be used by functions of other classes.
I can write classes with code to extract the input forms from the database and to extract the different elements & attributes of a particular input-form from the db.
I'll also be creating classes which'll have functions for updating the values of element and attributes in the inputform table and also for updating the values of new input forms.
I can make a description step that will be reconfigured based on item "type".So there will need to be input forms per item type, which is just a column in that db table .
Now,when the user will start the submission process the DescribeStep class will display the desired form depending on the item type extracting values from the db instead of the xml file.
I'll have to change the DCInput.java file so that it extracts the various form elements with attributes from the database(input form table)rather than from the XML file. For example the original code

```
dcElement = (String) fieldMap.get("dc-element");
dcQualifier = (String) fieldMap.get("dc-qualifier");
```

can be changed to

```
dcElement = (String)DatabaseManager.func("itemtype","dc-element");
dcQualifier = (String) DatabaseManager.func("itemtype","dc-qualifier");
```

where func will be a function in DatabaseManager class to execute a query.

Similarly,I can get other attributes for each input element.
When the user will start the submission process first of all a page will list the item type in a dropdown menu and I'll use a servlet and a class that will lead to the desired submission form depending on a choice .
I'll also include functions for sanity checks to ensure that the user doesn't pass null/invalid values for elements and its attributes and to prevent other such faults.
For doing this for XMLUI interface,I'll develop in a similar framework to that which is present for editing a metadataschema registry or creating a new one.
I'll create classes similar to the editmetadata schema & metadataregistryMain class in the authorizations aspect which'll generate the DRI document of the desired format.
In one class I'll develop function to generate the DRI document to present the existing elements & attributes of an existing input-form in a tabular format.
First it'll extract the names and attributes of the input form elements using some other class's funcions.Then it'll be generating the DRI using the addTable, addCell etc. functions from Division and Table class.

Similarly,I'll be creating 2 other functions to generate the DRI for "Add New Element Form" and "Update Element Form" for adding new element or updating an element to/in the form.

I'll build another class to generate the DRI to show the existing input-forms list and to display a form for entering item-type for a new input-form.Here also I'll first extract the names and attributes(like item-type)of existing input-forms from db and generate the desired DRI to display it in a table using addTable, addCell etc. functions of Division and Table class.
In this class I'll also write code to generate DRI for creating a form to enter item-type.
The code will include statements like

```
Division newForm = div.addDivision("edit-form");
List form = newForm.addList("new-form",List.TYPE_FORM);
```

I'll take almost similar steps for the db mgmt. of submission workflow.For JSPUI interface I'll be creating JSPs to display the form through which the admin can select the various options for workflow configuration like steps,order etc.This'll be updated in the above mentioned table through a class.For doing this task for XMLUI interface ,I'll use similar classes to that developed for db management of input-forms.

For moving through the file based approach I'll just have to updated the xml files once the admin completes the UI process.I can change the structure of input forms to be allowed like this

```
<name-map collection-handle="default"
form-name="traditional, thesis, book, chapter"/>
```

instead of

```
<name-map collection-handle="default" form-name="traditional"/>
```

The process of making the UI for admin for JSP interface and XML interface will be somewhat similar to that of db approach just the values chooosen by admin will be stored in the xml file rather than db.Also at the time user submits,the form will be displayed by extracting values from file instad of db.

Timeline

I'll wait for the opinion of the community on whether we should move via db approach or file based approach during which time I'll be analyzing the implementation detalis.By June end I plan to complete the UI configuration of input forms.The task of UI management of input-forms and allowing item-type based submission are a bit related and will go hand in hand.I plan to complete both these tasks by mid July.I plan to complete the configuration of item-submission.xml via the user interfaces by beginning of August so that I'll have time for developing add-ons for various sanity checks.I'll keep around 10 days for testing and documentation in the end.

Feedback

Remarks on Submission Enhancement - (Richard Rodgers 06/08/09)

I agree that a DB-based solution is appropriate, and the general approach
of an InputField object and an InputForm that consists of fields
seems good. I would recommend **not** using the org.dspace.content
package, since that was meant only for data model objects (not all things represented in
the DB are data model objects. This is tricky since these 2 nearly coincide,
but this work is a good example of how they can differ). Why not org.dspace.input or some such?

In terms of the API, I would recommend a few small but significant changes to how the
problem is modeled:

I would generalize the concept of an InputField in a few dimensions:

A. A Field can be a common resource to many Forms. In your proposed implementation,
Each Field has a FormID, (e.g. public int getInputFormID()) so if a given field is
used in 20 forms, the DB has 20 rows differing only by FormID. It's not the storage
efficiency that worries me, it's the fact that making a change to a Field label, e.g., would
mean changing it 20 different times in the UI. Of course if one **wanted** different
behaviour per context of use, you could simple create another Field.

B. Relax the constraint implied by

```
public static InputFormField findByElement(Context context, int InputFormID,
            String element, String qualifier)
```

I think that is a limitation imposed by the current input-forms.xml that we
don't need to carry forward (i.e. the 1-to-1 input/md field relation)
Since some metadata schemas are very limited (e.g. QDC), we might very well want
different prompts and language (i.e. different InputFields) that map to the same
DC field. Thus:

```
public static InputFormField[] findByElement(Context context, int InputFormID,
            String element, String qualifier)
```

C. Add additional useful columns. I would include at least:

- InputType (One-Box, etc)
- Metadata Field (element.qual.language)
- Label
- Required
- Warning
- Repeatable
- Hint
- ValueListName (for controlled vocab)
- Visiblity
- Vocabulary (for controlled vocab)
- ClosedVocabulary (flag)
- DefaultValue

The last one bears some elaboration - this is essentially a finer-grained version of the
Item 'template' that in current DSpace is configurable only at the collection level.
If we add it here, we can greatly expand the usefulness of default values (by Item type, e.g.)
and get rid of the all the Template code in the data model.

D. Allow easy Internationalization. There are various possible techniques here:
One might be to allow message keys instead of or in addition to literal values
in the Fields' properties. Then locale-aware text could appear in the UI.

I would also maybe consider adding a third class - ControlledList ? so that you would not have
to duplicate value lists in many different Fields (they are separate elements in input-forms.xml,
so this would just be maintaining that approach)

Finally, I'm not sure how you are handling the mapping layer between collections and forms. I see an 'itemType' string field in the API for the Field, but I'm not sure it ought to be there if we consider Fields in this more general way. An additional table or tables could be used to map forms to any desired value - collection, ItemType, etc.

Project Status

These are the screenshots of the interfaces I have developed so far
Missing Files:

1.jpg

JSPUI3.jpg

Jspui1.png

XMLui1.bmp

XMLui2.bmp

XMLui3.bmp