RDF transformations

Petr Křemen

Ontologies and Semantic Web
Winter 2025

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Problems

1. **How to create RDF from other data formats**

2. **How to reshape RDF to another RDF**

RDFize tabular data

ID	Given name	Family Name	Birth Date	Father ID
1	Alan	Turing	1912-06-23	2
2	Julius	Turing		

```
ex:AlanTuring
      a schema:Person ;
      schema:givenName "Alan" ;
      schema:familyName "Turing" ;
      schema:birthDate
            "1912-06-23"^^xsd:date ;
      schema:parent ex:JuliusMathisonTuring .

ex:JuliusTuring a schema:Person .
```

Approaches

single-purpose tools (CLI/libraries)

- TARQL (tool):
  - CSV -> RDF (using SPARQL syntax)
- R2RML: (language+tools)
  - SQL -> RDF
  - W3C recommendation

RDF libraries

- Java: Jena/RDF4J
- Python: rdflib
- …

multi-purpose tools

- RDF Mapping Language (RML) (language+tools)
  - CSV/JSON/XML/SQL -> RDF
  - W3C draft
  - RML Mapper, RML Streamer, RML Editor (UI)
- Ontotext Refine (UI):
  - CSV/XLS/JSON/XML -> RDF
- s-pipes - RDF pipelining tool
- barnard59 - CLI tool for RDF pipelines
- LinkedPipes ETL - suite + UI for RDF pipelines

TARQL

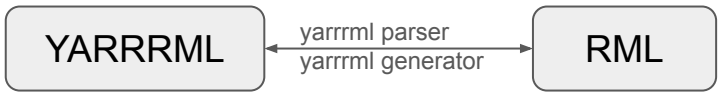
```
PREFIX : <http://data.sio2.cz/example/people/resource/>
PREFIX schema: <http://schema.org/>

CONSTRUCT {
   ?URI a schema:Person ;
        schema:familyName ?Family_Name ;
        schema:givenName ?Given_Name ;
        schema:birthDate ?Birth_Date ;
        schema:parent ?Father .
   ?Father schema:gender <https://schema.org/Male>
}
FROM <file:../examples/people/input/people.csv>
WHERE {
   BIND (IRI(CONCAT(STR(:),?ID)) AS ?URI)
   BIND (IRI(CONCAT(STR(:),?Father_ID)) AS ?Father)
}
```

RDF Mapping Language

RDF Mapping Language

- *rml.io*
- RML mappings themselves in RDF
- YARRRML is a human (more) readable syntax translatable to RML



Basic mapping


```
@base <http://example.org/people/rml/> .
@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 xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix schema: <http://schema.org/> .
```

ID	Given name	Family Name	Birth Date	Father ID
1	Alan	Turing	1912-06-23	2
2	Julius	Turing		

```
<people-logical-source>
    rml:source "people.csv" ;
    rml:referenceFormulation ql:CSV .

<people-mapping> a rr:TriplesMap ;
    rml:logicalSource <people-logical-source> ;
    rr:subjectMap <people-subject-map> ;
    rr:predicateObjectMap
        <people-family-name> ,
        <people-birth-date> ,
        <people-parent> .

<people-subject-map> rr:template "http://example.org/people/resource/{id}" ; rr:class schema:Person .
<people-family-name>  rr:predicate schema:familyName ; rr:objectMap <people-family-name-range> .
<people-family-name-range> rml:reference "Family Name" ; rr:datatype xsd:string .
<people-birth-date> rr:predicate schema:birthDate ; rr:objectMap <people-birth-date-range> .
<people-birth-date-range> rml:reference "Birth Date" ; rr:datatype xsd:date .
<people-parent> rr:predicate schema:parent ; rr:objectMap <people-parent-range> .
<people-parent-range> rr:template "http://example.org/people/resource/{Father ID}" .
```

Basic mapping YARRRML

```
prefixes:
  people: "http://data.sio2.cz/example/people/resource/"
  base: "http://data.sio2.cz/example/people/rml/"
  ql: "http://semweb.mmlab.be/ns/ql#"
  xsd: "http://www.w3.org/2001/XMLSchema#"
  schema: "http://schema.org/"

mappings:
  people-mapping:
    sources:
      - [ "../examples/people/input/people.json~jsonpath", "$.people[*]" ]
    s: people:$(ID)
    po:
      - [a, schema:Person]
      - p: schema:givenName
        o: $(Given Name)
        datatype: xsd:string
      - p: schema:familyName
        o: $(Family Name)
        datatype: xsd:string
      - p: schema:birthDate
        o: $(Birth Date)
        datatype: xsd:date
      - p: schema:parent
        o:
          type: iri
          value: people:$(Father ID)
```

