SDMX-XML to DDI-CDI structural mapping specification

1. Context and problem statement

The goal is to automate the transformation of metadata from an SDMX-XML format to a DDI-CDI (RDF) representation. The challenge is transforming hierarchical XML references into explicit RDF graph edges. For example, there is a challenge in translating the structural relationship between a **Dataflow** and its underlying **DataStructure Definition (DSD)** into the graph structure of a DDI-CDI representation.

The snippet below displays an example of a source SDMX-XML structure. Note specifically how the Dataflow element nests a reference (Ref) to the DataStructure, which contains the essential attributes required to build the RDF relationship.

```
<?xml version='1.0' encoding='UTF-8'?>
<mes:Structure xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    xmlns:mes="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/message"
    xmlns:str="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/structure"
    xmlns:com="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/common"
    xsi:schemaLocation="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/message
    https://registry.sdmx.org/schemas/v2_1/SDMXMessage.xsd">
    <mes:Header>
        <mes:ID>IREF206846</mes:ID>
        <mes:Test>false</mes:Test>
        <mes:Prepared>2025-11-16T19:18:06Z</mes:Prepared>
        <mes:Sender id="UNICEF"/>
        <mes:Receiver id="not_supplied"/>
    </mes:Header>
    <mes:Structures>
        <str:Dataflows>
            <str:Dataflow
                urn="urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=UNICEF:CHLD_PVTY(1.0)"
                isExternalReference="false" agencyID="UNICEF" id="CHLD_PVTY" isFinal="true"
                version="1.0">
                <com:Name xml:lang="en">Child Poverty</com:Name>
                <str:Structure>
                    <Ref package="datastructure" agencyID="UNICEF" id="DSD_CHLD_POV"
                         version="1.0" class="DataStructure"/>
                </str:Structure>
            </str:Dataflow>
        </str:Dataflows>
        <str:DataStructures>
            <str:DataStructure
        urn="urn:sdmx:org.sdmx.infomodel.datastructure.DataStructure=UNICEF:DSD_CHLD_POV(1.0)"
                isExternalReference="false" agencyID="UNICEF" id="DSD_CHLD_POV" isFinal="false"
                version="1.0">
                <com:Name xml:lang="en">Child Poverty</com:Name>
                <str:DataStructureComponents>
                  ...
                </str:DataStructureComponents>
            </str:DataStructure>
        </str:DataStructures>
    </mes:Structures>
</mes:Structure>
```

While the individual entity mappings are established:

- **Source:** `urn:sdmx:org...Dataflow=UNICEF:CHLD_PVTY(1.0)`
  - **Target:** `cdi:UNICEF/CHLD_PVTY/1.0/`
- **Source:** `urn:sdmx:org...DataStructure=UNICEF:DSD_CHLD_POV(1.0)`
  - **Target:** `cdi:UNICEF/DSD_CHLD_POV/1.0`

The missing component is the logic to generate the predicate `cdi:isStructuredBy` which links these two entities.

2. Source data analysis

In the source XML message, the link is defined via a nested **`<Ref>`** element within the **`Dataflow`** definition:

```
<str:Dataflow ... agencyID="UNICEF" id="CHLD_PVTY" version="1.0">
    <str:Structure>
        <Ref package="datastructure" agencyID="UNICEF" id="DSD_CHLD_POV" version="1.0" .../>
    </str:Structure>
</str:Dataflow>
```

3. Mapping logic

To generate the `cdi:isStructuredBy` triple, the mapping must define the `Dataflow` element as the subject node (the **Subject**). From this context, it must traverse the child axis to find the **`Ref`** element to construct the reference URI (the **Object**).

