VIADAT-DEPOSIT Documenation

Metadata and metadata workflow for oral history

Adaptation and installation

As the domain where the software is used changed from linguistic data and software to oral history, a different metadata schema and a submission workflow was required for VIADAT-DEPOSIT.

The proposed schema distinguishes/describes three „entities“ – the interview itself, the narrator and technical details. The submission itself distinguishes several types of files – consents from narrators, outputs and other materials, the interview itself and its transcription.

The narrator is assigned the following metadata: name, surname, id, alternative name (alias), academic degree, gender, date of birth, the project, consent, keywords, occupation, period, other characteristics, contact details and notes. The first three are mandatory; degree, project, keywords, occupation, period and other characteristics are extensible lists; gender and consent are lists of predefined values (plus there’s the option to upload the consent), date is in a predefined format.

The interview is assigned the narrator, file name, id, interview type, transcription available, date of the interview, length of the interview, place of the interview, the interviewer, annotation, keywords, codes, language, notes. The filename and id are mandatory, type, place, interviewer, keywords and codes are extensible lists; transcription available and language are lists of predefined values; date and length have a predefined format.

The technical details are assigned to the interview; they describe the file type, the format, the size, the creator, the license/rights and additional notes. The file type is mandatory; the creator and rights are extensible lists.

In theory, some of the metadata can be extracted in an automated way. Such extractions will be explored in the future.

Technically most of the metadata will be stored in newly created „viadat“ namespace – to give us the option to define our own semantics of the fields. For compatibility with the UI some fields are in the „dc“ namespace. If need for cooperation with software outside of the VIADAT project arises a mapping of the metadata scheme can be prepared (similar to what’s described below).

The current workflow mimics the three entities; it’s expected that additional changes will be required; these might be splitting the forms in multiple pages (in input-forms.xml) or addition of the automatic extraction steps (in item-submission.xml).

OAI-PMH

The Open Archives Initiative goal is to ease access to material on the Web. They decided to do so by devising a protocol that enables **Data providers** (web servers/repositories owners) to expose their "data about data" (metadata). The exposure happens in a way that allows **Service providers** to programmatically retrieve (**harvest**) these metadata and build some value added services on top of them (ie. adding them to a larger, searchable, collection). The protocol is called Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH).

The *service providers* issue HTTP requests to *data providers* and receive XML responses. The protocol does not define a format (schema?) for the metadata. It is up to particular *Data* and *Services* providers to come up with format that suits their needs. A *Data provider* can even offer more formats. However, among these formats, for interoperability purposes, there must be unqualified [Dublin Core](Metadata_info#Dublin Core).

As from the point of view of this project, we want to (have to?) be roles - Data and Service provider. We should support Meta-share schema and also offer metadata in cmdi format.

To see currently supported formats check https://ufal-point.mff.cuni.cz/oai/request?verb=ListMetadataFormats

Crosswalks

Metadata crosswalk contains mapping from one metadata element set to another. I.e., which elements are semantically close to witch. The mapping usually discards some information as it usually goes from fine grained elements to more coarse one. Eg. creationDate -> date. Ingestion and dissemination crosswalks kind of describe in which way the mapping "goes". Suppose you have all metadata in metashare "format", you have to offer them in DC thus you do some mapping - a dissemination crosswalk. Ingestion (or submission) crosswalk does the mapping other way around - you receive metadata in DC (eg. through harvesting) and want to save them in metashare "format".

Dublin Core

In DSpace you can meet element sets by Dublin Core Metadata Initiative - DCMI - in two places. You'll encounter DCMES and DCTERMS

DCMES (or *unqualified dc* or just *dc*) contains (and defines/describes) 15 elements for resource description, none of them is mandatory, some can be repeated (eg. more creators). This is the "minimal" format we must support in OAI-PMH. The categories

(elements) are very coarse so by mapping to *DC* we usually "throw" away a lot of information we have about our data. But, on the other hand, everyone, even people outside our "partner projects", can somehow interpret/use our data.

