Short introduction to the Frbrization tool

Trond.Aalberg@idi.ntnu.no

About the tool

The Frbrization tool is an XSLT-based solution for transforming MARC-records to FRBR-records. The tool supports the definition of rules for interpreting the structure and information in MARC-based records. These rules are then used to automatically create an XSLT conversion stylesheet that can used to transform collections of MARC records to a collection of FRBR records. The main motivation for this approach is to support different MARC formats and different cataloguing practice in a dynamic and flexible way. Different MARC formats and the way bibliographic information is coded can be very different and these differences have significant impact on the programmatic interpretation of records in the context of the FRBR model. Additionally, there is a need for a highly iterative development process to explore and test possible solutions. This is supported by the use of a scheme for expressing rules which then are used to automatically create an XSLT conversion stylesheet.

Input records

Records used as input to the Frbrization tool needs to be in a generic MARCXML/MarcXchange format. Records have to be in the following format. Namespaces are for pragmatic reasons not supported by the tool and have to be removed from the source files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT collection ((record+))>
<!ELEMENT record ((leader?, controlfield*, datafield*))>
<!ELEMENT leader (#PCDATA)>
<!ELEMENT controlfield (#PCDATA)>
<!ELEMENT datafield ((subfield+))>
<!ELEMENT subfield (#PCDATA)>
<!ATTLIST subfield
        code CDATA #REQUIRED
>
<!ATTLIST record
        id CDATA #REQUIRED
>
<!ATTLIST datafield
        ind1 CDATA #IMPLIED
        ind2 CDATA #IMPLIED
        tag CDATA #REQUIRED
>
<!ATTLIST controlfield
        tag CDATA #REQUIRED
>
<!--Records may contain other elements and attributes: e.g. additional indicators and identifiers-->
```

Output format

Records produced by the Frbrization process are in the following format (simplified dtd):

```
<?xml version="1.0" encoding="UTF-8"?>
<!--A simplified DTD for the records produced by the frbrization tool-->
<!ELEMENT collection ((record*))>
<!ELEMENT record ((datafield* | controlfield* | relationship*))>
<!ELEMENT controlfield (#PCDATA)>
<!ELEMENT datafield ((subfield*))>
<!ELEMENT subfield (#PCDATA)>
<!ELEMENT relationship EMPTY>
<!ATTLIST record
        id CDATA #REQUIRED
        label CDATA #REQUIRED
        type CDATA #REQUIRED>
<!ATTLIST controlfield
        tag CDATA #REQUIRED >
<!ATTLIST datafield
        tag CDATA #REQUIRED >
<!ATTLIST subfield
        code CDATA #REQUIRED >
<!ATTLIST relationship
        href CDATA #REQUIRED
        label CDATA #REQUIRED
        type CDATA #REQUIRED >
```

Output records

The tool creates a single record for each unique entity that is found in the input MARC records. Relationships between entities are expressed using relationship elements which basically are typed links to other records. Even if a specific person is described in several MARC records (possibly also in different fields) there will be only one record in the final output for this person (assuming that the person is consistently described). The same applies to other entities.

Description of the tool

The Frbrization tool basically consists of:

- An xml-format used to specify the rules for how to interpret MARC records.
- An XSLT-file (make.xslt) that transforms these rules to another XSLT-file that can be used to transform the MARC records.

Additional files/tools are used in the transformation and/or to support the Frbrization project:

- Database to manage the rules:
  Writing and maintaining rules for a transformation is a cumbersome process and it is recommended to use a database to maintain the rules rather than directly editing the xml-file. We have used a Microsoft Access database for this purpose. The data can be exported from MS Access as an xml-dump which then can be used to create the XML-based rule file (based on a rather simple XSLT conversion). This introduces an additional transformation process, but on the other hand it makes the job of creating and maintaining rules a lot easier (Raw XML is usually not very easy to read and edit).
