Mapping Memory Manager (3M)

Instance and Label Generator Rule Builder

Maria Theodoridou
(maria@ics.forth.gr)
Yannis Marketakis
(marketak@ics.forth.gr)
Nikos Minadakis
(minadakn@ics.forth.gr)
Giorgos Samaritakis
(samarita@ics.forth.gr)
Konstantina Konsolaki
(konsolak@ics.forth.gr)

Draft
Version 1.0

3MEditor version 3.4
X3ML version 1.9.2

1 Instance and Label Generator Rule Builder

During the first step of the mapping definition process, provider schema experts in cooperation with target schema experts exploit the ***Schema Matcher*** component (*Matching Table Tab* in 3M Editor) to define a schema matching which is documented in a ***Schema Matching Definition***.

The next step is the specification of the instance generation rules that define the URI generation policy for each target class. This task is accomplished by the ***Instance and Label Generator Rule Builder*** (*Generators Tab* in 3M Editor), which complements the Schema Matching Definition with the instance generation policies, producing the ***Mapping Definition*** (*X3ML file*). A description of the X3ML language is available at https://github.com/isl/x3ml/blob/master/docs/x3ml-language.md

The user interface of the ***Instance and Label Generator Rule Builder*** is similar to the Schema Matcher component. However, users can only edit details about the instance and label generators.

The component uses rules (templates) for specifying how a URI or a label will be created and they can be exploited throughout the mapping. The generator rules are defined separately in an XML file and linked to the actual mapping in order to be used by the generator editor for producing the actual identifier or label. In order to link a mapping with the generator rules the user must load the generator policy file that contains the rules and has been edited offline in the Info tab in 3M Editor.

The rule-based URI generation follows the RFC 6570 standard and the rules are specified in XML in the following form:

```xml
<generator_policy>
        <generator name="MyGenerator" prefix="myNS">
                <pattern>{arg1}/{arg2}</pattern>
        </generator>
</generator_policy>
```

Sample input:

```
myNS = http://www.mySpace.gr/
arg1=Object
arg2=12345
```

Generated URI:

```
http://ww.mySpace.gr/Object/12345
```

Whenever the required URIs or labels cannot be generated by the default generators, or simple rules as the above, it is possible to implement a custom generator in the form of a class implementing the InstanceGenerator component interfaces.

In the following sections we present the existing custom generators already built-in the X3ML framework, however more user-defined custom generators can be implemented.

A sample generator file is available in Appendix A.

Each target entity must have only one instance generator responsible for constructing an identifier for a resource, and any number of label generators. Generators use their names as identifiers, therefore the same generator can be exploited for many different target entities.

Each instance generator should be assigned a name, and associated with a list of arguments. The arguments are being exploited to provide the text segments that are required for constructing an identifier. Each argument has a name, which should be unique in the context of the generator, and a type and value pair. The type determines how the value of the argument is evaluated and can be defined either in the definition of the generator policy (in the generator policy file) or during the declaration of the generator (in the X3ML mappings); there are 3 different options for the type:

- **constant.** In this case the value of the argument should be used as it is defined. This type is used when we want to assign constant values to the generated identifiers.
- **xpath.** In this case the value of the argument is an XPATH expression, that should be evaluated with the given input. The result of the XPATH expression will be used for the generation of the identifier. This type is used when it is required to exploit data from the input file in the generated identifiers.
- **position.** In this case the value of the argument is ignored and the index position of the corresponding source node, within its context, is being used.

Label generators follow the same logic as instance generators but they also exploit one extra optional argument for the language of the generated value. If it is empty then it is implied that the generated value will not have it (i.e., in the case of number values). The X3ML engine provides default implementations for producing literal values in the form of rdfs:label and skos:prefLabel and constant values.

1.1 Custom Instance Generators

As described already, instance generators are templates that the user can define according to the needs of the data that are being transformed. However there are instance generators that are more complex and cannot be defined through a template. These are the Custom Instance Generators and are implemented via code. X3ML allows the incorporation of new custom instance generators. The Custom Instance Generators that are currently implemented are:

1.1.1 UUID

1.1.1.1 Description

This function creates a UUID. It does not need to be defined in the generator policy file. It is built in the X3ML engine. The 3M editor assigns by default the UUID generator to all classes allowing the user to perform a fast transformation checking without defining detailed generators.

