CrosswalkPlugins

Contents

1 Metadata Crosswalk Plug-ins
   1.1 Renaming note
   1.2 XML Formats Only
   1.3 Sample Implementation
2 Where Do I Use Crosswalk Plugins?
   2.1 OAI-PMH plugin-driven Crosswalk
3 Dissemination
   3.1 Configuration
   3.2 Dissemination Issues
   3.3 Submission (Ingest)
   3.4 Ingestor Issues
   3.5 Exceptions
      3.5.1 CrosswalkInternalException
      3.5.2 MetadataValidationException
   3.6 Generics

Metadata Crosswalk Plug-ins

We use a few different metadata standards in DSpace in various places:

- Qualified Dublin Core
- Simple Dublin Core (oai_dc in OAI-PMH code)
- MODS (METS exporter)
- METS
- Simple batch importer/exporter format

However, they're used differently in different places. E.g. MODS is in the METS exporter code, Simple DC is in the OAI-PMH code. Why can't the OAI-PMH code serve up MODS? Then we could have e.g.a single batch import/export tool to concentrate our development efforts on instead of having multiple ones.

This proposal is only concerned with importing and exporting metadata; see PackagePlugins for a parallel module to handle packages (SIPs and DIPs). The packager plugins call these crosswalk plugins to handle the metadata when ingesting or disseminating.

Ingestion in general is far more complicated and awkward than dissemination, so they are considered separately, and naturally the easiest is considered first.

Renaming note

After conversations with Richard R & Larry, various renaming has been done, for consistency and predictable behaviour.

*SubmissionCrosswalk -> *IngestionCrosswalk
MetsDissemination -> METSDisseminationCrosswalk
ModiCrosswalk -> MODSDisseminationCrosswalk
NullSubmissionCrosswalk -> NullIngestionCrosswalk
PremisCrosswalk -> PREMISCrosswalk
SimpleDCCrosswalk -> SimpleDCDisseminationCrosswalk
Xslt*Crosswalk -> XSLT*Crosswalk

XML Formats Only

The Crosswalk Plugin interface described here only addresses XML-based metadata formats. Since OAI-PMH can only export XML, and metadata containers like METS and IMS-CP have a preference for XML metadata, this is not seen as an important limitation at this time. If there is a need, anyone can add a new plugin interface to handle binary or text-based metadata (e.g. old-style MARC).

Sample Implementation

This file contains the interfaces and some sample crosswalk implementations. It is still highly experimental and subject to sudden changes so do not rely on the stability of this code: crosswalk2.zip

Also see XsltCrosswalk for another sample use of this plugin.

Where Do I Use Crosswalk Plugins?
Whenever a DSpace object has to translate its metadata into some external metadata format, and whenever an external metadata record is applied to a DSpace object, call on a Crosswalk Plugin. All crosswalk activity should live in the plugins, so every crosswalk developed for one purpose can be shared by all the consumers of crosswalks.

Consumers are typically:

- The OAI-PMH metadata provider server.
- Network-based Package ingest and dissemination e.g. through LightweightNetworkInterface
- Batch (command-line) ingest of packages through PackagerPlugins
- Classic &lt;i&gt;ItemImporter&lt;/i&gt; batch importer.

OAI-PMH plugin-driven Crosswalk

This implementation includes a module for the OAI-PMH server, &lt;i&gt;oaicat&lt;/i&gt;, which lets it use any &lt;i&gt;DisseminationCrosswalk&lt;/i&gt; plugin. The single class &lt;i&gt;org.dspace.app.oai.PluginCrosswalk&lt;/i&gt; implements any metadata prefix that matches the name of a dynamic plugin crosswalk.

Just add lines like these to &lt;i&gt;oaicat.properties&lt;/i&gt;:

```properties
Crosswalks.MODS=org.dspace.app.oai.PluginCrosswalk
Crosswalks.OCW-LOM=org.dspace.app.oai.PluginCrosswalk
```

Dissemination

For dissemination, the metadata crosswalk turns an item’s internal DC values into a serialized representation, such as XML.

Note that metadata disseminations can be nested. For example, the METS format is actually a framework that includes (or refers to) objects in other standard formats. One disseminator could produce METS with MODS descriptive metadata, while another produces METS with DC.

Here is the disseminator interface, as implemented experimentally:

```java
public interface DisseminationCrosswalk {
    // returns array of namespaces, which may be empty.
    public org.jdom.Namespace[] getNamespaces();

    // returns SchemaLocation string, including URI namespace,
    // followed by whitespace and URI of XML schema document, or
    // empty string if unknown.
    public String getSchemaLocation();

    // predicate, true if the given object can be crosswalked.
    public boolean canDisseminate(DSpaceObject dso);

    // returns results of crosswalk as list of XML elements.
    public java.util.List disseminateList(DSpaceObject dso)
        throws CrosswalkException,
        IOException, SQLException, AuthorizeException;

    // returns results of crosswalk as one XML element, root of document.
    public org.jdom.Element disseminateElement(DSpaceObject dso)
        throws CrosswalkException,
        IOException, SQLException, AuthorizeException;
}
```

Note: The Dissemination methods do not have a Context object parameter, since it is not required to get an object’s DC values, and some callers (notably the OAI-PMH server) don’t have a context available.