- FRBR types and labels
  The Frbrization tool requires uniquely identified types (for FRBR entities, attributes and relationships). These are listed in the file ***frbr.types.xml*** and are used to associate type identifiers to the entities, attributes and relationships during the frbrization process. This file

can be edited if needed and it is automatically included if you use the access2entities.xsl file. The label associated to a type is merely intended to improve the readability of a record and the numbering of types is ad hoc (based on sections in the FRBR report).

- Additional XSLT-files:
  The ***make.xslt*** file imports from two additional xslt-files: ***basic.xslt*** and ***merge.xslt***.
  It is additionally possible to hand-code user-defined templates for processing specific entities (in the usertemplates.xlt file), but this requires that the template works in more or less the same way as the automatically created templates. The merge.xslt file is convenient if you later need to merge different files containing frbr records etc.
- MD5 utility:
  Entity identifiers are created based on selected data items from the records (defined in the rules). This information is used to produce an MD5-based identifier that is created in the frbrization process. This feature is implemented as a java-based extension function in the ***MD5Generator.jar*** file.

How the records are processed by the tool

Input records are processed by the tools in a chain of steps that perform different transformations of the records:

- The first step is to find entities in each MARC record using the templates that are described in the rules. At this stage we process each record individually. The datafields and subfields related to each entity are simply copied to the FRBR records created for a particular MARC record (based on the datafields and subfields listed as in the rules file). Each record is at this stage identified by an internal identifier and relationships are based on these internal identifiers.
- The next step is to transform all internal identifiers to description-based identifiers. The rules file includes a specification of what datafield/subfield values (and the order of these) that should be used to identify an entity and this is used to create an identifier string (e.g. based on the name and date for a person). Creating string based identifiers includes conversion to lowercase and removal of whitespaces (and other special characters). All internal identifiers are then replaced with the md5-version of the description based identifier.
- The final main part of the process is to merge equivalent records. From the previous steps we may have multiple records for the same persons or works but we only want one record for each unique entity. Merging is simply a process of combining the union of unique datafields and relationships in the set of records that have the same identifier. Datafields that differ in any way (indicators, subfields used, any difference in text content) are considered as different and will all be included in the final record.

Using the tool

Step 1 (Using the database):

- Use an Access database to create and edit entries for entities, relationships and attributes. The sample database contains all the tables that are needed.
- Export the database to xml using the build-in support for dumping databases to xml:
  Run the access2rules stylesheet to produce the final file that describes the rules in the approriate format.The resulting file can be named e.g. "***rules.xml***"

Step 1 (Alternative: without using the database)

- Create a rules.xml file by hand. Look at the example file (and the dtd) to learn how this file should look like (or adapt an existing one).

Step 2

- Use the "***make.xls***" stylesheet file to create the conversion file from the "***rules.xml***" file. The resulting file can e.g. be named "***conversion.xsl***".

Step 3

- Perform the actual conversion by applying the "***conversion.xsl***" file on one or more files containg MARC records. The conversion is based on XSLT 2.0 and XPATH 2.0. The Saxon XSLT processor is one of the few that supports XSLT 2.0 and is probably the best processor to use (available from sourceforge). Please note that you need to include a jar-file for the XSLT md5 extension functions in your CLASSPATH. The use of this extension function also requires that you use a java-based processor (or a processor that is able to load java-based extension functions). Due to changes in the way external functions are supported in the Saxon 9.2 Home edition, you should use an earlier version (9.0 or 9.1) of this software (unless you purchase a license for the Professional or Enterpise edition of the 9.2 version).

Step 4

- Evaluate your records to determine if the transformation was according to the intention. Typically you would start out by defining a few templates, run the conversion, inspect the result, correct errors. Adding templates one by one makes it easier to inspect the results and correct possible errors caused by incorrect XPATHs etc. Start e.g. with templates for the different person fields, and continue with work- templates etc.

Sample commands

In Access:

- External Data -> Export -> Saved Exports (rememeber to edit the path and the filename) Filename can be anything you like, but in the following we will assume that you dump it to a file called access.dump.xml. Remember to include all tables in the export.