ID	Given name	Family Name	Birth Date	Father ID
1	Alan	Turing	1912-06-23	2
2	Julius	Turing		

Logical source

Logical Source defines the source data

- CSV (resp. XML, JSON)

```
<source>
      rml:source "<INPUT_FILE_PATH>" ;
      rml:referenceFormulation ql:CSV .
            # (resp ql:XPath, ql:JSONPath)
```

- relational database

```
<source>
      rml:source [
            a d2rq:Database ;
            d2rq:jdbcDSN "<JDBC_URL>";
            d2rq:jdbcDriver "<JDBC_DRIVER";
            d2rq:username "<JDBC_USER>";
            d2rq:password "<JDBC_PASSWORD>" . ] ;
      rr:sqlVersion rr:SQL2008;
      rml:query """<SELECT_QUERY>""" .
```

```
<source>
      rml:source [
            a d2rq:Database;
            d2rq:jdbcDSN ;
            "jdbc:mysql://localhost/example";
            d2rq:jdbcDriver "com.mysql.jdbc.Driver";
            d2rq:username "user";
            d2rq:password "password" . ] ;
      rr:sqlVersion rr:SQL2008;
      rml:query """
            SELECT ID, FAMILY_NAME, BIRTH_DATE, FATHER_ID
            FROM PEOPLE;""" .
```

Logical source - JSONPath

Logical Source defines the source data

- CSV (resp. XML, JSON)

```
<source>
      rml:source “people.json” ;
      rml:referenceFormulation ql:JSONPath ;
      rml:iterator "$.people[*]" .
```

```
{
 "people": [
   {
     "ID": "1",
     "Given Name": "Alan",
     "Family Name": "Turing",
     "Birth Date": "1912-06-23",
     "Father ID": "2"
   },
   {
     "ID": "2",
     "Given Name": "Julius",
     "Family Name": "Turing"
  }
 ]
}
```

Triples maps

Triples map defines the transformation of a record to *one or more triples*

- *exactly one* **`rr:logicalSource`** (see above)
- *exactly one* **`rr:subjectMap`** or **`rr:subject`** (how to generate subjects)
- *zero or more* **`rr:predicateObjectMap`** (how to generate predicates/objects) consisting of
  - **`rr:predicateMap`** or **`rr:predicate`**
  - **`rr:objectMap`** or **`rr:object`**

```
<people-mapping> a rr:TriplesMap ;
    rml:logicalSource <people-logical-source> ;
    rr:subjectMap [
      rr:template
"http://example.org/people/resource/{id}" ;
      rr:class schema:Person .
    ] ;
    rr:predicateObjectMap [
      rr:predicate schema:familyName ;
      rr:objectMap [
        rml:reference "Family Name" ;
        rr:datatype xsd:string
      ]
    ].
```

subjects are created based on a IRI template with possible variables

objects are created by a reference to a field in the source data

predicate is `schema:familyName`

Subject maps

Subject map can be defined by

- **a template** (see above)
- **a constant** (`rr:subject` means `rr:subjectMap/rr:constant`)

```
rr:subjectMap [
  rr:constant "http://example.org/people/resource/catalog";
];
```

(all triples will have the same subject)

- **a reference**

```
rr:subjectMap [
  rr:reference "URL";
];
```

(subject IRI is taken from a field of the input source, or an XPath/JSONPath expression)

```
rr:subjectMap [
 rr:constant "http://example.org/dataset";
 rr:class dcat:Dataset .
] ;
```

generates

```
<http://example.org/dataset>
      a dcat:Dataset.
```

Predicate/object maps

Templates/constants/references can also be used for predicate maps / object maps in an analogous way.

```
@base <http://example.org/people/rml/> .
@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 xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix schema: <http://schema.org/> .

<people-logical-source>
    rml:source "input/people.csv" ;
    rml:referenceFormulation ql:CSV .

<people-catalog-mapping> a rr:TriplesMap ;
    rml:logicalSource <people-logical-source> ;
    rr:subjectMap [
      rr:constant "http://example.org/people/dataset" ;
      rr:class dcat:Dataset, void:Dataset . ] ;
    rr:predicateObjectMap [
      rr:predicate void:exampleResource ;
      rr:objectMap
"http://data.sio2.cz/example/people/resource/{ID}" ] ;
    rr:predicateObjectMap [
      rr:predicate dcterms:creator ;
      rr:object <mailto:petr.kremen@gmail.com> .] .
```

```
@base <http://example.org/people/rml/> .
@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 xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix schema: <http://schema.org/> .
@prefix: <http://example.org/people/resource/> .

<http://data.sio2.cz/example/people/dataset>
  a void:Dataset, dcat:Dataset;
  dcterms:creator <mailto:petr.kremen@gmail.com>;
  void:exampleResource :1,:2 .
```

