Declarative Construction and Validation of Knowledge Graphs

Half-day tutorial at K-CAP23

Ana Iglesias-Molina[1] and Xuemin Duan[2]

[1]Ontology Engineering Group (OEG) – Universidad Politécnica de Madrid
[2]Declarative Languages and Artificial Intelligence (DTAI) – KU Leuven

About us...

Ana Iglesias-Molina

PhD Student
Ontology Engineering Group (OEG)
Universidad Politécnica de
Madrid, Spain

✉ ana.iglesiasm@upm.es

@_aieme

Xuemin Duan

PhD Student
Declarative Languages and
Artificial Intelligence (DTAI)
KU Leuven, Belgium

✉ xuemin.duan@kuleuven.be

@xueminduan

Agenda

- 13:00 – 13:10 Introduction
- 13:10 – 15:00 Declarative Knowledge Graph Construction
- 15:00 – 15:30 Break
- 15:30 – 17:20 Declarative Knowledge Graph Validation
- 17:20 – 17:30 Conclusions

Running example

- In both sessions (construction and validation) we will have a hands-on exercise with a running example
- Data about aircrafts, airlines and aircraft models
- CSV file

Knowledge Graph Construction

13:10 - 15:00

Outline:

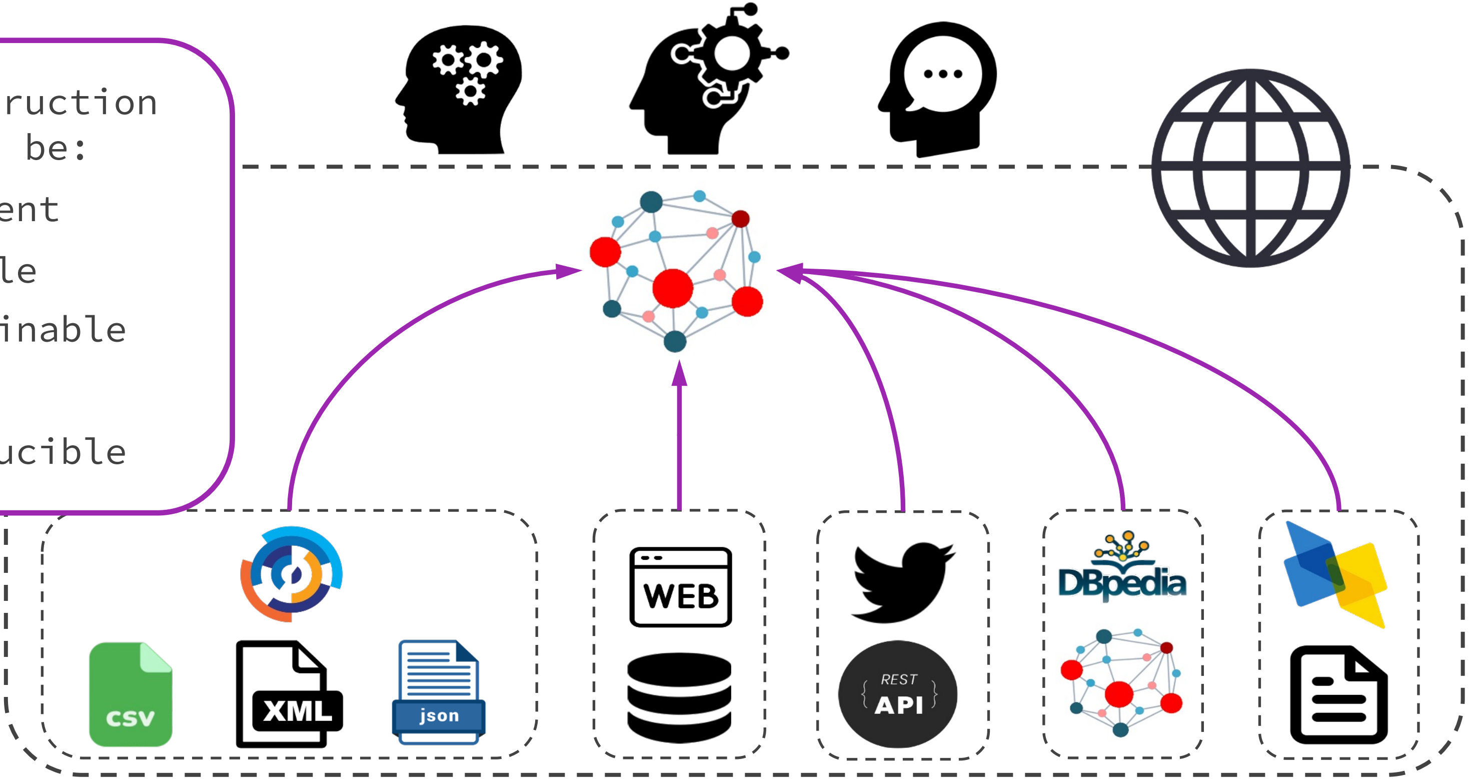
- Background
- How to write mappings
- KGC engines
- Hands-on exercise
- The new RML release

