Duke - Trident Project - Metadata Application Profiles

Duke University - Trident Project

Note:

This project is no longer in active development.

Explaining the Metadata Application Profile

At Duke, we have impemented a Fedora-based repository, and a web-based editor interface to manage the repository. The design is very modular, and very flexible. It was designed to be extended to fit different data models, as well as it is configurable to work with different descriptive metadata schemas. In addition, it was designed not only to work with different metadata schema, but can be configured to have different metadata requirements per "collection" (ie. Topical Subject is required for items in collection A but is optional for items in collection B). In order to make the editor configurable to work with different metadata schemas, we decided to make the editor completely dynamic and driven by data from the repository which instructs the editor how to create metadata forms for editing. I will briefly explain below what led us to this decision as well as explain how it works. There are also more details on our internal project wiki, http://library.duke.edu/trac/dc/wiki/Trident/MetadataApplicationProfile .

Editing XML Metadata in HTML Forms

The first challenge that we face when dealing with XML metadata is that we want to make it simple to edit. XML is certainly readable and there are editors available to simplify creating well-formed XML. But, in libraries, often we deal with complex schemas, and XML nodes have attributes, and sometimes we nest XML nodes inside other nodes, and some nodes are repeatable and others not, and some fields' values should be restricted to a defined authority list, and we can leave a lot open to the interpretation of the person who is editing the XML directly. In my experience, direct editing of the XML requires an intimate knowledge of the schema.

Also, direct editing of the XML is sometimes challenging when the XML is already integrated into a repository system, such as Fedora. We don't want our catalogers to have to navigate the Fedora Administrative Client in order to edit descriptive metadata.

Rather, we opt for web interfaces for editing the metadata. And we want to allow our catalogers to use html forms to create and update the descriptive metadata. With web forms, we can simplify the metadata entry, create contextual help for each metadata field, selects or lookups against authority lists, and we can provide mechanisms for error checking. OK, so we decided that we wanted to use HTML forms.

Mapping XML to HTML Forms

Now, we needed to figure out how to map the XML metadata into the HTML forms. HTML forms are very flat. XML is not necessarily so, nodes can be repeatable, nodes can have attributes, nodes can be optional or required, nodes can be nested inside other nodes. There are a lot of challenges that need to be considered.

The first step was to map the XML into a relatively flat format so that it could be turned into an HTML form. So, we decided that we would transform the native XML into a flat XML schema, which we call the Metadata Form schema, so that the Metadata Form could be consumed by the editor interface and turned into an HTML form. In the Metadata Form, we conceived of fields and field_groups. A field corresponds to a top-level element in the native XML schema, for instance, dc:title in Dublin Core. Since top-level elements can have defined attributes as well as further nested elements, we conceived of elements in the Metadata Form that are children of a field.

Let's say for the sake of a simple example, that we are using Dublin Core, and we want to map a dc:title element.

```
<dc:title type='main'>The main title</dc:title>
```

In our Metadata Form, this would map to:

```
<field name='title'>
  <element name='type'>main</element>
  <element name='title'>The main title</element>
</field>
```

We further defined field groups within our Metadata Form to handle repeatable fields. So adding on to the previous example, let's say we add an alternate title:

```
<dc:title type='main'>The main title</dc:title>
<dc:title type='alternate'>An alternate title</dc:title>
```

Using the same Metadata Form mapping:

```
<field_group name='title'>
  <field>
    <element name='type'>main</element>
    <element name='title'>The main title</element>
  </field>
  <field>
    <element name='type'>alternate</element>
    <element name='title'>An alternate title</element>
  </field>
</field_group>
```

As we will see later in this discussion, we may want to treat main and alternate titles separately. It may be the case that the main title is required and not repeatable, while the alternate title is optional and repeatable. So, this example might be modified to produce a slightly different Metadata Form:

```
<field_group name='main_title'>
  <field>
    <element name='title'>The main title</element>
  </field>
</field_group>
<field_group name='alternate_title'>
  <field>
    <element name='title'>An alternate title</element>
  </field>
  <field>
    <element name='title'>Another alternate title</element>
  </field>
</field_group>
```

The mappings in the above examples are done with XSLT. As you can imagine, the XSLT to produce a Metadata Form (MDF) from a different XML schema is rather straightforward, as well as is the XSLT to produce a native XML schema from a MDF. In our repository, we have a single mapping for all DukeCore descriptive metadata records (DukeCore is the descriptive metadata schema we use, based on Dublin Core). We can also create as many of these mappings as we have metadata schema in our system. So we could create a DC_TO_MDF mapping as well as a MODS_TO_MDF mapping (and the converse transformations MDF_TO_DC and MDF_TO_MODS). So, we're done, right?