Term maps based on term type

Subject/predicate/object maps are examples of **term maps**. For templates/references their **`rr:termType`** can be specified:

- **rr:IRI**
- **rr:BlankNode**
- **rr:Literal**

```
@base <http://example.org/people/rml/> .
@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 xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix schema: <http://schema.org/> .


…

<object-map-iri>
       rr:reference "Email" ;
       rr:termType rr:IRI .
…
```

Language Maps

Object maps with term type `rr:Literal` may have defined, how to generate a language tag for that literal. **A language map is another type of term map.**

```
@base <http://example.org/people/rml/> .
@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 xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix schema: <http://schema.org/> .


…

<people-family-name>
       rr:predicate schema:familyName;
       rr:objectMap [
              rml:reference “Family Name”
              rr:language “en-GB”
       ]
…
```

Joining triple maps

ID	Given name	Family Name	Birth Date	Father ID
1	Alan	Turing	1912-06-23	2
2	Julius	Turing		

Name	Date
Alan	8.3.
Julius	11.4.

```
<people-mapping> a rr:TriplesMap ;
    rml:logicalSource [ rml:source "input/people.csv" ; rml:referenceFormulation ql:CSV] ;
    rr:subjectMap [ rr:template "http://example.org/people/resource/{ID}" ] ;
    rr:predicateObjectMap [
      rr:predicate ontology:nameDay ;
      rr:objectMap [
            rr:parentTriplesMap <nameday-mapping>;
            rr:joinCondition [
                  rr:child "Given Name";
                  rr:parent "Name" ] ] ].

<nameday-mapping> a rr:TriplesMap ;
    rml:logicalSource [ rml:source "input/namedays.csv" ; rml:referenceFormulation ql:CSV ] ;
    rr:subjectMap [ rr:template "http://example.org/namedays/resource/{Name}"] ;
    rr:predicateObjectMap [
      rr:predicate ontology:value ;
      rr:objectMap [
            rml:reference "Date" ;
            rr:datatype xsd:string ]] .
```

Functions

How to transform the input data? E.g. string regular expressions, computations, etc. …

```
@base <http://example.org/people/rml/> .
@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 xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix fno: <https://w3id.org/function/ontology#> .
@prefix grel: <http://users.ugent.be/~bjdmeest/function/grel.ttl#> .
…
fnml:functionValue [
  rr:predicateObjectMap    [
    rr:predicate fno:executes ;
    rr:objectMap [ rr:constant grel:toUpperCase ] ] ;

  rr:predicateObjectMap    [
    rr:predicate grel:valueParameter ;
    rr:objectMap [ rml:reference "Family Name" ]
  ] ;
]
…
```

```
@prefix p : <http://example.org/people/resource> .
@prefix p-rml : <http://example.org/people/rml> .
@prefix n : <http://example.org/namedays/resource>.

p:1 a schema:Person;
 ontology:nameDay n:Alan;
 p-rml:family-name-uppercase "TURING";
 schema:birthDate "1912-06-23"^^xsd:date;
 schema:familyName "Turing"@en;
 schema:givenName "Alan";
 schema:parent p:2 .
```