Background

Knowledge Graph Construction

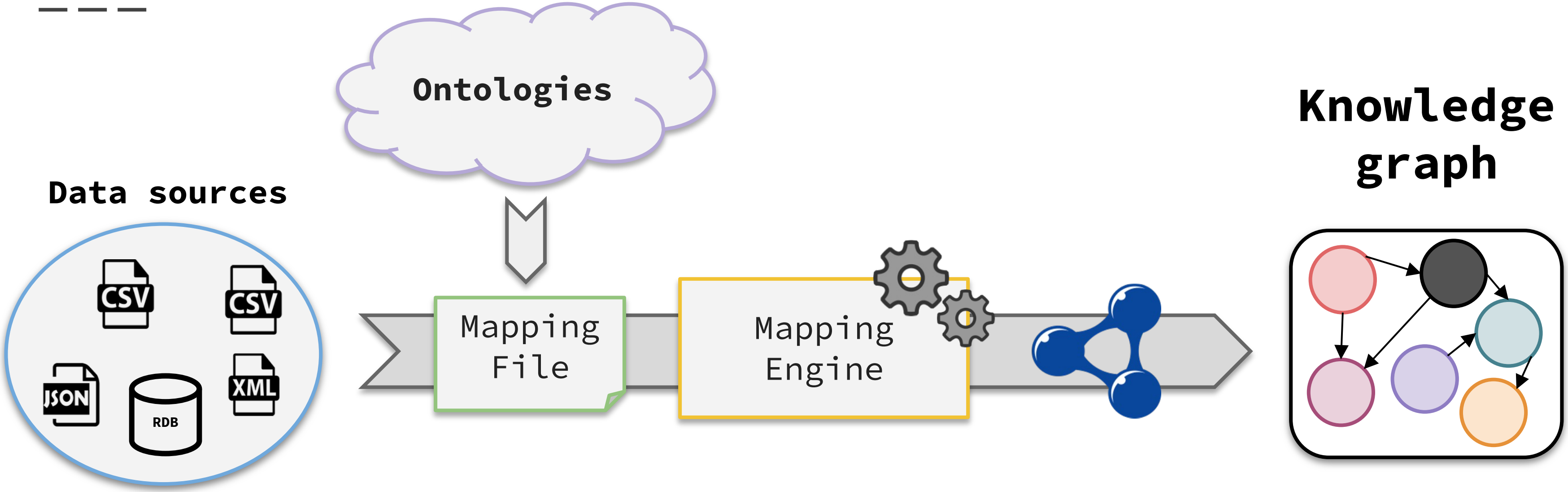
KG Construction needs to be:

- Efficient
- Scalable
- Maintainable
- Robust
- Reproducible



Adapted from: David Chaves-Fraga “Human-Friendly RDF Graph Construction: Which one do you choose?” ICWE 2023