1.1.1.2 Example of Use

Definition in the Generator Policy file:

```
It does not need to be defined in the generator policy file
```

Sample XML input:

```
<dataroot>
    <COIN>
        <ID>123</ID>
    </COIN>
    <COIN>
        <ID>456</ID>
    </COIN>
</dataroot>
```

Sample X3ML mapping file:

```
<x3ml version="1.0" source_type="xpath">
    <namespaces>
        <namespace prefix="crm" uri="http://www.cidoc-crm.org/cidoc-crm/"/>
        <namespace prefix="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <namespace prefix="rdfs" uri="http://www.w3.org/2000/01/rdf-schema#"/>
    </namespaces>
    <mappings>
        <mapping>
        <domain>
                <source_node>//COIN</source_node>
                <target_node>
                <entity>
                        <type>crm:E22_Man-Made_Object</type>
                        <instance_generator name="UUID"/>
                </entity>
                </target_node>
        </domain>
        </mapping>
    </mappings>
</x3ml>
```

RDF/XML Output:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <crm:E22_Man-Made_Object rdf:about="urn:uuid:c99d5c2b-db59-4f20-b5c6-2051b5795e02"/>
  <crm:E22_Man-Made_Object rdf:about="urn:uuid:c99d5c2b-db59-4f20-b5c6-2051b5795e03"/>
</rdf:RDF>
```

1.1.2 Literal

This function creates a Literal value (i.e. a text node). It does not need to be defined in the generator policy file. It is built in the X3ML engine.

Sample XML input:

```
<dataroot>
    <COIN>
        <DESCRIPTION>an ancient coin</DESCRIPTION>
    </COIN>
</dataroot>
```

Sample X3ML mapping file:

```
<x3ml version="1.0" source_type="xpath">
<mapping>
…
        <path>
                <source_relation>
                        <relation>DESCRIPTION</relation>
                </source_relation>
                <target_relation>
                        <relationship>crm: P3_has_note</relationship>
                </target_relation>
        </path>
        <range>
                <source_node>DESCRIPTION</source_node>
                <target_node>
                        <entity>
                                <type>crm:E62_String</type>
                                <instance_generator name="Literal">
                                        <arg name="text">text()</arg>
                                        <arg name="language" type="constant">en</arg>
                                        </instance_generator>
                        </entity>
                </target_node>
        </range>
…
</mapping>
</x3ml>
```

RDF/XML Output:

```
<rdf:RDF
…
  <crm:P2_has_note xml:lang="en">an ancient coin</crm:P2_has_note>
</rdf:RDF>
```

1.1.3 URIorUUID

1.1.3.1 Description

The URIorUUID generator is responsible for generating a URI or a UUID. It takes as input a single parameter (with name "text") which can be assigned a source XPATH value, or a constant. If the assigned value is a valid URI or URN, it is copied to the output. Otherwise, if the assigned value is not a valid URI or URN, the URIorUUID generator generates a UUID.
The URIorUUID generator is also applicable in cases where the corresponding XPATH value does not exist. For example, given the input <FOO><BAR/></FOO>, we want to generate a value using the XPATH expression BAR/text(). The value does not exist for the given XPATH expression, however the engine will generate a UUID value for the given entity (based on the fact that the source_node exists (i.e. FOO) and it will NOT trigger an error.

1.1.3.2 Example of Use

Definition in the Generator Policy file:

```
<generator_policy>
        <generator name="URIorUUID">
                <custom generatorClass="gr.forth.URIorUUID">
                        <set-arg name="text"/>
                </custom>
        </generator>
</generator_policy>
```

Sample XML input:

```
<dataroot>
    <COIN>
        <ID>http://example/C1</ID>
    </COIN>
    <COIN>
        <ID>2</ID>
    </COIN>
</dataroot>
```

Sample X3ML mapping file:

```
<x3ml version="1.0" source_type="xpath">
    <namespaces>
        <namespace prefix="crm" uri="http://www.cidoc-crm.org/cidoc-crm/"/>
        <namespace prefix="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <namespace prefix="rdfs" uri="http://www.w3.org/2000/01/rdf-schema#"/>
    </namespaces>
    <mappings>
        <mapping>
        <domain>
                <source_node>//COIN</source_node>
                <target_node>
                <entity>
                        <type>crm:E22_Man-Made_Object</type>
                        <instance_generator name="URIorUUID">
                                <arg name="text">ID/text()</arg>
                        </instance_generator>
                </entity>
                </target_node>
        </domain>
        </mapping>
    </mappings>
</x3ml>
```

RDF/XML Output:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <crm:E22_Man-Made_Object rdf:about="http://example/C1"/>
  <crm:E22_Man-Made_Object rdf:about="urn:uuid:c99d5c2b-db59-4f20-b5c6-2051b5795e03"/>
</rdf:RDF>
```

1.1.4 UriExistingOrNew

1.1.4.1 Description

The UriExistingOrNew generator works in a similar manner with URIorUUID. More specifically it will reuse an existing URI from the input and if the URI does not exist then it will construct one using a specific pattern.