The logical mapping definition is:

- **Subject:** Constructed from the attributes of the current node (`str:Dataflow`).
  - Formula: `cdi:{Dataflow/@agencyID}/{Dataflow/@id}/{Dataflow/@version}/`
- **Predicate:** Constant.
  - Value: `cdi:isStructuredBy`
- **Object:** Constructed from the attributes of the child node (`str:Structure/Ref`).
  - Expression: `cdi:{Ref/@agencyID}/{Ref/@id}/{Ref/@version}`

4. Implementation specifications

Below are two possible technical implementations for this logic using RML (Declarative) and XSLT (Procedural), respectively.

Approach A: RML (RDF mapping language)

In RML, this is achieved by using a **`predicateObjectMap`** that operates within the iteration of the `Dataflow`, but selects the object values using a relative XPath

→ Create inside RML URIs for each of the rr:TripleMap

```
@prefix rr: http://www.w3.org/ns/r2rml#  .
@prefix rml: http://semweb.mmlab.be/ns/rml# .
@prefix ql: http://semweb.mmlab.be/ns/ql# .
@prefix cdi: http://ddialliance.org/Specification/DDI-CDI/1.0/RDF/ .
@prefix str: http://www.sdmx.org/resources/sdmxml/schemas/v2_1/structure .

# ------------------------------------------------------------
# 1. Dataflow Mapping
#    Defines the Dataset and links to the DSD URI
# ------------------------------------------------------------

<#DataflowMapping> a rr:TriplesMap ;
    rml:logicalSource [
        rml:source "source.xml" ;
        rml:referenceFormulation ql:XPath ;
        rml:iterator "//*[local-name()='Dataflow']"
    ] ;

    # Generate the Subject (The Dataflow)

    rr:subjectMap [
        rr:template "cdi:{@agencyID}/{@id}/{@version}/" ;
        rr:class cdi:DimensionalDataset
    ] ;

    # Generate the Relationship

    rr:predicateObjectMap [
        rr:predicate cdi:isStructuredBy ;
        rr:objectMap [
            rr:template "cdi:{str:Structure/Ref/@agencyID}/
                         {str:Structure/Ref/@id}/
                         {str:Structure/Ref/@version}"

        ]

    ] .

# ------------------------------------------------------------
# 2. DataStructure Mapping
#    Defines the DSD and asserts its Class
# ------------------------------------------------------------

<#DataStructureDefinitionMapping> a rr:TriplesMap ;
    rml:logicalSource [
        rml:source "source.xml" ;
        rml:referenceFormulation ql:XPath ;
```

```
    rml:iterator "//*[local-name()='DataStructure']"
] ;

# Generate the Subject (The DSD)

rr:subjectMap [
    rr:template "cdi:{@agencyID}/{@id}/{@version}/" ;
    rr:class cdi:DimensionalDataStructure
] .
```

In this approach, the RML artifact acts as a declarative set of instructions. It tells a mapping engine to "loop" through specific XML elements and turn them into RDF triples. For example, in the case of the Dataflow Mapping block:

a) Iterator: The `rml:logicalSource` section establishes the **context**.

```
rml:iterator "//*[local-name()='Dataflow']"
```

- This is equivalent to a `foreach` loop. It scans the entire source.xml file and stops at every element named `Dataflow` (ignoring namespace prefixes).
- The expression `"//*[local-name()='Dataflow']"` is a specific instruction to the XML parser to find the source data regardless of how it is labeled with namespaces.
    - **//: Recursive descent**. Start from the root of the document and look inside every single folder and sub-folder, no matter how deep.
    - ***: Wildcard selector**. Select any element node, regardless of its name. (the parser selects every tag in the source file).
    - `[local-name()='Dataflow']`**: Filter**. Keep only the elements whose name (stripped of namespace prefix) is exactly `'Dataflow'`.
- **Current context:** The engine is now "standing" on the `<str:Dataflow>` node. All subsequent XPath queries in this mapping are executed **relative** to this specific node.

b) Subject: The `rr:subjectMap` defines the subject of the RDF triple.

```
rr:template "cdi:{@agencyID}/{@id}/{@version}/"
```