Transformation steps using Saxon:

a) java net.sf.saxon.Transform -s:***access.dump.xsl*** -o: ***rules.xml*** -xsl:***access2rules.xsl***
b) java net.sf.saxon.Transform -s: ***rules.xml*** -o:***conversion.xsl*** -xsl:***make.xsl***
c) java net.sf.saxon.Transform -s:***marcrecord.xml*** -o:***frbrrecord.xml*** -xsl:***xxx.conversion.xsl***

Writing templates

The tool is intended for those who want to experiment with different interpretations of MARC records. Many adaptations of MARC are in use and even catalogues that use the same MARC format uses the fields differently. The process of doing a conversion using the tool is rather straight forward, but the different interpretation rules have to be coded using criteria specified as tags, codes, XPATH-conditions etc. Setting up and testing the rules can be difficult in the beginning, but once you learn how to use the database or the xml-format, it should be not be very difficult.

Rules for entities are specified as “entity templates” and each of these will end up as an XSLT template in the final conversion file. An entity template is generally a receipt for how to identify an entity (and all entity-templates that you create define what entities you will identify by the tool). This is best explained using an example: The 100-field in a MARC 21 record is used to describe the author of a work. Thus, we know that if there is a 100-field, we should create a FRBR-entity record for this person. In the database (or in the XML-rule file) you would then have to create a template for persons identified from the 100 field.

Person creators can additionally be found in other fields as well such as 700 Added entry fields (MARC 21). To solve this you need to create another template for persons identified by the 100 field, and another template for persons identified by the 700 field. Correspondingly you would need templates for corporate bodies in the 110, 111, 710, 711 fields. Many different templates are needed to be able to manage the diversity of entities that can be recognized in a record. Some templates require quite intricate rules, such as creating a work entity from a 245-title, if there is no 240-title or 130-title. In this case you need to use the 245 tag and write an XPATH-expression to test that there is no occurrence of 240 and/or 130 fields.

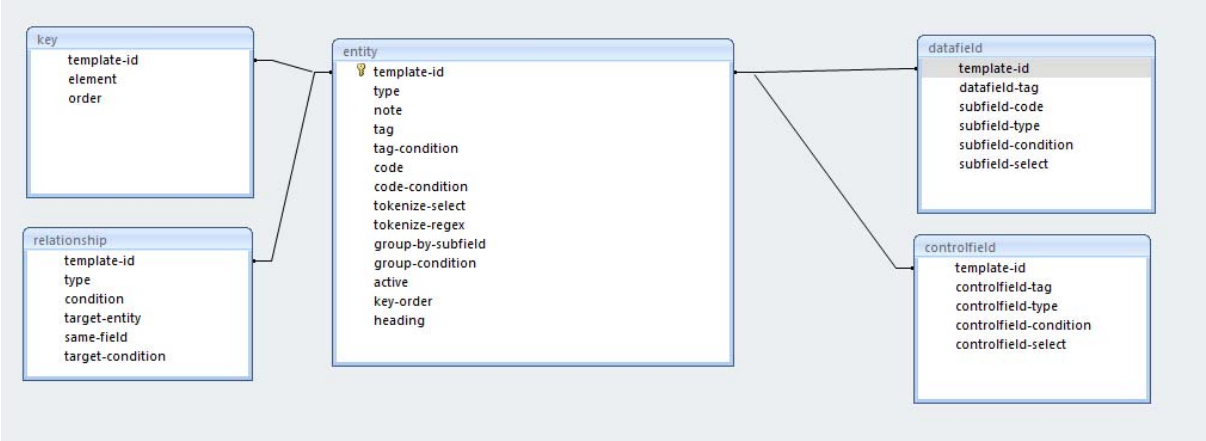
The next part of the job is typically to define what data fields and subfields that should be mapped to the entity. This part is basically to list the tags and subordinate codes that should be included in the entity record.

Specifying relationships is typically the most difficult job. Relationships are described by type and the target entity that a relationship links to. To cover all possible relationships that may exist there is a need to list all combinations of relationship types and template-types. In addition you will have to specify the additional conditions that are needed to set up the correct type of relationship. A person identified in the 100-field may typically be related to works identified by a 240 fields. If relator codes are used to specify the role of the person you will have to write a test for specific relator code values to determine if the relationship should be to the 240 work. Some relator codes indicate that the relationship is to the expression and to support this you need to create another relationship that links to the 240 expression with the corresponding test.