The generator has the following mandatory arguments:

- **uri**: It is used for declaring the XPATH expression where a potential URI can be found in the input
- **text#**: It is used for expressing which parts of the input will be used for constructing a new URI (if one does not exist). The generator can accept many arguments of this type. The assumption is that the first one will have the name text1, the second one will have the name text2, etc.
- **uri_separator#**: It is used for expressing the characters that will be used between text arguments. The generator can accept many arguments of this type. The assumption is that the first one will have the name uri_separator1, the second one will have the name uri_separator2, etc.

It becomes evident that uri is used for retrieving the value of an existing URI, while text# and uri_separator# are used for constructing a new one. The latter are combined using the following scheme:
namespace_prefix+text1+uri_separator1+text2+uri_separator2+...

Important: The number of text arguments should be equal to the number of uri_separator_arguments.

A new URI will be generated as soon as there is not an existing valid URI.

The namespace to be used with all the new URIs can be declared in the generation-policy file, within the declaration of the generator (under the attribute prefix).

1.1.4.2 Example of Use

Sample XML input:

```xml
<dataroot>
    <COIN>
        <ID>http://ID-100</ID>
        <COUNTRY_ID>C1</COUNTRY_ID>
    </COIN>
    <COIN>
        <ID>200</ID>
        <COUNTRY_ID>C2</COUNTRY_ID>
    </COIN>
    <COIN>
        <ID>300</ID>
        <COUNTRY_ID>C3</COUNTRY_ID>
    </COIN>
</dataroot>
```

Sample Generator Policy file:

```
<generator_policy>
<generator name="UriExistingOrNew" prefix="ex">
        <custom generatorClass="gr.forth.UriExistingOrNew">
                <set-arg name="uri" type="xpath"/>
                <set-arg name="text1"/>
                <set-arg name="uri_separator1" type="constant"/>
                <set-arg name="text2"/>
                <set-arg name="uri_separator2" type="constant"/>
        </custom>
</generator>
</generator_policy>
```

Sample X3ML mapping file:

```
<x3ml version="1.0" source_type="xpath">
    <namespaces>
        <namespace prefix="crm" uri="http://www.cidoc-crm.org/cidoc-crm/"/>
        <namespace prefix="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <namespace prefix="rdfs" uri="http://www.w3.org/2000/01/rdf-schema#"/>
         <namespace prefix="ex" uri="http://example/"/>
    </namespaces>
    <mappings>
        <mapping>
            <domain>
                <source_node>//COIN</source_node>
                <target_node>
                    <entity>
                        <type>crm:E22_Man-Made_Object</type>
                        <instance_generator name="UriExistingOrNew">
                                 <arg name="uri" type="xpath">ID/text()</arg>
                                 <arg name="text1" type="xpath">ID/text()</arg>
                                 <arg name="uri_separator1" type="constant">/</arg>
                                 <arg name="text2" type="xpath">COUNTRY_ID/text()</arg>
                                 <arg name="uri_separator2" type="constant"></arg>
                         </instance_generator>
                    </entity>
                </target_node>
            </domain>
        </mapping>
    </mappings>
</x3ml>
```

RDF Output:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex="http://example/"
    xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <crm:E22_Man-Made_Object rdf:about="http://example/300/C3"/>
  <crm:E22_Man-Made_Object rdf:about="http://ID-100"/>
  <crm:E22_Man-Made_Object rdf:about="http://example/200/C2"/>
</rdf:RDF>
```

1.1.5 ConcatMultipleTerms

1.1.5.1 Description

This generator function supports the construction of new values by merging multiple similar elements (elements that have a common name).
Apart from the similar entries, the generator defines the delimiter that will be used for merging the values. The generator can be used for creating both URIs and literal values.

The generator takes at least three arguments:

- **prefix**: it is a constant value and denotes the prefix that should be used before concatenating the terms from the input. If the prefix value starts with http then the generated value will be a URI, otherwise, if it is empty or anything else it will generate a literal value.
- **delimiter**: the delimiter that will be used within the concatenated terms
- **text#:** an arbitrary number of terms indicating the XPath expressions that will fetch the terms that will be concatenated. This term text should be followed by an integer indicating the execution order for the concatenation; for example the terms under text1 will be concatenated, then the terms under text2, etc.

1.1.5.2 Example of Use

Sample XML input:

```
<COIN>
    <ID country="gr">ID_GR_1</ID>
    <ID country="uk">ID_UK_1</ID>
    <TITLE country="gr">Nomisma1</TITLE>
    <TITLE country="uk">Coin1</TITLE>
</COIN>
<COIN>
    <ID country="gr">ID_GR_2</ID>
    <ID country="uk">ID_UK_2</ID>
</COIN>
```

Sample Generator Policy file:

```
<generator_policy>
<generator name="ConcatMultipleTerms">
        <custom generatorClass="gr.forth.ConcatMultipleTerms">
                <set-arg name="prefix" type="constant"/>
                <set-arg name="delimiter" type="constant"/>
                <set-arg name="text1"/>
                <set-arg name="text2"/>
         </custom>
</generator>
</generator_policy>
```

Sample X3ML mapping file:

```
<x3ml version="1.0" source_type="xpath">
 <namespaces>
        <namespace prefix="crm" uri="http://www.cidoc-crm.org/cidoc-crm/"/>
        <namespace prefix="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <namespace prefix="rdfs" uri="http://www.w3.org/2000/01/rdf-schema#"/>
         <namespace prefix="ex" uri="http://example/"/>
 </namespaces>
 <mappings>
 <mapping>
```

```
    <domain>
        <source_node>//COIN</source_node>
        <target_node>
        <entity>
            <type>crm:E22_Man-Made_Object</type>
              <instance_generator name="ConcatMultipleTerms">
                <arg name="prefix" type="constant">http://www.ics.forth.gr/isl/</arg>
                <arg name="text1" type="xpath">ID/text()</arg>
                <arg name="text2" type="xpath">TITLE/text()</arg>
                <arg name="delimiter" type="constant">_</arg>
              </instance_generator>
        </entity>
        </target_node>
     </domain>
   </mapping>
   </mappings>
</x3ml>
```

RDF Output:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex="http://example/"
    xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <crm:E22_Man-Made_Object
            rdf:about="http://www.ics.forth.gr/isl/ID_GR_1_ID_UK_1/Nomisma1_Coin1"/>
  <crm:E22_Man-Made_Object rdf:about=" http://www.ics.forth.gr/isl/ID_GR_2_ID_UK_2"/>
</rdf:RDF>
```

The only case where a value should not be generated is, if all terms (i.e. text1, text2) cannot be found in the input. If some of the terms are missing (but not all of them), then the URI will be generated with the existing ones.

1.1.6 DateNormalizer

1.1.6.1 Description

This generator function implements a date interpreter that generates a formal timestamp based on the given input. The generator transforms a limited part of a date such as a year (e.g. 2018), a year and month (e.g. 2018-01) etc. into a full date (e.g. 2018-01-15) or timestamp (2018-01-15T23:59:59). The generator uses a date value, a binary argument “report” which defines the format of the transformed value (i.e. either date or timestamp), and a binary argument “bound” indicating whether the generated date/timestamp should be in the beginning or in the end of the given value. In details, the arguments are:

- **text**: the value that indicates which part of the input will be used as the date that will create the date or timestamp. The value can be either a year (i.e. 2018), a year and month (2018-01) or a date (i.e. 2018-01-15).
- **report**: defines the format of the transformed date/timestamp. The acceptable values are “Date” and “Date_and_Time”. “Date” produces a date literal (e.g. 2018-01-15), while “Date_and_Time” produces a timestamp (e.g. 2018-01-15T26:59:59).
- **bound**: is a constant that takes two possible values, “Upper” or “Lower”, indicating whether the date/timestamp should be in the end of the given date/timestamp, or in the beginning respectively.