Declarative Knowledge Graph Construction



Ontologies

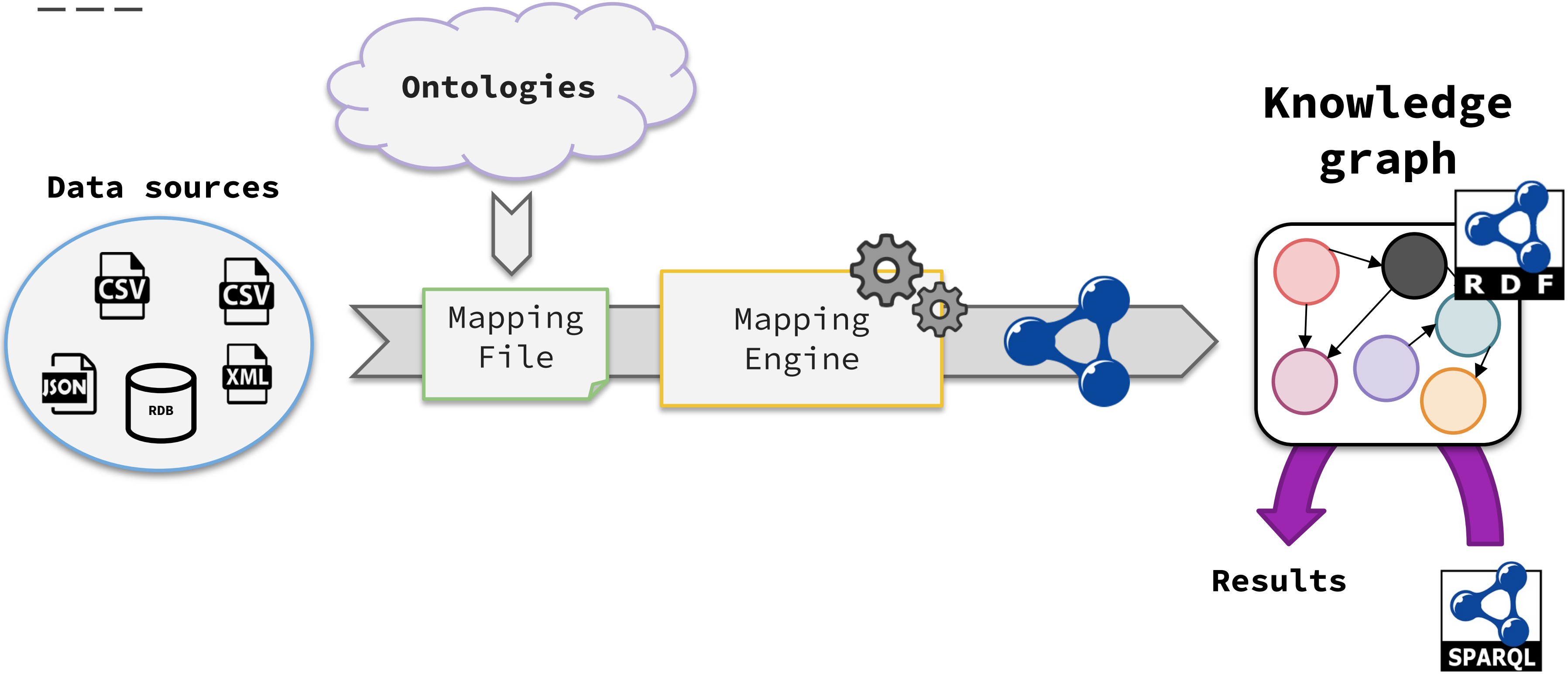
Data sources

Mapping File

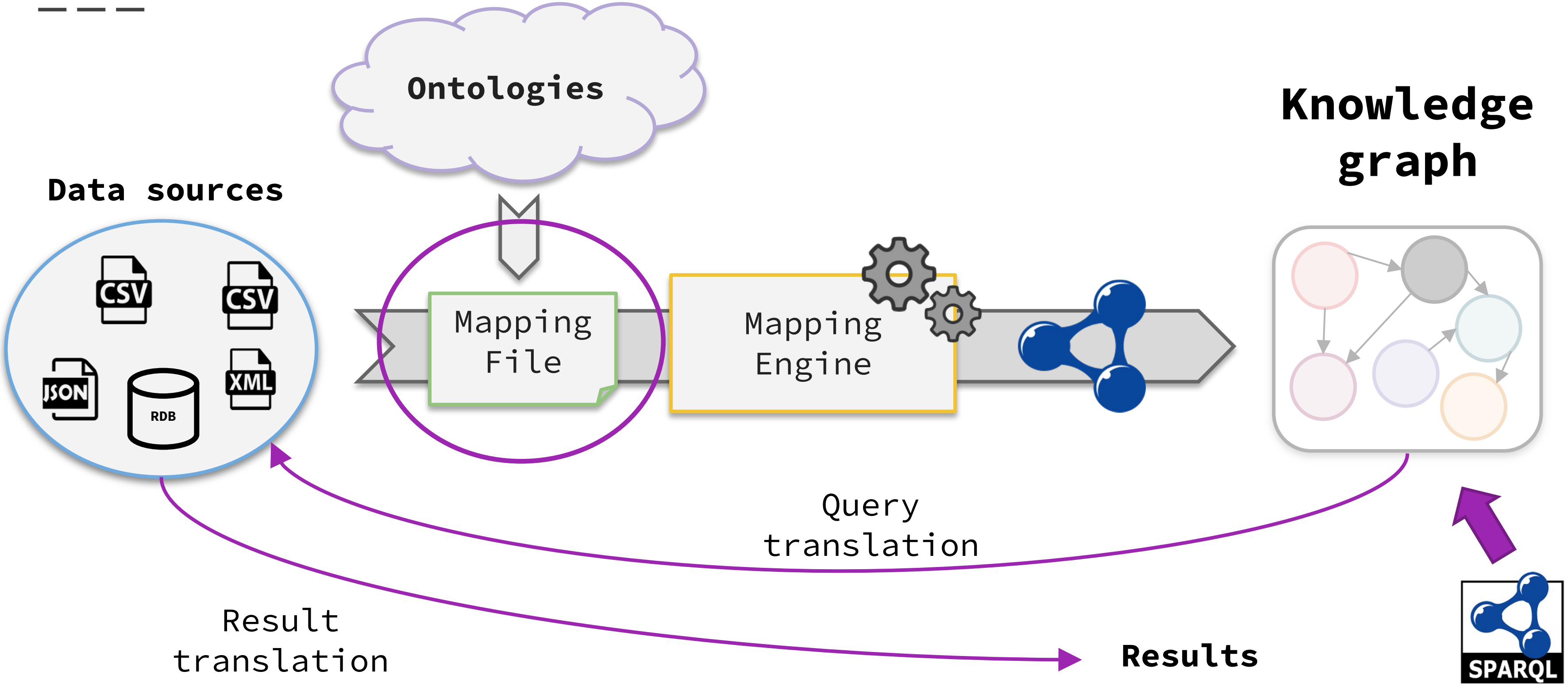
Mapping Engine

Knowledge graph

Declarative KGC: Materialization



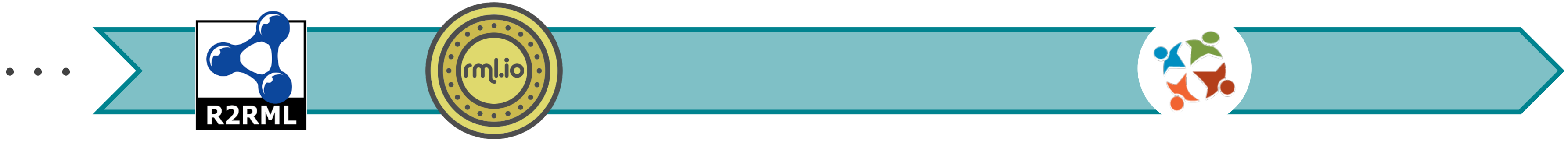
Declarative KGC: Virtualization



Mapping languages

- Allow specifying **transformation rules** from heterogeneous data sources into RDF graphs
  - **Data source** and **access** description
  - **Triple** generation:
    - Subject, Predicate, Object
    - Language tags and datatypes
  - Additional:
    - Join conditions
    - Data transformation functions
- Many different approaches throughout the years
  - Different **features**
  - Variety of **implementations**

The history so far



XLWrap

D2RQ

R2O

W3C Recommendation
R2RML

RML

KR2RML

xR2RML

FunUL

R2RML-F

TARQL

SPARQL-Generate

CSVW

RML+FnO

D-REPR

ShExML

Knowledge Graph
Construction
W3C Community Group

SPARQL-Anything

RML Fields

RML-star

RML Target

RML

2012

2015

2020

2023

The RML ecosystem

Compliant systems

- RMLMapper
- RocketRML
- Chimera
- Morph-KGC
- SDM-RDFizer
- CARML
- RDF Processing Toolkit
- Mapeathor
- MEL
- Excel in RML
- Matey
- Yatter
- Spread2RML
- Dragoman
- RMLEditor
- YARRRML-parser