DCTERMS is another set (another namespace) but it contains the same 15 elements as DCMES and their refinements - together aprox. 50 elements. There are eg. these elements: date, dateAccepted, dateCopyrighted... DSpace uses this set (or some modification of it - it refers to *dcmi-terms* but the sets are not exactly the same) when you submit an item or display metadata of an item through "show full item record".

While browsing you see eg. dc.title.alternative (alternative is refinement of title in dcterms), but for OAI-PMH the output is just dc:title.

Our metadata in dc: https://ufal-point.mff.cuni.cz/oai/request?verb=ListRecords&metadataPrefix=oai_dc

CMDI

The idea behind CMDI (or Component MetaData Infrastructure) is that one metadata schema can't "fit" a large community. The elements might be too detailed for one and still to coarse for others, subcommunities might prefer to use different names for same thing etc.

So the solution is to let the users/organizations create own schemas but with ensured "semantic interoperability". Schema/profile is created from components - sets of metadata elements and other components. The components and profiles are stored in component registry and thus can be shared and reused. User can create new components, but the elements used must be linked to a database of atomic concepts (**data categories**). This then allows to "see" that for example 'a noun' and 'a substantive' is the same concept and for example search for tools/articles/... about 'nouns' can also refer to 'substantives'. On such data category registry is Isocat and the "purl" identifiers maintained by dcmi are also accepted.

The component registry was populated by components/profiles based on some of the element sets/schemata widely used (eg. IMDI, OLAC...). To allow a quick transfer for organizations (data providers) using these. So there is a dcmi-terms profile already prepared; it seems natural to be using this as long as we have only few items with metashare md, or while there is no profile available for metashare (they plan to do it in future (???)).

The OAI-PMH cmdi crosswalk is mapping the "DSpace dc" to dcterms, for the metashare elements some mapping is done when inserting an item; the email and validation is not mapped, the rest as follows:

```
dc.contributor.advisor = contributor
```

```
dc.contributor.author = creator
dc.contributor.editor = contributor
dc.contributor.illustrator = contributor
dc.contributor.other = contributor
dc.contributor = contributor
dc.coverage.spatial = spatial
dc.coverage.temporal = temporal
dc.creator = creator
dc.date.accessioned = date
dc.date.available = available
dc.date.copyright = dateCopyrighted
dc.date.created = created
dc.date.issued = issued
dc.date.submitted = dateSubmitted
dc.date.updated = date
dc.date = date
dc.description.abstract = abstract
dc.description.provenance = provenance
dc.description.sponsorship = description
dc.description.statementofresponsibility = description
dc.description.tableofcontents = tableOfContents
dc.description.uri = description
dc.description.version = description
dc.description = description
dc.format.extent = extent
dc.format.medium = medium
dc.format.mimetype = format
dc.format = format
dc.identifier.citation = bibliographicCitation
dc.identifier.govdoc = identifier
dc.identifier.isbn = identifier
dc.identifier.ismn = identifier
dc.identifier.issn = identifier
dc.identifier.other = identifier
dc.identifier.sici = identifier
dc.identifier.slug = identifier
dc.identifier.uri = identifier
dc.identifier = identifier
dc.language.iso = language
dc.language.rfc3066 = language
dc.language = language
dc.publisher = publisher
dc.relation.haspart = hasPart
dc.relation.hasversion = hasVersion
dc.relation.isbasedon = relation
dc.relation.isformatof = isFormatOf
dc.relation.ispartof = isPartOf
dc.relation.ispartofseries = relation
dc.relation.isreferencedby = isReferencedBy
dc.relation.isreplacedby = isReplacedBy
dc.relation.isversionof = isVersionOf
dc.relation.replaces = replaces
dc.relation.requires = requires
dc.relation.uri = relation
dc.relation = relation
dc.rights.holder = rightsHolder
dc.rights.uri = rights
dc.rights = rights
dc.source.uri = source
```

```
dc.source = source
dc.subject.classification = subject
dc.subject.ddc = subject
dc.subject.lcc = subject
dc.subject.lcsh = subject
dc.subject.mesh = subject
dc.subject.other = subject
dc.subject = subject
dc.title.alternative = alternative
dc.title = title
dc.type = type
metashare.ResourceInfo#ContentInfo.mediaType = type
metashare.ResourceInfo#DistributionInfo.availability = rights
metashare.ResourceInfo#DistributionInfo#LicenseInfo.distributionAccessMedium =
medium
metashare.ResourceInfo#DistributionInfo#LicenseInfo.restrictionsOfUse = rights
metashare.ResourceInfo#ResourceCreationInfo#FundingInfo#ProjectInfo.fundingType =
description
metashare.ResourceInfo#ResourceCreationInfo#FundingInfo#ProjectInfo.projectName =
description
metashare.ResourceInfo#TextInfo#SizeInfo.* = extent
metashare.ResourceInfo#ContactInfo#PersonInfo#OrganizationInfo.organizationName =
contributor
```