The following table shows some indicative examples

Text	**Report**	**Bound**	**Result**
2018	Date	Lower	2018-01-01
		Upper	2018-12-31
	Date_and_Time	Lower	2018-01-01T00:00:00
		Upper	2018-12-31T23:59:59
2018-04	Date	Lower	2018-04-01
		Upper	2018-04-30
	Date_and_Time	Lower	2018-04-01T00:00:00
		Upper	2018-04-30T23:59:59
2018-07-21	Date	Lower	2018-07-21
		Upper	2018-07-21
	Date_and_Time	Lower	2018-07-21T00:00:00
		Upper	2018-07-21T23:59:59

1.1.6.2 Example of Use

Sample XML input:

```xml
<COIN>
    <DATE_FROM>1625</DATE_FROM>
    <DATE_TO>1810</DATE_TO>
</COIN>
```

Sample Generator Policy file:

```xml
<generator name="DateNormalizer">
        <custom generatorClass="gr.forth.DateNormalizer">
                <set-arg name="bound" type="constant"/>
                <set-arg name="report" type="constant"/>
                <set-arg name="text"/>
        </custom>
```

```
</generator>
```

Sample X3ML mapping file:

```
...
    <link>
        <path>
                <source_relation>
                        <relation>DATE_FROM</relation>
                </source_relation>
                <target_relation>
                        <relationship>crm:P82a_begin_of_the_begin</relationship>
                </target_relation>
        </path>
        <range>
                <source_node>DATE_FROM</source_node>
                <target_node>
                        <entity>
                                <type>http://www.w3.org/2001/XMLSchema#dateTime</type>
                                <instance_generator name="DateNormalizer">
                                        <arg name="bound" type="constant">Lower</arg>
                                        <arg name="report" type="constant">Date</arg>
                                        <arg name="text" type="xpath">text()</arg>
                                </instance_generator>
                        </entity>
                </target_node>
        </range>
     </link>
     <link>
        <path>
                <source_relation>
                        <relation>DATE_FROM</relation>
                </source_relation>
                <target_relation>
                        <relationship>crm:P82b_end_of_the_end</relationship>
                </target_relation>
        </path>
        <range>
                <source_node>DATE_TO</source_node>
                <target_node>
                        <entity>
                                <type>xsd:dateTime</type>
                                <instance_generator name="DateNormalizer">
                                        <arg name="bound" type="constant">Upper</arg>
                                        <arg name="report" type="constant">Date</arg>
                                        <arg name="text" type="xpath">text()</arg>
                                </instance_generator>
                        </entity>
                </target_node>
        </range>
      </link>
...
```

RDF Output:

```

<crm:P82a_begin_of_the_begin rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
        1625-01-01
</crm:P82a_begin_of_the_begin>
<crm:P82b_end_of_the_end rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
        1810-12-31
</crm:P82b_end_of_the_end>
```

1.1.7 RemoveTerm

1.1.7.1 Description

This function takes a string and a term as input, removes the term from the string and returns the reduced string. The generator works both for URIs and literals.

1.1.7.2 Example of Use

Definition in the Generator Policy file:

```
<generator name="RemoveTerm">
        <custom generatorClass="gr.forth.RemoveTerm">
                <set-arg name="termToRemove" type="constant"/>
                <set-arg name="text"/>
        </custom>
</generator>
```

Sample XML input:

```
<dataroot>
    <COIN>
        <URI>http://localhost/UNWANTED_TERM-coin_100</URI>
    </COIN>
</dataroot>
```

Sample X3ML mapping file:

```
<x3ml version="1.0" source_type="xpath">
    <namespaces>
        <namespace prefix="crm" uri="http://www.cidoc-crm.org/cidoc-crm/"/>
        <namespace prefix="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <namespace prefix="rdfs" uri="http://www.w3.org/2000/01/rdf-schema#"/>
    </namespaces>
    <mappings>
        <mapping>
        <domain>
                <source_node>//COIN</source_node>
                <target_node>
                <entity>
                <type>crm:E22_Man-Made_Object</type>
                   <instance_generator name="RemoveTerm">
                        <arg name="termToRemove">UNWANTED_TERM-</arg>
                        <arg name="text">text()</arg>
                   </instance_generator>
                   <label_generator name="RemoveTerm">
                        <arg name="termToRemove">http://localhost/UNWANTED_TERM-</arg>
                        <arg name="text">text()</arg>
                   </label_generator>
                </entity>
                </target_node>
        </domain>
        </mapping>
    </mappings>
</x3ml>
```

RDF/XML Output:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <crm:E22_Man-Made_Object rdf:about="http://localhost/coin_100">
        <rdfs:label>C100</rdfs:label>
    </crm:E22_Man-Made_Object>
</rdf:RDF>
```

1.1.8 TextualContent

1.1.8.1 Description

This generator function generates a URN based on the textual contents. It receives as input only one parameter with name "text". Given a particular String value it generates a URN (using the scheme URN:UUID:<UUID_GENERATED_VALUE>). The generator is responsible for producing different values for different input texts. As a result if a text value is given twice then only one URN will be generated (in the first time and re-used afterwards).

When the TextualContent generator is defined with a prefix, then a URI (with the given prefix) will be generated, otherwise a UUID under the URN scheme will be generated.

1.1.8.2 Example of Use

Definition in the Generator Policy file:

```
<generator_policy>
    <generator name="URNfromTextualContent">
        <custom generatorClass="gr.forth.TextualContent">
            <set-arg name="text" type="xpath"/>
        </custom>
    </generator>
    <generator name="URIorUUID">
        <custom generatorClass="gr.forth.URIorUUID">
            <set-arg name="text"/>
        </custom>
    </generator>
</generator_policy>
```

Sample XML input:

```
<dataroot>
    <COIN>
        <ID>http://localhost/coin_1</ID>
        <COUNTRY_ID>Ancient Greece</COUNTRY_ID>
    </COIN>
    <COIN>
        <ID>http://localhost/coin_2</ID>
        <COUNTRY_ID>Ancient Rome</COUNTRY_ID>
    </COIN>
    <COIN>
        <ID>http://localhost/coin_3</ID>
        <COUNTRY_ID>
                 Ancient Greece
         </COUNTRY_ID>
    </COIN>
</dataroot>
```

Sample X3ML mapping file:

```
<x3ml version="1.0" source_type="xpath">
    <namespaces>
        <namespace prefix="loc" uri="http://localost/schema/"/>
        <namespace prefix="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <namespace prefix="rdfs" uri="http://www.w3.org/2000/01/rdf-schema#"/>
    </namespaces>
    <mappings>
        <mapping>
            <domain>
                <source_node>//COIN</source_node>
                <target_node>
                    <entity>
                        <type>loc:Man-Made_Object</type>
                         <instance_generator name="URIorUUID">
                                 <arg name="text" type="xpath">ID/text()</arg>
                         </instance_generator>
                    </entity>
                </target_node>
            </domain>
            <link>
                <path>
                    <source_relation><relation>COUNTRY_ID</relation></source_relation>
                    <target_relation>
                        <relationship>loc:has_country</relationship>
                    </target_relation>
                </path>
                <range>
                    <source_node>COUNTRY_ID</source_node>
                    <target_node>
                        <entity>
                            <type>loc:Country</type>
                            <instance_generator name="URNfromTextualContent">
                                 <arg name="text" type="xpath">text()</arg>
                             </instance_generator>
                            <label_generator name="Literal">
                                <arg name="text" type="xpath">text()</arg>
                                 <arg name="language" type="constant">en</arg>
                            </label_generator>
                        </entity>
                    </target_node>
                </range>
            </link>
        </mapping>
    </mappings>
</x3ml>
```

RDF/XML Output:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:loc="http://localost/schema/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <loc:Man-Made_Object rdf:about="http://localhost/coin_1">
        <loc:has_country>
            <loc:Country rdf:about="URN:UUID:d6d2800a-f5d1-3768-9c47-3a5494246c2e">
                <rdfs:label xml:lang="en">Ancient Greece</rdfs:label>
            </loc:Country>
        </loc:has_country>
    </loc:Man-Made_Object>
    <loc:Man-Made_Object rdf:about="http://localhost/coin_2">
        <loc:has_country>
            <loc:Country rdf:about="URN:UUID:c921a09f-e9ed-3bf2-a810-c8d290e01691">
                <rdfs:label xml:lang="en">Ancient Rome</rdfs:label>
            </loc:Country>
        </loc:has_country>
    </loc:Man-Made_Object>
    <loc:Man-Made_Object rdf:about="http://localhost/coin_3">
        <loc:has_country rdf:resource="URN:UUID:d6d2800a-f5d1-3768-9c47-3a5494246c2e"/>
    </loc:Man-Made_Object>
</rdf:RDF>
```

1.1.9 TypedLiteralGenerator

1.1.9.1 Description

Implement a new generator that will export literal values of specific types (i.e. xsd:date, xsd:anyUri). The purpose is to export the type of the literal in the RDF output i.e.

https://localhost/resource1 http://www.w3.org/2002/07/owl#sameas

"http://localhost/resource2"^^http://www.w3.org/2001/XMLSchema#anyURI

1.1.9.2 Example of Use

Definition in the Generator Policy file:

```
<generator name="TypedLiteralGen">
        <custom generatorClass="gr.forth.TypedLiteralGenerator">
                <set-arg name="text"/>
        </custom>
</generator>
```

