Definition Breakdown

The data is extracted from a relational source and is output as XML, which is loaded into MarkLogic. This is the start of the process, resulting in patient documents and encounter documents (for example). While it is possible to ingest the data “as-is,” the data is often more useful when enriched. In this application, the enrichment process is performed within MarkLogic. Part of this enrichment is to facilitate search by adding collections or “rearranging” the data to make search faster or more feature rich. In the case of DVIEVR, an important aspect of the enrichment is the creation of triples from the XML such that semantic queries and inferencing about documents may provide more useful search.



A key goal is to provide a way for the ontologist to map fields from the XML documents to elements in the ontology that defines the concepts and elements in DVIEVR. Starting at the relational database level, a gender definition might have the following values: “1”, “2”, and “3” for: “male”, “female”, and “unknown” respectively. The ontology for DVIEVR may not track these as “1”, “2”, “3”, but may define a concept of gender. The definition might be a predicate such as “DVIEVR:hasGender” and that has a range of “DVIEVR:male”, “DVIEVR:female”, and “DVIEVR:unknown.” The resulting triple, from the ingested data, should have the URI of the patient as its subject, the “DVIEVR:hasGender” predicate, and the appropriate gender object (such as “DVIEVR:female”).

The URI for a patient, encounter, or other entity will be rooted in their document URI within MarkLogic. Each document in MarkLogic has a unique URI. This can follow the form of “http://some-domain.tld/some/path/1234.xml” be as simple as “/patients/1234.xml.” The URI is a sufficiently good starting point to root a noun in the ontology, so that “/patients/1234.xml” will serve as the subject URI for statements in RDF about the given patient. The URI can easily be calculated from the patient id.

Likewise, an encounter would be rooted at the URI for an encounter “/encounters/4567.xml.” As with patients, the URI is quickly calculated from the encounter ID. The same pattern can be used for other entities that will be represented as XML documents such as providers. In some cases the data will be de-normalized and represented multiple times. For example, for purposes of search, a patient will have their encounters de-normalized into the patient document. An encounter will also be represented as its own document. However, when making statements about patients, the patient URI is used. When making assertions about encounters, the encounter URI is used.

# Defining a transformation:

A definition for a transformation has a root entity. In the case of the examples below, the simplified URIs with “xform-instance” indicate the mapping defined by the ontologist while the simplified URIs with “xform-ontology” refer to the ontology that defines the mapping. In the case below we have a transformation named “translate-std-gender-id.” This will map gender values from a patient XML document to a triple that ties the patient to a concept of gender in the DVIEVR ontology. (A simplified URL containing “example-ontology.org” is a stand-in for the actual ontology.)

The transformation has two attributes. The first is an XPath expression used to extract values from the XML document. The software that executes this mapping will execute the XPath against a document to be enriched to produce a raw value. The second attribute defines a predicate to be used when constructing the triple that maps a patient to the concept of gender. For patient 1234, the triple would result in something like:

{

“/patients/1234.xml”,

“http://example-ongology.org/demographics#patient-gender”,

“http://example-ontology.org/demographics#female”

}

The transformation has some object relationships to other entities that define how a value such as “2” is mapped to the URI “http://example-ontology.org/demographics#female”. These object relationships point to the details of how to translate those values. This is covered below in “Defining a Mapping.”

<owl:NamedIndividual

rdf:about="http://xform-instance/transformations#translate-std-gender-id">

<rdf:type rdf:resource="http://xform-ontology/transformations#IngestTransformation"/>

<transformations

rdf:resource="http://xform-instance/transformations#gender-id-default-mapping"/>

<transformations

rdf:resource="http://xform-instance/transformations#gender-id-female-mapping"/>

<transformations

rdf:resource="http://xform-instance/transformations#gender-id-male-mapping"/>

<predicate

rdf:datatype="http://www.w3.org/2001/XMLSchema#string"><http://example-ontology.org/demographics#patient-gender></predicate>

<xpath

rdf:datatype="http://www.w3.org/2001/XMLSchema#string">//env:original-data/STD\_GENDER\_ID/text()</xpath>

</owl:NamedIndividual>

# Defining a mapping:

The transformation entity refers mappings which translate a raw value such as “2” to a URI that represents the concept of female gender. Initially there are three different mappings. The first is a simple value mapping, which is meant to map a raw value to a translated raw value or to a URI. The second is a “past-through” mapping which takes the raw value and simply returns it. Thus, the ontologist might map a comments field using the pass-through mapping so that it appears as a value object. Third is a function definition that executes code to generate a mapping value or URI.

