Configurable Entities

- Introduction
- Configuration of item types and their relations
  - Re-using one of the models previously created in the community
    - Configuration of the submission forms per item type
    - Configuration of workflow for item types
    - Configuration of community/collection list for item types
  - Designing your own model
    - Thinking about the object model
    - Configuring the object model
    - Configuring the metadata fields
    - Configuring the item display pages
    - Configuring virtual metadata
    - Configuring discovery

Introduction

The DSpace working group has expressed the need for DSpace to be able to provide more support for different types of digital objects related to open access publications, such as authors/author profiles, data sets etc. The objectives of the working group were to:

- Create awareness on the topic of additional entities
- Propose a solution
- Specify the global architecture and requirements
- Propose a roadmap that was not tied to a particular version of DSpace

The working group has decided the following design principles should be applied:

- We would avoid hard-coding a particular object model, for example the code shouldn't contain specific Java classes for specific objects such as Person, Project, Dataset, etc.
- The Implementation of the data model should be configurable so that any institution can define which additional objects would require more detailed/elaborate support in DSpace and which objects can be represented by the generic default Item DSpace object.
- The design will start from the DSpace Item object and extend it.
- And in order to provide more specific support for particular entities, items can be typed and relations between typed items can be defined through configuration

For this implementation, the current DSpace Item object has been used to avoid hardcoding a particular object model. This is in order to be able to have a low barrier to use all the existing functionality in DSpace that already supports the item object for any new entities, such as the submission/workflow process, search, batch import, OAI-PMH, etc.

The model has similarities with the Portland Common Data Model, but is less focused on building a tree structure, and more on building a graph structure as is required for certain object models such as a CRIS object model. Where PCDM is focused on a tree structure with the related object as an addition to include some basic graph-like functionality, this object model starts immediately from the graph concept.

The relations are stored in a separate database table, which is the only change needed to the current DSpace database structure, using foreign keys to the item table to ensure the relations are not broken, and to avoid any data replication.

The implementation of a specific object model is completely configurable, without using specific hardcoded Java classes for the objects used in a specific object model.

Configuration of item types and their relations

There are 2 approaches possible here:

- Re-using one of the models previously created in the community
- Designing your own model

Both parts will be explained in detail

Re-using one of the models previously created in the community

The models currently available in the community are:

- Research Entities
- Journals
- OpenAire (in progress)

In order to start using such a model, the following steps should be used

Configuration of the submission forms per item type
A collection should contain only objects of one item type. E.g. there would be one (or even multiple) collections for Person objects. The submission forms of the collection(s) for Person objects would contain metadata only applicable for the Person item type. Since the item type is stored in the metadata, the collection template can be used to store the item type metadata value.

There’s no need to remain stuck in the model where collections represent departments, and both Publication and Person objects should be stored under the department’s collection.

- In the edit collection, an item template should be created containing relationship.type with value Publication, Person or whatever type should be submitted to that collection.
- In the edit collection, the submit group should be defined to identify who can submit to this collection.
- In item-submission.xml, the collection handle should be linked to the applicable submission-definitions in the file, e.g.

```xml
<name-map collection-handle="123456789/6" submission-name="People" />
<name-map collection-handle="123456789/279" submission-name="People" />
<name-map collection-handle="123456789/7" submission-name="Project" />
<name-map collection-handle="123456789/280" submission-name="Project" />
<name-map collection-handle="123456789/8" submission-name="OrgUnit" />
<name-map collection-handle="123456789/28" submission-name="Journals" />
<name-map collection-handle="123456789/29" submission-name="JournalVolumes" />
<name-map collection-handle="123456789/30" submission-name="JournalIssues" />
```

**Configuration of workflow for item types**

The DSpace workflow can be used for reviewing all objects in the Object Model since these objects are all items, and separate collections can be used. The workflow used for e.g. a Person Object can be configured to be identical to a publication, different from a publication, or use no workflow at all.

The users who are granted permissions to create a Person Object are determined by the submitters group of that collection.

**Configuration of community/collection list for item types**

The community list is very flexible, and institutions can use it for various purposes. By using collections for the item types, additional collections can be created in the community/collection list. Extending existing community/collection structures to include collections per item type, can be done using a single collection for the other item types. This is typically useful if the structure is not applicable for the other item types, or should not be displayed. In this use case, it would also be possible to not grant Anonymous READ rights on the People, Projects collections to ensure they’re not visible on the community/collection list page. Or extending these to include collections per item type, can be performed using multiple collections for the other item types. This is typically applicable when they should be clearly displayed in the structure:

**Structure based on the departments:**

- Department of Architecture
  - Building Technology Program
  - Theses - Department of Architecture
- Department of Biology
  - Theses - Biology
- People
- Projects

**OR**

- Department of Architecture
  - Building Technology Program
  - Theses - Department of Architecture
  - People in Department of Architecture
  - Projects in Department of Architecture
- Department of Biology
  - Theses - Biology
  - People in Department of Biology
  - Projects in Department of Biology

**Structure based on the publication type:**

- Books
  - Book Chapter
  - Edited Volume
  - Monograph
- Theses
  - Bachelor Thesis
  - Doctoral Thesis
  - Habilitation Thesis
  - Master Thesis
- People
- Projects

**OR**
Designing your own model

When using a different entities model, the new model has to be configured and loaded into your repository.

Thinking about the object model

First step: identify the entity types

- Which types of objects would you want to create items for: e.g. Person, Publication, JournalVolume
- Be careful not to confuse a type with a relationship. A Person is a type, an author is a relationship between the publication and the person

Second step: identify the relationship types

- Which relationship types would you want to create between the entity items from the previous step: e.g. isAuthorOfPublication, isEditorOfPublication, isProjectOfPublication, isOrgUnitOfPerson, isJournalIssueOfPublication
- Multiple relationships between the same 2 types can be created: isAuthorOfPublication, isEditorOfPublication
- Relationships are automatically bidirectional, so no need to worry about whether you want to display the authors in a publication or the publications of an author

Third step: visualize your model

- By creating a drawing of your model, you’ll be able to quickly verify whether anything is missing

Configuring the object model

Configure the model in relationship-types.xml

- Similar to the default relationship-types.xml, configure a relationship type per connection between 2 entity types
- Include the 2 entity type names which are being connected
- Determine a clear and unambiguous name for the relation in both directions
- Optionally: determine the cardinality (min/max occurrences) for the relationships
- Optionally: determine default behavior for copying metadata if the relationship is deleted

Configuring the metadata fields

Determining the metadata fields to use

- Dublin Core works for publications, but not for a Person, JournalVolume, …
- There are many standards which can be easily configured: schema.org, eurocris, datacite, …
- Pick a schema which suits your needs
Configure the submission forms

- Add a form in submission-forms.xml for each entity type, containing the relevant metadata fields
- Configure which relationships to create

**Configuring the item display pages**

- The metadata configuration is not specific to configurable entities.
- Similar to other customizations to the item display pages, configure in Angular which metadata fields to display and their label. A template per entity type can be created
- The relationship display is similar to the metadata configuration
- Similar to the metadata configuration: configure in Angular which relationship to display and their label

**Configuring virtual metadata**

- The isAuthorOfPublication relationship can be displayed for the Publication item as dc.contributor.author
- The isOrgUnitOfPerson relationship can be displayed for the Person item as organization.legalName
- This can be configured in virtual-metadata.xml

**Configuring discovery**

- Configure the discovery facets, filters, sort options, …
  - The facets for a Person can be job title, organization, project, …
  - The filters for a Person can be person.familyName, person.givenName, …