Sample XML input:

```
<dataroot>
    <COIN>
        <ID>ID_1</ID>
        <FOUND>2018-03-12</FOUND>
     </COIN>
</dataroot>
```

Sample X3ML mapping file:

```
...
<link>
        <path>
                <source_relation>
                        <relation>FOUND</relation>
                </source_relation>
                <target_relation>
                       <relationship>crm:PX_was_found_on_typed</relationship>
                 </target_relation>
         </path>
         <range>
                 <source_node>FOUND</source_node>
                 <target_node>
                     <entity>
                          <type>xsd:date</type>
                          <instance_generator name="TypedLiteralGen">
                                 <arg name="text">text()</arg>
                        </instance_generator>
                     </entity>
                 </target_node>
          </range>
</link>
...
```

NTRIPLES Output:

```
<uuid:XXXX> <http://www.cidoc-crm.org/cidoc-crm/PX_was_found_on_untyped> "2018-03-12"^^xsd:date .
```

1.1.10 Producing shortened URIs

1.1.10.1 Description

A shorten attribute has been implemented in all custom generators. This attribute supports the generation of a shortened version of a lengthy URI. For example instead of creating the URI http://example.org/foundation%20for%20research%20and%20technology%20hellas it will create the shortened URI http://example.org/802DE79F-E672-3EDE-9A69-0C64B5A391E0

If the attribute "shorten" has the value "yes" then the generator:

- creates a UUID-like hash (e.g. 3644A684-F98E-38FE-A23C-713B77189A77)
- hashes only the last part of the URI. This allows preserving parts of the URI that reveal the hierarchy (e.g. The URI http://localhost/object/appellation/the%20name%20of%20the%object will be hashed to http://localhost/ object /appellation/802DE79F-E672-3EDE-9A69-0C64B5A391E0).

1.1.10.2 Example of Use

Definition in the Generator Policy file:

```
<generator name="LocalTermURIShorten" prefix="abc" shorten="yes">
        <pattern>{hierarchy}/{term}</pattern>
</generator>
```

Sample XML input:

```
<dataroot>
    <COIN>
        <ID>123</ID>
    </COIN>
    <COIN>
        <ID>456</ID>
    </COIN>
</dataroot>
```

Sample X3ML mapping file:

```
...
<namespaces>
        <namespace prefix="abc" uri="http://www.example.com/"/>
</namespaces>
...
```

```
<instance_generator name="LocalTermURIShorten">
        <arg name="hierarchy" type="constant">coin</arg>
        <arg name="term" type="xpath">text()</arg>
</instance_generator>
...
```

RDF/XML Output:

```
Will generate a URI of the form: http://www.example.com/85e0f73c-2450-4d28-98ed-
aaa3c51a84e4
```

1.1.11 Producing UUID-suffixed URIs

1.1.11.1 Description

The Hashed URI generators shown above are useful whenever there are contents from the input to be used for constructing the hash UUID suffix. However many times, such input data do not exist. Despite this, it is necessary to be able to create different URIs for different input elements. For this reason, there is the UUID-suffix facility for constructing URIs. This facility will create a URI using all the arguments of the user-defined generator and it will append the resulted URI with a random UUID. It is evident that in comparison with the previous generator (Hashed URI generator) this one will create a different URI every time (since the UUID is created in a random manner). In order to enable this functionality the optional attribute uuid must be used, with its value set to yes. The user-defined generator shown below will create URIs of the form: http://www.example.com/product/identifier/37252d3b-244f-4d10-85e4-c2415dac2453

1.1.11.2 Example of Use

Definition in the Generator Policy file:

```
<generator name="SampleGenerator" prefix="ex" uuid="yes">
        <pattern>{concept}/{kind}/</pattern>
</generator>
```

Sample XML input:

```
<dataroot>
    <COIN>
        <ID>123</ID>
    </COIN>
    <COIN>
        <ID>456</ID>
    </COIN>
</dataroot>
```

```
<x3ml version="1.0" source_type="xpath">
    <namespaces>
        <namespace prefix="crm" uri="http://www.cidoc-crm.org/cidoc-crm/"/>
        <namespace prefix="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <namespace prefix="rdfs" uri="http://www.w3.org/2000/01/rdf-schema#"/>
        <namespace prefix="ex" uri="http://www.example.com/"/>
    </namespaces>
    <mappings>
        <mapping>
        <domain>
                <source_node>//COIN</source_node>
                <target_node>
                <entity>
```

```
                         <type>crm:E22_Man-Made_Object</type>
                         <instance_generator name="SampleGenerator">
                                 <arg name="concept" type="constant">object</arg>
                                 <arg name="kind" type="constant">value</arg>
                         </instance_generator>
                 </entity>
                 </target_node>
         </domain>
        </mapping>
    </mappings>
</x3ml>
```