To make a proper CMD file we also have to add the administrative metadata ("headers"):

- MdCreator is omitted as it has minOccurs=0; but it can be set to something like "UFALCmdiCrosswalk" (name of script can be used)
- MdCreationDate is set to item.getLastModified()
- MdSelfLink - should be set to "the URL or PID of this file"; but we actually don't have any real CMD file so it is set to url that will return CMDI metadata for this particular item. Ie. a link to OAI app with prefilled query (verb, identifier, metadataprefix).
- MdProfile - URL of profile's XSD in the component registry.
- MdCollectionDisplayName - item.getOwningCollection(). Eg. "UFAL - Published Data". According to FAQ "an (optional but recommended) plain text indication to which collection this file belongs. Used for the Collection facet in the VLO".
- Resources - section, containing links to:
    - external files (e.g. an annotation file or a sound recording) and/or other CMDI metadata files (to build hierarchies)
    - The only attached resource, referenced through "hdl.handle.net + item.getHandle()", is the item's "webpage" in DSpace.
- JournalFileProxyList(links to file(s?) tracking the changes of a resource) and ResourceRelationList are left empty.

Updating the schema in new installs

The dspace ant build file contains a job to populate the metadata tables. It does so by running MetadataImporter on xml file with described schema. If the schema should change (eg. new items) this file needs to be changed. That can be done either

manually or you can update the schema in any running instance (through xmlui) and use MetadataExporter to create this file.

Run the following from [installation]/config

In 1.8 it is simmilar (-f contains the path to created file; -s name of the schema in dspace instance you are exporting from)

```
java -Ddspace.configuration=./dspace.cfg -cp ../lib/dspace-api-1.8.2.jar:../lib/*
org.dspace.administer.MetadataExporter -f [git-checkout-
dir]/sources/dspace/config/registries/metashareSchema.xml -s metashare
```

Quality

We should try to provide as consistent metadata as possible. That means from time to time checking the values and if we come to conclusion there are multiple values for one entity (author, affiliated organization, etc.) we should replace it with just one… Since we don't display all the metadata in item browse, our main feedback are currently the organizations that harvest our repository (eg. VLO). One "drawback" with this approach is we actually see our metadata after certain mapping (eg. our publisher and affiliated institution are both organization of some kind, so there is nothing "preventing" from grouping this under one facet called organization).

So identifying from which field the value came might be somewhat tedious. In VLO you can display the source metadata, the dc metadata should be clear and most of the (X)Paths in the resourceInfo component should resemble our database entries. But some values are generated automatically, database uses some "fake" fields (eg. detailedType) which are named differently based on the type of resource and some entries resemble the structure of the first version of schema which might have changed (something moved to other component). The easiest way to identify the field is thus probably following the links to the item display; going to full display and searching the value through browsers find. Or connecting to psql and doing the search there, which is currently probably the best way…since you'll want to find other items, or do some mass replace.

Other thing is that from time to time we might not like under which facet certain value appears, there's really not much we can do. We should check the semantics of the field, to see if it really matches our usage (if we are using the right fields for the right values). Some fields might have broader semantics then we choose to use (eg. author = person || institution, we use it just for people; it can make sense when people and institutions are under one facet). The grouping of different fields under one is usually application specific.