Interoperability

- SPARQL-Generate
- ShExML
- Helio
- YARRRML
- XRM

Benchmarks

- GTFS-Madrid Bench
- SDM-Genomic benchmark
- LUBM4OBDA

Use cases

- National and European Projects
- European Commission (ERA, PPDS)
- Companies (Orange, BASF, REALE)

Mapping example

—

RML

Data sources

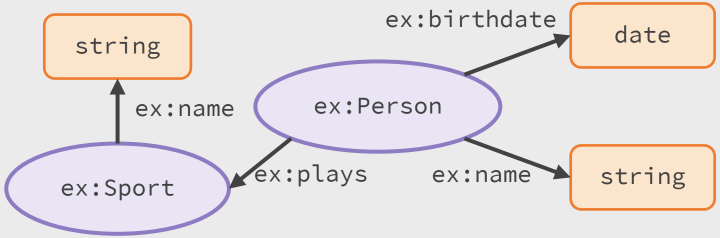
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

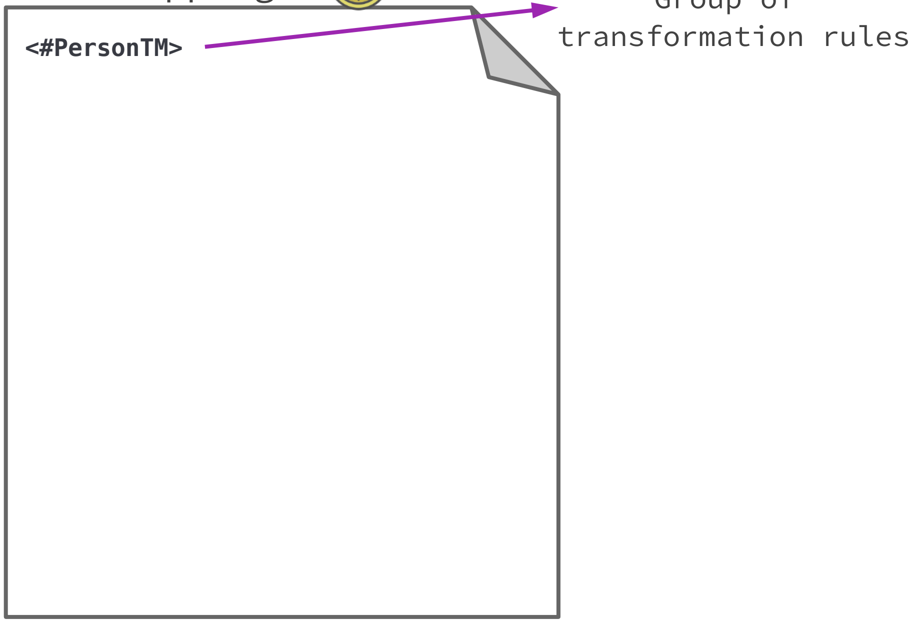
ID	Sport
1	Ice Skating
2	Rugby

Ontology



Mapping

Output Knowledge Graph



<#PersonTM>

Group of transformation rules

Data sources

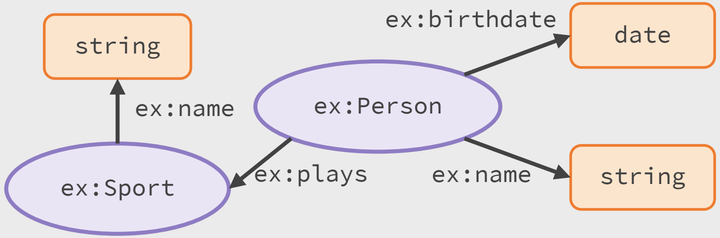
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

Ontology



Mapping

```
<#PersonTM>
rml:logicalSource [
  rml:source ”people.csv" ;
  rml:referenceFormulation ql:CSV ];
```

Output Knowledge Graph

Data sources

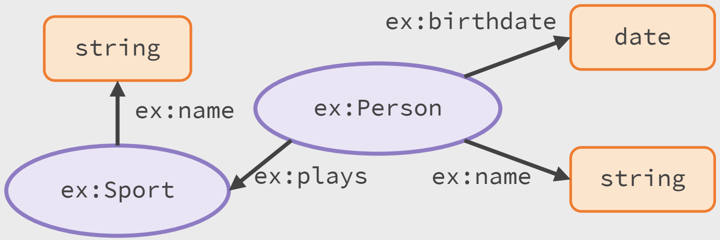
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