Finally, you will need to list the values that should be used to generate an identifier for the entity. Descriptive identifiers are used to merge duplicate records and the components of the identifier are various values found in an entity record (in ordered sequence). Two records with equal identifiers will be merged to one record (without duplicating the data field contents).

The Access database and the rules.xml file

The following description explains the various tables and fields in the Access database and the corresponding entries in the rules.xml file. The green text documents the corresponding entry in the rules format (that can be automatically produced from the Access database). The best way to get an idea of the use of these fields is to look at the final conversion file that is produces and spot where the various parts are included.



The "entity" table

template-id:

- string that uniquely identifies a template (good idea to use an informative string)
- //templates/entity@id

type:

- an identifier for the frbr type of the resulting entity
- //templates/entity@type

note:

- short description
- //templates/entity/note

tag:

- the field tag (this tag will be used to automatically create a for-each loop for iterating over the fields with this tag in a record)
- //templates/entity/anchor@tag

tag-condition

- XPATH condition that can be used to specify conditions associated to the tag-loop
- //templates/entity/anchor@condition

code

- creates a second level for-each loop to iterate over multiple occurrences of a subfield within a datafield
- //templates/entity/anchor/code@code

code-condition

- XPATH condition that can be used to specify conditions for the code-loop
- //templates/entity/anchor/code@condition

tokenize-select

- XPATH expression for selecting text to tokenize (useful for removing prefixes etc).
- //templates/entity/anchor/code/tokenize@filter

tokenize-regex

- XPATH expression selecting tokenizing text-values (e.g. for multiple titles listed in a note field)
- //templates/entity/anchor/code/tokenize@regex

group-by-subfield

- Use in case you need to iterate over groups of repeatable subfields within a datafield. The code you insert here should be the code that starts a group.
- //templates/entity/anchor/group-by@code

group-condtition

- Use for testing if a group is a relevant entity or not
- //templates/entity/anchor/group-by@condition

key-order

- This field enables you to manipulate the order identifiers are created. Typically persons and corporate body identifiers has to be created first and should have the value 1, work identifiers should be created next and should have the number 2, then expression 3 and manifestations 4. Some identifiers are depending on other ones. A work identifier would include the identifier for creator and for this reason the latter must be created first.
- //templates/entity/key@order

active

- Use to include or exclude entity templates. Yes (checked) means that this entity-template will be included in the created rules file, No (unchecked) means that it will not be included. Unchecking an entity will also prevent relationship having that entity as target from being created.
- This field is only related to the conversion of the access dump to the rules.xml file

heading

- This field can contain an XPATH expression for creating additional textual content that is needed for display or identification of the entity. Only needed e.g. when there is a need to add a heading that cannot be found in the final record etc.
