PledgePolicyPrototype

Contents

Policy Storage Prototype

This page describes the *Policy Repository* that was created for the PLEDGE Project, an add-on-style extension to DSpace 1.5.

Introduction

What do we mean by Policy?

In this context, *Policy* refers to the general definition of the word rather than the specific meaning it has acquired in the authorization area of the DSpace data model (e.g. around the ResourcePolicy and PolicySet classes).
On this page, a *policy* is typically either a rule describing (or prescribing) the interactions of actions that take place within the archive, or a constraint determining when and by whom an action may be taken.

For example, a policy could demand that every Item being submitted include an approved deposit license. Another policy might demand that every Bitstream in the asset store be checked for content integrity (i.e. checksum recomputed and compared with the checksum on record) at least once in every six moths.

A Policy Repository

Since one of the goals of the PLEDGE project was to develop a machine-readable encoding of policies, we needed a place to put them. Policies may be associated with all types of data model objects: Items, Collections, Communities, etc. It would have required database changes to manage this within the existing object model.

However, the data we have to store is in RDF, and History System Prototype includes an RDF triple-store that can associate RDF statements with instances of DSpace objects, all outside of the DSpace data model.
The

```
RDFRepository
```

class from the History system is easily subclassed to serve as a policy store as well.

Implementation Details

Policies are written in the Rei language,
which is based on RDF and OWL. A policy is thus a collection of
RDF statements. See the Rei examples in these
samples of policy expressions.

Binding of Policies to DSpace Objects

The

```
PolicyRepository
```

class lets you store RDF statements
keyed to a DSpace object, so they can be retrieved later in the context
of the object. An object thus accumulates policies bound to it.

The policy information model assumes that the policies of an object
also apply (when relevant) to objects below it in the "ownership" hierarchy.
For example, a policy dictating replication terms at the Community level
would also apply to each Collection and Subcommunity under that Community,
and to the Items, etc, belonging to them.

In practice, this "inheritance" behavior would be implemented by a policy
enforcement engine, but that has not even been designed yet. It is sufficient
for the policy repository to retrieve the policies related to one DSpace
object; a policy engine or other application can use the data model API
to find other related objects (e.g. ancestors) and retrieve their policies.

As an example, the class

```
PolicyStackStreamDisseminationCrosswalk
```

exports all of the policies belonging to an object *and* to its "stack"
of owner/ancestors – its owner, its owner's owner, etc. on up to the Site.
This was done so that *all* possibly-relevant policies can be put into
a Dissemination Information Package (DIP) which is sent to a policy-aware
storage repository such as the SDSC's iRODS.

Installation

Prerequisites

- Latest DSpace 1.5 development source
- Install History System Prototype, which requires, in order:
    1. Install the Event System prototype
    2. Install the AIP prototype patch
    3. Install History System Prototype

Download and Install

1. Download Policy-new-files.zip and <tt>unzip</tt> it in your DSpace install (source) directory.
2. Download Policy-dspace_cfg_diff.mht and apply to <tt>dspace/config/dspace.cfg</tt> with the <tt>patch</tt> utility. Note that you must update the config file used by the running DSpace instance.

1. To install, rebuild and install <tt>dspace.jar</tt> with the command: (There is no need to rebuild the WARs since the UIs never call the policy repository.)

```
ant install_code
```

2. Also, be sure the configuration changes are installed in the DSpace
configuration file: <tt>dspace/config/dspace.cfg</tt> .

3. Finally, create the directory mentioned in the configuration
as the value of <tt>policy.dir</tt>, e.g. <tt>dspace/policy</tt> .
Be sure it is writable by the user who runs DSpace.

Usage

These examples assume the following contents in the archive, so substitute
equivalent objects in your archive:

- <tt>123456789/8</tt> - an Item
- <tt>123456789/7</tt> - a Collection
- <tt>123456789/3</tt> - a Community
- <tt>123456789/0</tt> - the Site

Run the <tt>PolicyRepository</tt> command-line application with <tt>--help</tt>
to learn about all of its options. It is the same as the
<tt>HistoryRepository</tt> application in the History System Prototype.

```
dsrun edu.mit.pledge.PolicyRepository --help
```

Adding Policies to Objects

- Add Deposit Agreement policy to Item 123456789/8
  <tt>dsrun edu.mit.pledge.PolicyRepository -s 123456789/8 Policy-cu0006rei.xml </tt>
- Add Replication policy to Collection 123456789/7
  <tt>dsrun edu.mit.pledge.PolicyRepository -s 123456789/7 Policy-tu0011rei.xml
- Add Public Availability policy to Community 123456789/3
  <tt>dsrun edu.mit.pledge.PolicyRepository -s 123456789/3 Policy-cu0008rei.xml
- Add required-metadata policy to the Site (123456789/0)
  <tt>dsrun edu.mit.pledge.PolicyRepository -s 123456789/0 Policy-pp0004rei.xml

Examining Policies

This command will "disseminate" all of the policies associated with
an object identified by Handle, in this case the Site:

```
dsrun edu.mit.pledge.PolicyRepository -d 123456789/0
```

Add the <tt>-f</tt> option to change the output format to e.g. N3:

```
dsrun edu.mit.pledge.PolicyRepository -f n3 -d 123456789/0
```

Make an AIP that includes Policies

Be sure your DSpace Configuration includes a line like this:

```
aip.disseminate.techMD = PREMIS, AllPolicies:POLICY_STACK, ObjectPolicies:POLICY
```

The following command creates an AIP of the Item 123456789/8

```
org.dspace.app.packager.Packager -d -t AIP -i 123456789/8 -e ADMIN-USER policy-aip.zip
```

You can download the sample AIP here Policy-aip.zip

Note that the <tt>mets.xml</tt> manifest includes the element:

```
<techMD ID="techMD_7">
<mdRef LOCTYPE="URL" xlink:type="simple" xlink:href="metadata_69"
MDTYPE="OTHER" OTHERMDTYPE="AllPolicies" MIMETYPE="text/xml"/>
</techMD>
```

This identifies <tt>metadata_69</tt> as the whole "stack" of policies that
applies to the Item. Another file in the AIP, <tt>metadata_67</tt>,
contains just the policies that are actually bound to the Item itself.

Future Work

This is only an experimental prototype. The
<tt>PolicyRepository</tt> implementation is crude, but it is
adequate to get policy metadata in the archive and into AIPs for
experimenting with other, policy-aware, repositories.

Next Steps

- Deploy and use this prototype to test it.
- Consider whether there is a need to modify a policy's statements as they are added to the repository, e.g. inserting the identifiers of specific objects and concrete values to replace variables or placeholders in the RDF.

- Editing or removing policies is not implemented. The History System's RDF repository is only concerned with adding statements, and has no mechanism to group the statements belonging to a policy (i.e. a subset of the statements bound to an object) so they can be removed or replaced as a group. (Perhaps use RDF reification to implement grouping.)

</html>