Since the disseminator is a dynamic plugin, use the PluginManager to get one:

```java
DisseminationCrosswalk crosswalk =
    PluginManager.getNamedPlugin(DisseminationCrosswalk.class, "MODS");
```

```java
List mods = crosswalk.disseminateAsXml(item);
```

Configuration
Crosswalk plugins are "configured" by being listed as dynamic plugins in the PluginManager configuration properties; it does the rest. A single class may implement both <i>DisseminationCrosswalk</i> and <i>SubmissionCrosswalk</i>.

**Dissemination Issues**

The main issue being, is an `Item` object the most appropriate to pass in? Perhaps just a Handle? Or database ID? We need to enable efficient implementations of this, but at the same time the 'MetadataDisseminator' implementation shouldn't have to do all the work.

- I think an object makes the most sense, maybe let it be a <i>DSpaceObject</i> if we can stomach writing metadata for communities and collections. (Implementations would have the option of returning a "not implemented" exception.) --lcs

/*! Changed to pass a <i>DSpaceObject</i> to disseminator, since collections and communities have metadata too; however, each class doesn't need to implement it. */

The submission crosswalk only needs to handle Items since only items have DC metadata, and we only "import" Items.

Passing in options. Disseminators might have common or specific parameters. e.g.:

- "Disseminators" the right word? Maybe just "crosswalk" or `createPackage`  
  - I vote "crosswalk" --lcs  
  - Filter out certain things? e.g. in OAI-PMH export of METS, you may want to exclude some provenance information. Already in OAI-PMH export, `description.provenance` is filtered out for privacy reasons. (It includes the email address of submitters.)  
  - Package disseminators might want to know which metadata format to package. e.g. METS could include Dublin Core, MODS, or anything else...  
    - It would be helpful to parameterize a crosswalk "type" in the configuration. This could get hairy since they would each want different sets of parameters (e.g. DC has none, METS has several sub-formats).  
    - This could be an issue for a generalized plug-in framework to solve; each instance of a plugin is further specialized by a set of other plugins. The PackagePlugins could use a mechanism like that to refer to crosswalk plugins, as well. <B>NOTE: I think the best way to do this is a superclass parameterized by subclassing it; harder to code but gives more flexibility, and since metadata has to meet outside specifications it is probably not the kind of thing you want to be configuring on the fly anyway.</B>

**Submission (Ingest)**

Trickier. This will need to support 'ingest new stuff' as well as 'update stuff that's already there'; and also to support ingesting stuff that already has a persistent ID (such as a Handle) rather than having a new one created by the system.

The contract of a metadata ingester is to interpret the XML structure it is given as metadata values, and set the appropriate values in the DSpace Object (e.g. Item) metadata. See MetadataSupport for a proposal to attach metadata fields from schemas other than DC to Items.

Here is a possible interface:

```java
public interface SubmissionCrosswalk
{
    // crosswalk from root element of a document
    public void ingest(Context context, DSpaceObject dso, org.jdom.Element root)
        throws CrosswalkException, IOException, SQLException, AuthorizeException;

    // crosswalk from list of metadata fields
    public void ingest(Context context, DSpaceObject dso, java.util.List elements)
        throws CrosswalkException, IOException, SQLException, AuthorizeException;
}
```

**Ingester Issues**

Should the behavior be different when the Item already has valid metadata? Is there a need for a filter, limiting what fields can be set?

The ingester has the same nesting configuration issue as the disseminator; a framework format such as METS may call on several other ingesters to interpret the metadata embedded in (or linked from) its stream.

**Exceptions**

The Submission and Dissemination crosswalk plugins share a family of exceptions, under the superclass 'CrosswalkException'. They are:

**CrosswalkInternalException**

Something went wrong inside the crosswalk, not necessarily caused by the input or state (although it could be an incorrectly handled pathological case). This is most likely a configuration problem. It deserves its own exception because many crosswalks are configuration-driven (e.g. the XSLT crosswalks) so configuration errors are likely to be common enough that they ought to be easy to identify and debug.

**MetadataValidationException**
This indicates a problem with the input metadata (for submission) or item state (dissemination). It is invalid or incomplete, or simply unsuitable to be crosswalked.

Generics

I think the interface should use Java generics to allow compile time data type checking. I think everywhere we currently use the Java collection API we should use generics. Thus, replace `<i>java.util.List</i>` use `<i>java.util.List&lt;org.jdom.Element&gt;</i>`. This will require that DSpace be compiled with a Java 1.5 compiler (However an older jvm, such as 1.4, will still be able to execute the bytecode). – ScottPhillips