## Simple Value Mapping

A simple value mapping is meant to map specific values to a another value. For example, mapping an indicator such as the “hospitalized indicator” to a value or to a URI that represents a concept in the DVIEVR ontology. As example, assume the following table regarding gender mapping:

|  |  |
| --- | --- |
| Raw Value | DVIEVR Ontology URI |
| 1 | http://example-ontology.org/demographics#male |
| 2 | http://example-ontology.org/demographics#female |
| 3 | http://example-ontology.org/demographics#unknown |
| 4 | http://example-ontology.org/demographics#trans-male |
| 5 | http://example-ontology.org/demographics#trans-female |

In this case the XML document may have the values from “1” to “5”. If the value is a “1” in the XML document, then it should be translated to “http://example-ontology.org/demographics#male”. The resulting triple for patient 1234 (given the above translation definition) would be:

{

“/patients/1234.xml”,

“http://example-ongology.org/demographics#patient-gender”,

“http://example-ontology.org/demographics#male”

}

<owl:NamedIndividual

rdf:about="http://xform-instance/transformations#gender-id-default-mapping">

<rdf:type

rdf:resource="http://xform-ontology/transformations#SimpleValueMapping"/>

<inputValue

rdf:datatype="http://www.w3.org/2001/XMLSchema#string">3</inputValue>

<subjectUri

rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://example-ontology.org/demographics#unknown</subjectUri>

</owl:NamedIndividual>

## Pass-Through Mapping

In some cases there is no logical translation for a value. For example, given encounter 4567, there is a “http://example-ongology.org/encounter#diagnosisNotes” predicate that relates an encounter to the diagnosis notes for that encounter. In this case the transformation simply creates a triple such as:

{

“/encounters/4567.xml”,

“http://example-ontology.org/encounter#diagnosisNotes”,

“The patient has multiple ….”

}

<owl:NamedIndividual

rdf:about="http://xform-instance/transformations#translate-std-gender-id">

<rdf:type rdf:resource="http://xform-ontology/transformations#IngestTransformation"/>

<transformations

rdf:resource="http://xform-instance/transformations#diag-notes-passthrough"/?

<predicate

rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://example-ontology.org/encounter#diagnosisNotes</predicate>

<xpath

rdf:datatype="http://www.w3.org/2001/XMLSchema#string">//env:original-data/PHYSICIAN\_DIAG\_NOTES\_TEXT/text()</xpath>

</owl:NamedIndividual>

<owl:NamedIndividual

rdf:about="http://xform-instance/transformations#diag-notes-passthrough">

<rdf:type

rdf:resource="http://xform-ontology/transformations#PassthroughValueMapping"/>

</owl:NamedIndividual>

## Invoke Transform Mapping

In some cases it may be necessary to invoke a transformation function to produce a correct value. For the sake of exposition, assume in the XQuery code there is a file “/ext/transforms/date-utils.xqy” that contains some utility functions around dates. For example, it has a function that translates a date such as “4/15/2016” to the ISO standard “2016-04-15.” In this case the object will be the value returned by this function.

<owl:NamedIndividual

rdf:about="http://xform-instance/transformations#invoke-date-translator">

<rdf:type

rdf:resource="http://xform-ontology/transformations#InvokeTransformMapping"/>

<transformModuleFunction

rdf:datatype="http://www.w3.org/2001/XMLSchema#string">fix-database-date</transformModuleFunction>

<transformModuleNamespace

rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://my-xquery-namespace/transforms/functions</transformModuleNamespace>

<transformModuleUri

rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">/ext/transforms/date-utils.xqy</transformModuleUri>

</owl:NamedIndividual>