Well, no. Although we now have mapped the descriptive metadata into a flat format for the editor, the editor has no clue how it should display these field_groups and elements to the end user. Which field groups are repeatable? Which elements are free text, which should be selects, and which should be auto-completes? If we want to use an authority list, how will the editor know which authority list to use?

Metadata Form Definitions

It would be short-sighted to think that our metadata schema will never change, or that we will always have just one descriptive metadata schema within our repository. Or, if someone from another university has an interest in using our editor tool, we don't want it tied to our specific metadata schema. So, we want the HTML forms to be dynamic, meaning we want them to be generated on the fly based on the metadata coming from the repository as well as some instruction from the repository on how the metadata fields should be displayed and edited.

So, at this point, we have the metadata in the MDF format, which is abstracted from the underlying metadata schema (could be MODS, could be DC, we don't care anymore). From this point on, the editor is really just interested in the MDF. But, the MDF doesn't provide any instruction on answering the above questions about how to display the fields for editing.

So, in addition to the XSLT mappings (one per schema) we have created Metadata Form Definitions. Metadata Form Definitions mimic the structure of the MDF, meaning that they contain field_groups, fields and elements. However, they do not contain the actual descriptive data and are embellished with attributes to instruct the editor how to display the data for editing. So, we have added a handful of cues to the MDF. The first of which is cardinality. Cardinality defines whether or not the metadata field is repeatable.

```
<field_group name='main_title' cardinality='single'>
  <field>
    <element name='title'/>
  </field>
</field_group>
<field_group name='alternate_title' cardinality='multiple'>
  <field>
    <element name='title'/>
  </field>
</field_group>
```

We also added labels and cues for whether or not the field group is required:

```
<field_group name='main_title' cardinality='single' required='yes' label='Title (main)'>
  <field>
    <element name='title'/>
  </field>
</field_group>
<field_group name='alternate_title' cardinality='multiple' required='no' label='Alt title'>
  <field>
    <element name='title'/>
  </field>
</field_group>
```

We also needed to instruct the editor how to display each element for editing, whether to use free-text box, a lookup field, a select. So we added a type attribute to the element in the metadata form definition. Since both of the fields in the example we have been using are free text inputs, I will add another metadata field in for example. The new field is Topical Subjects. In our editor interface, topical subjects should be drawn from a controlled authority list, and so should be input as a select.

```
<field_group name='main_title' cardinality='single' required='yes' label='Title (main)'>
  <field>
    <element name='title' type='text'/>
  </field>
</field_group>
<field_group name='topical_subject' cardinality='multiple' required='no' label='Topical Subjects'>
  <field>
    <element name='topical_subject' type='select' values='subject_topical'/>
  </field>
</field_group>
```

You will notice that there is an additional attribute, values. We have also developed an authority list application, and have designed the editor to work with the authority list application both for select drop downs and for auto-complete fields. The values attribute instructs the editor which authority list to use for which element.

Are we done yet?

Well, not really. There are a few more bells and whistles that we have added, error checking, styles, the ability to segment the editor form into multiple tabs /sections. And I did not really get into how the metadata application profiles are defined in the repository or related to individual items or collections. But, as far as this wiki page, I think we have covered the gist of the metadata application profile. For more information, please visit our project page, http://library.duke.edu/trac/dc/wiki/Trident .

We have designed this model to be extensible, so that it could meet others' needs (as well as make our own edits down the road less painful). We welcome feedback on our design. Please send comment to David Kennedy, david dot kennedy at duke dot edu.