Ontology



Mapping

```
<#PersonTM>
rml:logicalSource [
  rml:source ”people.csv" ;
  rml:referenceFormulation ql:CSV ];
rr:subjectMap [
  rr:template "Person/{ID}";
  rr:class ex:Person ];
```

Output Knowledge Graph

<Person/1>
ex:Person
<Person/2>
ex:Person

Data sources

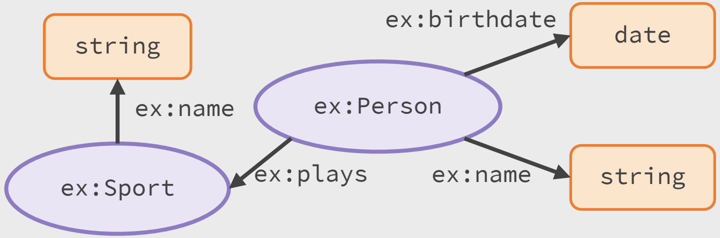
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

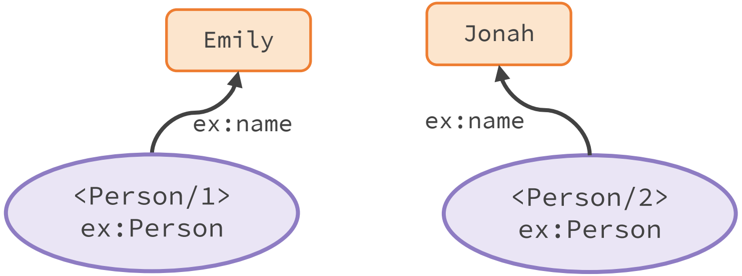
Ontology



Mapping

```
<#PersonTM>
rml:logicalSource [
  rml:source ”people.csv" ;
  rml:referenceFormulation ql:CSV ];
rr:subjectMap [
  rr:template "Person/{ID}";
  rr:class ex:Person ];
rr:predicateObjectMap [
  rr:predicate ex:name;
  rr:objectMap [
    rml:reference ”Name”] ];
```

Output Knowledge Graph



Data sources

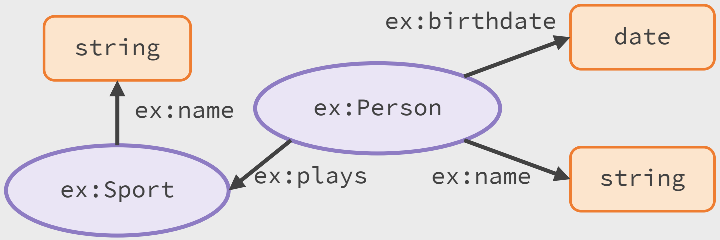
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