- //templates/entity/heading

The datafield table

Generally used to map MARC data fields/sub fields to entity-templates

template-id

- foreign key that associates a datafield rule to an entity template
- //templates/entity@id

datafield-tag

- the datafield that should be associated to the template entity
- //templates/entity/attributes/datafield@tag

subfield-code

- the subfield that should be included (not all subfields are necessarily associated to the same kind of entity)
- //templates/entity/attributes/datafield/subfield@code

subfield-type

- FRBR type identifier (attribute type) – can be blank in case it cannot be mapped
- //templates/entity/attributes/datafield/subfield@type

subfield-condition

- XPATH expression for testing whether to include a subfield or not (should evaluate to true or false)
- //templates/entity/attributes/datafield/subfield@condition

subfield-select

- XPATH expression for selecting textual content within a specific subfield (not in use yet)
- //templates/entity/attributes/datafield/subfield@select

The controlfield table

Generally used to map MARC controlfields to entity-templates

template-id

- foreing key that associates a controlfield rule to an entity_template
- //templates/entity@id

controlfield-tag

- the controlfield that should be associated to the template entity
- //templates/entity/attributes/controlfield@tag

controlfield-type

- FRBR type identifier (attribute type) – can be blank in case it cannot be mapped
- //templates/entity/attributes/controlfield@type

controlfield-condition

- XPATH expression for testing whether to include a controlfield or not (should evaluate to true or false)
- //templates/entity/attributes/controlfield@condition

controlfield-select

- XPATH expression for selecting textual content within a specific subfield (not in use yet)
- //templates/entity/attributes/controlfield@select

The relationship table

Generally used to specify relationship rules

template-id

- foreing key that associates a relationship rule to an entity_template
- //templates/entity@id

type

- Identifier for the FRBR relationship type
- //templates/relationships/relationship@type

condition

- XPATH test for whether to create this relationship or not – (the conversion file will test automatically if the target exist so this is only needed in specific cases)
- //templates/relationships/relationship@cond

inverse

- The inverse type of the relationship: only included to be able to check the database for consistency of two-way relationships (you can create an sql expression for this).
- na

target-entity

- The template-id for the entity that this relationship goes to
- //templates/entity/relationships/relationship/target@entity

target-condition

- XPATH condition testing for wheter a relationship should be created or not
- //templates/entity/relationships/relationship/target@cond

same-field

- This field is used to limit relationships to entities identified from the same datafield: typically used for fields such as added entries 700 (MARC21) where there is a title that identifies a work and a name that identifies the creator. By checking this field the appropriate conditions are automatically created to link only the work and author in this particular datafield (and not all authors and all works in all 700 fields). Basically included because it is more convenient to do it this way than hardcoding such conditions using XPATH expressions.
- //templates/entity/relationships/relationship/target@same_field (use values 0 and 1)

note

- for notes
- not included in the rules.xml file

The keys table

This table is used to specify the various data elements that should be used to uniquely identify an entity (and the order these elements should be arranged in the identifier).

template-id

- foreing key that associates a key rule to an entity_template
- //templates/entity@id

element

- XPATH expression for selecting a text value that should be included in the identifier string. This should be a single value selected from the record, but can also be a literal etc. These identifiers are created after the first step of the Frbrization process and can for that reason include relationships and other fields created during the process.
- //templates/entity/key/element

order

- this field is used to specify the arrangement of key elements, 1 is the first element, 2 the second element etc. This field is only found in the database and is used to make sure that the ordering of key-elements in the rules.xml file is correct.
- na

Additional elements not found in the database (but included in the rules.xml file)

- keyfilter.xml contains an XPATH expression that is used to clean up identifier strings by converting to lower-case, removing punctuation and white-spaces etc.
- //templates/keyfilter

- The rules-file has to include a list of the types used. The type identifier is a unique string for a type and the label is any kind of textual description that describes the type (mainly intended to make the records more readable). The element content is just a placeholder for documenting the type internally.
- //templates/types/type
- //templates/types/type@id
- //templates/types/type@label<

What are these fields used for?

The various fields in the database (and correspondingly in the xml-based rule file that is created), are used as in a parameterized creation of XSLT-templates. These templates are what finally is used to do the actual conversion.

The following rule simply states that we want a frbr work-record created for each 700$t field and that this record should have a “is created by” relationship to the person-record that is created from

the same field (unless there is a relator code – in this collection the rule is that relator codes only are used if the relationship is not “author”).

```
<entity id="MARC21_700_Work" type="4.2">
    <note>Work identified by the occurence of marc field 700</note>
    <anchor tag="700">
       <code code="t"/>
    </anchor>
    <attributes>
       <datafield tag="700">
          <subfield code="t"/>
       </datafield>
    </attributes>
    <key order="2">
       <element>frbr:relationship[@type = ('5.2.2.1.R')]/@href</element>
       <element>datafield[@tag = '700']/subfield[@code = 't']</element>
       <element>'#'</element>
    </key>
    <relationships>
       <relationship type="5.2.2.4.R">
          <target entity="MARC21_700_Person"
                  condition="not(exists(subfield[@code = '4']))"
                  same-field="true"/>
       </relationship>
    </relationships>
 </entity>
```

This XML-fragment will result in the following code in the generated conversion file (xslt).