- It looks at the *current* XML node (Dataflow) and extracts the attributes `agencyID`, `id`, and `version`.
- It creates the URI by concatenating the attributes (e.g., `cdi:UNICEF/CHLD_PVTY/1.0/`).

c) Predicate: The `rr:predicateObjectMap` defines the relationship.

```
rr:predicate cdi:isStructuredBy
```

- It establishes that the connection between the Subject and the Object will be `cdi:isStructuredBy`.

d) Object: The `rr:objectMap` defines what the Dataflow is linked *to*.

```
rr:objectMap [
    rr:template "cdi:{str:Structure/Ref/@agencyID}/
                 {str:Structure/Ref/@id}/
                 {str:Structure/Ref/@version}"
```

```
]
```

The logical source is still "standing" on `<str:Dataflow>`. To find the DSD information, it cannot look at the current node's attributes. It must "drill down" into the children.

- `str:Structure`: Go down into the child element named Structure (in the `str` namespace).
- `/Ref`: Go down further into the child element named `Ref`.
- `/@id`: Extract the id attribute from that Ref element.

Summary of the transformation

```
<str:Dataflow ... id="CHLD_PVTY" ...>   <-- Iterator acts here (Subject Source)
      |
      +-- <str:Structure>               <-- Path traversal
            |
            +-- <Ref id="DSD..." />     <-- Data extraction here (Object Source)
```

The engine combines these three parts into a single RDF triple:

1. **Subject:** `cdi:UNICEF/CHLD_PVTY/1.0/`
2. **Predicate:** `cdi:isStructuredBy`
3. **Object:** `cdi:UNICEF/DSD_CHLD_POV/1.0`

By using relative XPaths inside the template (`{path/to/element/@attribute}`), a link is created based on the *specific* reference nested inside the current Dataflow. If there were multiple Dataflows in one file, the iterator would process them one by one, always grabbing the correct corresponding Ref for that specific Dataflow.

Approach B: XSLT (Extensible stylesheet language transformations)

To translate the RML logic described above into XSLT, we define two separate `<xsl:template>` blocks. This mirrors the structure of having two `TriplesMap` definitions in RML.

The XSLT processor acts as the iterator. When it encounters a `Dataflow`, it triggers the first template. When it encounters a `DataStructure`, it triggers the second template.

```
<xsl:transform version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cdi="http://ddialliance.org/Specification/DDI-CDI/1.0/RDF/"
    xmlns:str="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/structure">

    <xsl:output method="xml" indent="yes"/>

<xsl:variable name="cdi_base"
    select="'http://ddialliance.org/Specification/DDI-CDI/1.0/RDF/'"/>

    <xsl:template match="/">
        <rdf:RDF>
            <xsl:apply-templates select="//str:Dataflow | //str:DataStructure"/>
        </rdf:RDF>
    </xsl:template>

    <xsl:template match="str:Dataflow">

        <xsl:variable name="subjectURI"
            select="concat('cdi:', @agencyID, '/', @id, '/', @version, '/')"/>

        <rdf:Description rdf:about="{$subjectURI}">

            <rdf:type rdf:resource="{$cdi_base}DimensionalDataset"/>

            <xsl:variable name="ref" select="str:Structure/Ref"/>
            <xsl:variable name="objectURI"
                select="concat('cdi:', $ref/@agencyID, '/', $ref/@id, '/', $ref/@version)"/>

            <cdi:isStructuredBy rdf:resource="{$objectURI}"/>

        </rdf:Description>

    </xsl:template>

    <xsl:template match="str:DataStructure">

        <xsl:variable name="subjectURI"
            select="concat('cdi:', @agencyID, '/', @id, '/', @version, '/')"/>

        <rdf:Description rdf:about="{$subjectURI}">

            <rdf:type rdf:resource="{$cdi_base}DimensionalDataStructure"/>

        </rdf:Description>
    </xsl:template>
```

```
</xsl:transform>
```