Logical Item Filtering and DOI Filtered Provider for DSpace

Section One: DSpace Logical Item filtering (org.dspace.content.logic.*)

Inspired by the powerful conditional filters in XOAI, this component offers a simple but flexible way to write logical statements and tests, and use the results of those tests in other services or DSpace code.

LogicalStatement

LogicalStatement is a simple interface ultimately implemented by all the other interfaces and classes described below. It just requires that a class implements a method.

```
  Boolean getResult(context, item)
```

Filters

Filters are at the root of any test definition, and it is the filter ID that is used to load up the filter in spring configurations for other services, or with DSpace Service Manager.

A filter bean is defined with a single "statement" property - this could be an Operator, to begin a longer logical statement, or a Condition, to perform a simple check.

There is one simple implementation of Filter included - DefaultFilter.

Operators

Operators are the basic logical building blocks that implement operations like AND, OR, NOT, NAND and NOR. An Operator can contain any number of other Operators or Conditions.

So statements like this can be created:

```
  (x AND (y OR z) AND a AND (b OR NOT(d)))
```

Conditions

Conditions are where the actual DSpace item evaluation code is written. A condition accepts a Map<String, Object> map of parameters. Conditions don't contain any other LogicalStatement classes – they are at the bottom of the chain.

A condition could be something like MetadataValueMatchCondition, where a regex pattern and field name are passed as parameters, then tested against actual item metadata. If the regex matches, the boolean result is true.

Typically, commonly used Conditions will be defined as beans elsewhere in the spring config and then referenced inside Filters and Operators to create more complex statements.

Configuring Filters in Spring

Conditions, Operators and Filters are all defined in $dspace/config/spring/api/item-filters.xml

Here's a complete example of a filter definition that implements the same rules as the XOAI openAireFilter. As an exercise, some statements will be defined as beans externally, and some will be defined inline as part of the filter.

New Condition: driver-document-type_condition

This condition creates a new bean to test metadata values. In this case, we're implementing "ends with" for a list of type patterns.
New Condition: item-is-public_condition

This condition accepts group and action parameters, then inspects item policies for a match - if the supplied group can perform the action, the result is true.

New Filter: openaire_filter

Here is the full definition for the OpenAIRE filter.

The first statement is an And Operator, with many sub-statements – four Conditions, and an Or statement.

The first two statements in this Operator are simple Conditions defined in-line, and just check for a non-empty value in a couple of metadata fields.

The third statement is a reference to the document type Condition we made earlier:

The fourth statement is another Operator, in this case an Or Operator with two Conditions (the is-public Condition we defined earlier, and an in-line definition of as “is-withdrawn” Condition)

The fifth statement is an in-line definition of a Condition that checks dc.relation metadata for a valid OpenAIRE identifier.

So the full logic implemented is:

(has-title AND has-author AND has-driver-type AND (is-public OR is-withdrawn) AND has-valid-relation)
Running Tests on the Command Line

There is a launcher command that can arbitrarily run tests on an item or all items, eg.

```
${dspace}/bin/dspace test-logic -f openaire_filter -i 123456789/100
```

A simple true or false is printed for each item tested.

Using Filters in other Spring Services

The Filter beans can be referenced (or defined) in other services, for instance, here is adding the bean we configured earlier, as a filterService to a new FilteredDOIIdentifierProvider:
In the provider, we just define the property with the other services and class variables:

```java
private Filter filterService;
```

And make sure there is a setter for it:

```java
@Required
public void setFilterService(Filter filterService) {
    this.filterService = filterService;
}
```

Then you can actually run the tests with the service, like this:

```java
try {
    Boolean result = filterService.getResult(context, (Item) dso);
    // do something with result
} catch(LogicalStatementException e) {
    // ... handle exception ...
}
```

In the TestLogicRunner, you can see a way to get the filters by name using the DSpaceServiceManager as well.

## Section Two: DOI Filtered Provider

### New FilteredProvider: DOIIdentifierProvider

DOIIdentifierProvider now extends a base FilteredIdentifierProvider, which looks for any configured filters and only allows minting DOIs for items where the filter returns `true`.

This filter is always applied to the DOI consumer and other internal DOI service calls, and is applied by default to the `doi-organiser` tool (though it can be optionally skipped with a command-line argument).

The filter is a spring property configured in identifier-service.xml, in the provider bean declaration.

The `filterService` property is **optional**. If it is missing from spring configuration, all items will get DOIs minted as per normal and the provider's filter service will be null.

It is defined as follows:

```xml
<bean id="org.dspace.identifier.DOIIdentifierProvider"
     class="org.dspace.identifier.FilteredDOIIdentifierProvider"
     scope="singleton">
    <property name="configurationService" ref="org.dspace.services.ConfigurationService" />
    <property name="DOIConnector" ref="org.dspace.identifier.doi.DOIConnector" />
    <property name="filterService" ref="openaire_filter"/>
</bean>
```

Where the "openaire_filter" reference is the ID of a filter bean defined in item-filters.xml

### New Curation Task:

In DSpace 5 and 6 implementations of this feature, JSPUI and XMLUI buttons were added to the Edit Item administrative pages so that DOIs could be manually registered (queued for registration) by administrators, explicitly **skipping** the filter.

In the DSpace 7 implementation, this feature can be used via the existing curation task framework, either in the CLI or in the Angular UI (when curation tasks are implemented).

#### Configuration

This task is configured in curage.cfg as 'registerdoi' with the label "Register DOI".

There is a configuration file in `${dspace}/config/modules/doi-curation.cfg` that can be used to customise the behaviour regarding filter skipping, and distribution over multiple items.
```yaml
### DOI registration curation task configuration module

## Should any logical filters be skipped when registering DOIs? (ie. *always* register, never filter out the item)
# Default: true
#doi-curation.skip-filter = true

## Should we allow the curation task to be distributed over communities / collections of items or the whole repository?
# This *could* be dangerous if run accidentally over more items than intended.
# Default: false
#doi-curation.distributed = false
```