Function to execute

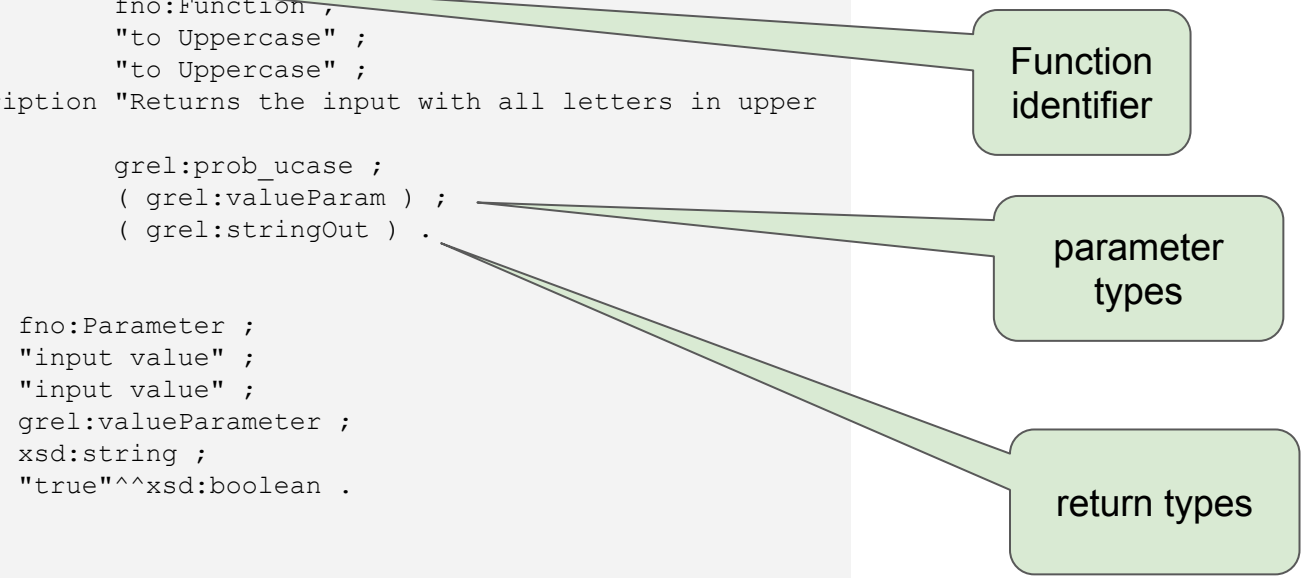
Its parameters

Function detail

```
…
grel:toUpperCase
    a                   fno:Function ;
    fno:name            "to Uppercase" ;
    rdfs:label          "to Uppercase" ;
    dcterms:description "Returns the input with all letters in upper
case." ;
    fno:solves          grel:prob_ucase ;
    fno:expects         ( grel:valueParam ) ;
    fno:returns         ( grel:stringOut ) .

grel:valueParam
    a             fno:Parameter ;
    fno:name      "input value" ;
    rdfs:label    "input value" ;
    fno:predicate grel:valueParameter ;
    fno:type      xsd:string ;
    fno:required  "true"^^xsd:boolean .


grel:stringOut
    a             fno:Output ;
    fno:name      "output string" ;
    rdfs:label    "output string" ;
    fno:predicate grel:stringOutput ;
    fno:type      xsd:string .
…
```



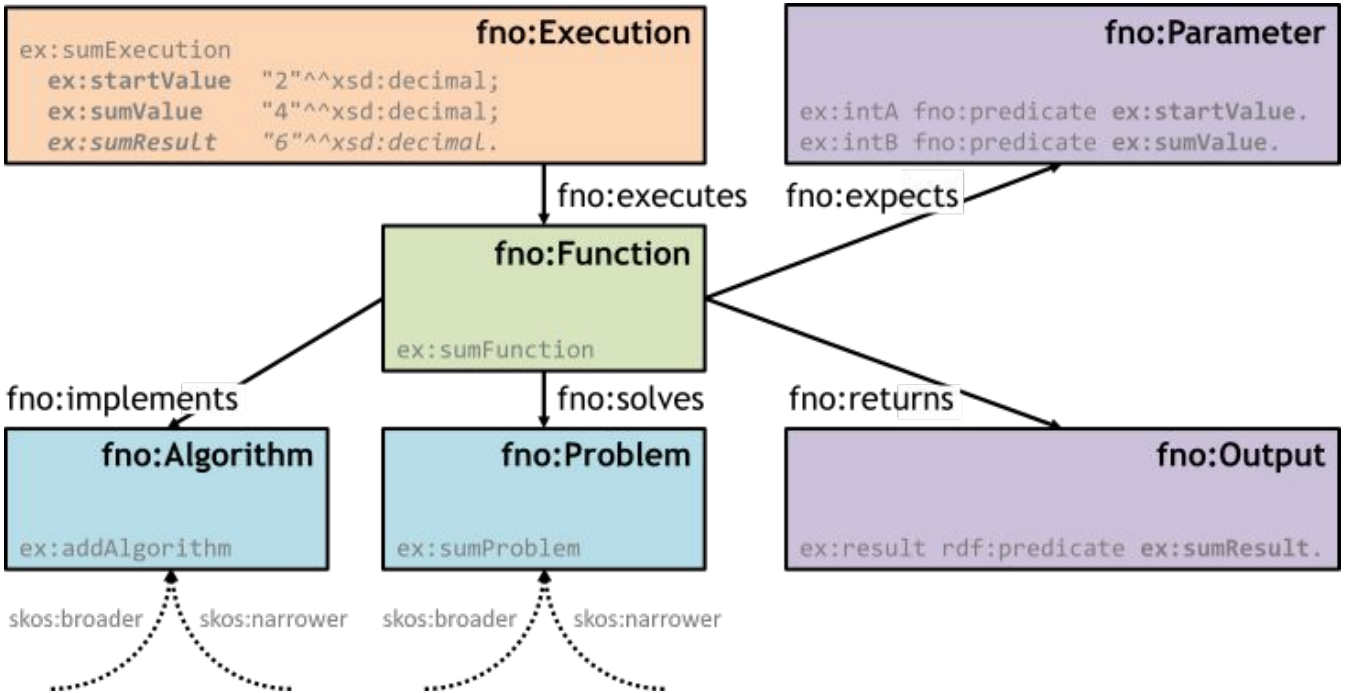
Complex Function example - transforming email address to IRI