The anchor element (and attributes and sub-elements of this) is used to create loop-statement(s) for creating records as in the following simplified code:

```
<xsl:for-each select="node()[@tag='700']">
    <xsl:for-each select="node()[@code='t']">
        <frbr:record>
```

If you use the tokenize field or the group-by field the nested loops will be more extensive. The tokenize field will add an “analyze-string” xslt-expression and the group-by field will add a group-by expression.

The attributes element is used to create the code that copies the specified controlfields, datafields and subfields to the record (this code is simplified too…):

```
<xsl:for-each select="$record/datafield[@tag='700']">
    <xsl:copy>
        <xsl:call-template name="copy-attributes"/>
        <xsl:for-each select="subfield[@code = ('t')]">
            <xsl:copy-of select="."/>
        </xsl:for-each>
    </xsl:copy>
</xsl:for-each>
```

The relationship element is used to create relationship-elements. This code is a bit more complicated because it actually uses the anchor information of the target entity to create a loop-statement(s) for what to point to.

```
<xsl:for-each select="$record/node()[@tag='700']">
   <xsl:if test="(not(exists(subfield[@code = '4']))) and ($target_field = $this_field)">
      <frbr:relationship type="5.2.2.4.R" target_type="4.6">
         <xsl:attribute name="href" select=…….>
      </frbr:relationship>
   </xsl:if>
</xsl:for-each>
```

Parameters in the conversion file

The final conversion file contains a number of boolean parameters that can be used to change the information that is included in the final frbr records. Parameters have a default value but specific values can be passed to the xslt processor (depending on what xslt processor you use).

- debug (default=false): If set to true the records will include some fields that can be useful for debugging purpose.
- include_labels (default=false): If set to true the records will include frbr type labels in addition to the type id. Labels for entities, relationships and attributes adds up to a lot of data, but on the other they improve the readability of the records.
- include_anchorvalues (default=false): adds information about the unique values that are used to create a frbr record in a marc record
- include_templateinfo (default=false): adds information about what template that has been used to create a record
- include_sourceinfo (default=false): adds information about what MARC record the frbr record is created from
- include_keyvalues (default=false): include the concatenated values that are used to create the record key
- include_internal_key (default=false): include the internal key that is used in the first stage of creating a record: tag + code and additional information
- include_counters (default=false): include some statistical information
- md5_identifiers (default=true): set to false if you want text-based identifiers rather than the calculated md5 sum of the text-based identifier
- merge (default=true): set to false if you do not want frbr records to be merged
- include_id_as_element (default=false): include identifiers as element in addition to attribute

Additional information

There are a number of variables defined in the conversion file that can be used in the XPATH-expressions. Some are context-dependant (meaning that they only can be used in some fields, others can be used in all fields).

NB! The best way to learn about the use of the use of the different fields is to inspect the resulting conversion file.

NB!! The conversion file contains a number of parameters that can be turned on/of to remove/create additional information in the resulting frbr records.

Working with many large files

While experimenting with frbrization it is often sufficient to process only one single file of MARC-records. The frbrization tool only works on a single file at a time, and the maximum size of this file will be limited by the amount of memory you allocate to the xslt-processor. Java-based processors typically use a lot of memory and to be able to process a 10-20 Mb file you will typically need a Java heap space >= 1024Mb (-Xmx1025M) . Working with many smaller files is generally more efficient than having fewer but larger files.

To be able to process a complete catalogue you will need to use many files, frbrise the files independently and then merge the files into a database. Recommended file size is app 5-10 Mb. After the frbrization process the frbr records have identifiers that can be used for merging new records into a database. Records in different files that have the same identifiers should be merged together (combined by removing as much as possible of duplicate information). The merge template that is found in the merge.xsl file can be used for this purpose. This is the same template that is used internally in the frbrization process.