RDF/XML Output:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:ex="http://www.example.com/">
<crm:E22_Man-Made_Object
  rdf:about="http://www.example.com/object/value/c99d5c2b-db59-4f20-b5c6-2051b5795e02"/>
<crm:E22_Man-Made_Object
  rdf:about="http://www.example.com/object/value/ ab7a8332-aa34-4db7-a611-537db1bb14b4"/>
</rdf:RDF>
```

1.2 Label Generators

X3ML handles label generators in a similar way to instance generators. Each created instance can be assigned one instance generator but more than one label generators.

There are two built in label generators: Literal which will create an rdfs:label and prefLabel which will create a skos:prefLabel.

Usage in the X3ML file:

```
<label_generator name="Literal">
       <arg name="text" type="xpath">text()</arg>
</label_generator>

<label_generator name="prefLabel">
       <arg name="text" type="xpath">text()</arg>
</label_generator>
```

Label generators can also be defined with templates as instance generators:

Definition in the Generator Policy file:

```
<generator name="SimpleLabel">
        <pattern>{label}</pattern>
</generator>
```

Usage in the X3ML file:

```
<label_generator name="SimpleLabel">
        <arg name="label">text()</arg>
</label_generator>

<label_generator name="SimpleLabel">
        <arg name="label" type="constant">The-British-Museum</arg>
</label_generator>

<label_generator name="SimpleLabel">
        <arg name="label" type="constant">The-British-Museum</arg>
        <arg name="language" type="constant">en</arg>
</label_generator>
```

A typical range in X3ML would look like:

```
….
<range>
    <source_node>priref</source_node>
    <target_node>
        <entity>
            <type>crm:E42_Identifier</type>
            <instance_generator name="PersistentIdThing">
                <arg name="persistent" type="xpath">../PersistentIdentifier/text()</arg>
                <arg name="thing" type="constant">priref</arg>
            </instance_generator>
            <label_generator name="Literal">
                <arg name="text" type="xpath">../../PersistentIdentifier/text()</arg>
            </label_generator>
        </entity>
    </target_node>
</range>
```

For the objects with:

```
<record priref="1" >
```

```
        <PersistentIdentifier>RM0001.COLLECT.1</PersistentIdentifier>
</record>
<record priref="2" >
        <PersistentIdentifier>RM0001.COLLECT.2</PersistentIdentifier>
</record>
```

and generator (with lan=`http://hdl.handle.net/10934/`):

```
<generator name="PersistentIdThing" prefix="han">
        <pattern>{persistent}/{thing}</pattern>
 </generator>
```

will produce the RDF triples:

```
<crm:E42_Identifier rdf:about="http://hdl.handle.net/10934/RM0001.COLLECT.1/priref"/>
        <rdfs:label> RM0001.COLLECT.1</rdfs:label>
</crm:E42_Identifier>
<crm:E42_Identifier rdf:about="http://hdl.handle.net/10934/RM0001.COLLECT.2/priref"/>
        <rdfs:label> RM0001.COLLECT.2</rdfs:label>
</crm:E42_Identifier>
```

2 APPENDIX A: Sample generator policy file

```xml
<generator_policy>
     <generator name="SimpleLabel">
        <pattern>{label}</pattern>
     </generator>

     <generator name="LocalTermURI" prefix="ex">
        <!-- The prefix is a namespace that must be declared in the X3ML definition. -->
        <!-- in info tab define <namespace prefix="ex" uri="http://example.gr/"/> -->
        <pattern>{hierarchy}/{term}</pattern>
     </generator>

     <generator name="DateNormalizer">
        <custom generatorClass="gr.forth.DateNormalizer">
                <set-arg name="bound" type="constant"/>
                <set-arg name="report" type="constant"/>
                <set-arg name="text"/>
        </custom>
     </generator>

     <generator name="URIorUUID">
        <custom generatorClass="gr.forth.URIorUUID">
            <set-arg name="text"/>
        </custom>
     </generator>
</generator_policy>
```