“Concatenate “mailto:” with Email field value, but only if it is not empty. Using standard Grel/idlab functions:

```
trueCondition(
    str(array_join("mailto:",?Email)),
    strBoolean(not(isNull(?Email)))
)
```

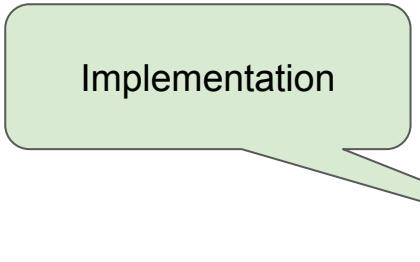
Function ontology

- https://fno.io/spec/



Function libraries

- predefined libraries
  - GrEL functions: http://users.ugent.be/~bjdmeest/function/grel.ttl
  - IDLab functions: https://github.com/FnOio/idlab-functions-java/blob/main/src/main/resources/fno/functions_idlab.ttl
- custom libraries




```
strUtils:abbreviate
      a fno:Function ;
      fno:name "abbreviate" ;
      rdfs:label "abbreviate" ;
      dcterms:description "Abbreviates a String using ellipses. This will
turn (Now is the time for all good men) into (Now is the time for...) if (...)
was defined as the replacement marker" ;
      fno:expects ( strUtils:finalText strUtils:abbrevMaker strUtils:maxWidth
) ;
      fno:returns    ( strUtils:resultStr ) ;
      lib:providedBy [ lib:localLibrary "StringUtils.java" ;
                       lib:class  "org.apache.commons.lang3.StringUtils" ;
                       lib:method "abbreviate" ] .
```

barnard59

barnard59

- one of several libraries defining full pipelines in RDF
  - you can process by RDF parsers
  - store in RDF databases
  - query via SPARQL …

https://github.com/zazuko/barnard59

barnard59

```
@base <http://example.org/pipeline/> .
@prefix code: <https://code.described.at/> .
@prefix p: <https://pipeline.described.at/> .
@prefix fs: <https://barnard59.zazuko.com/operations/core/fs/> .
@prefix n3: <https://barnard59.zazuko.com/operations/formats/n3/> .
@prefix rdf: <https://barnard59.zazuko.com/operations/rdf/> .
@prefix graph-store: <https://barnard59.zazuko.com/operations/graph-store/> .

<vars> p:variable [
           a p:Variable ;
           p:name "graph" ;
           p:value "https://example.org/data"
       ] .

<load> fs:createReadStream ( "inputFile"^^p:VariableName ) .
<parse> n3:parse ( ) .
<setGraph> rdf:setGraph ( "graph"^^p:VariableName ) .
<put> graph-store:put
       [ code:name "endpoint" ; code:value "endpoint"^^p:VariableName ],
       [ code:name "user" ; code:value "user"^^p:VariableName ],
       [ code:name "password" ; code:value "password"^^p:VariableName ] .
<upload> a p:Pipeline ; p:variables <vars> ;
   p:steps
       [ p:stepList ( <load> <parse> <setGraph> <put> )].
```

Examples

https://gitlab.fel.cvut.cz/osw/rml-tutorial