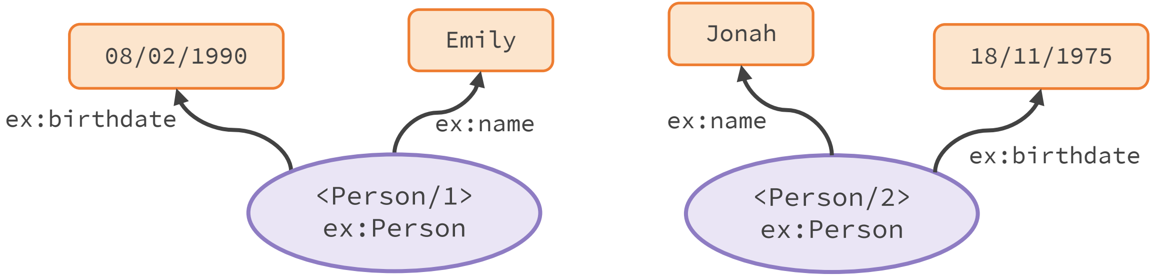
Ontology

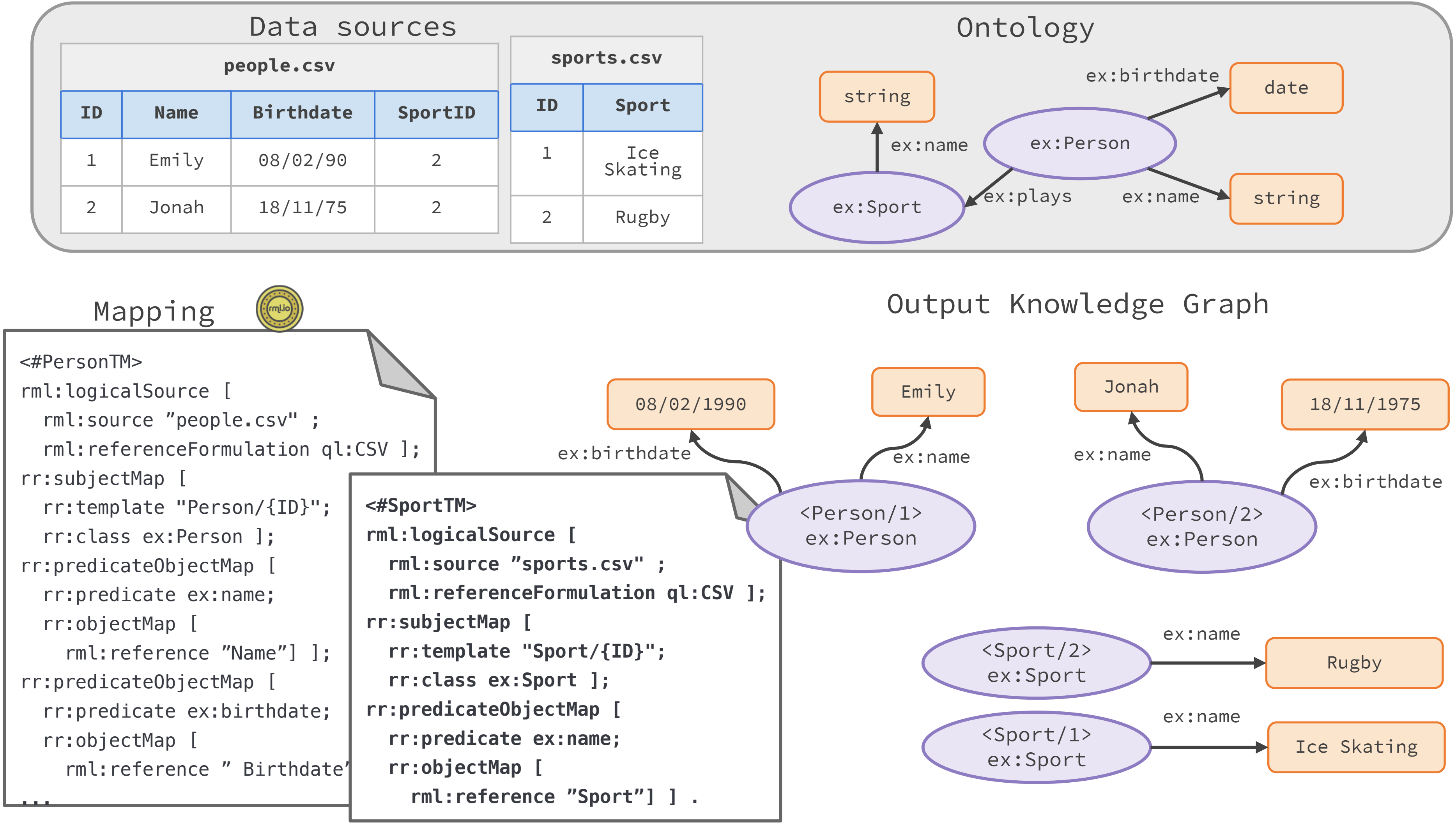


Mapping

```
<#PersonTM>
rml:logicalSource [
  rml:source ”people.csv" ;
  rml:referenceFormulation ql:CSV ];
rr:subjectMap [
  rr:template "Person/{ID}";
  rr:class ex:Person ];
rr:predicateObjectMap [
  rr:predicate ex:name;
  rr:objectMap [
    rml:reference ”Name”] ];
rr:predicateObjectMap [
  rr:predicate ex:birthdate;
  rr:objectMap [
    rml:reference ” Birthdate”] ;
...
```

Output Knowledge Graph





Data sources

people.csv			
ID	**Name**	**Birthdate**	**SportID**
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv	
ID	**Sport**
1	Ice Skating
2	Rugby

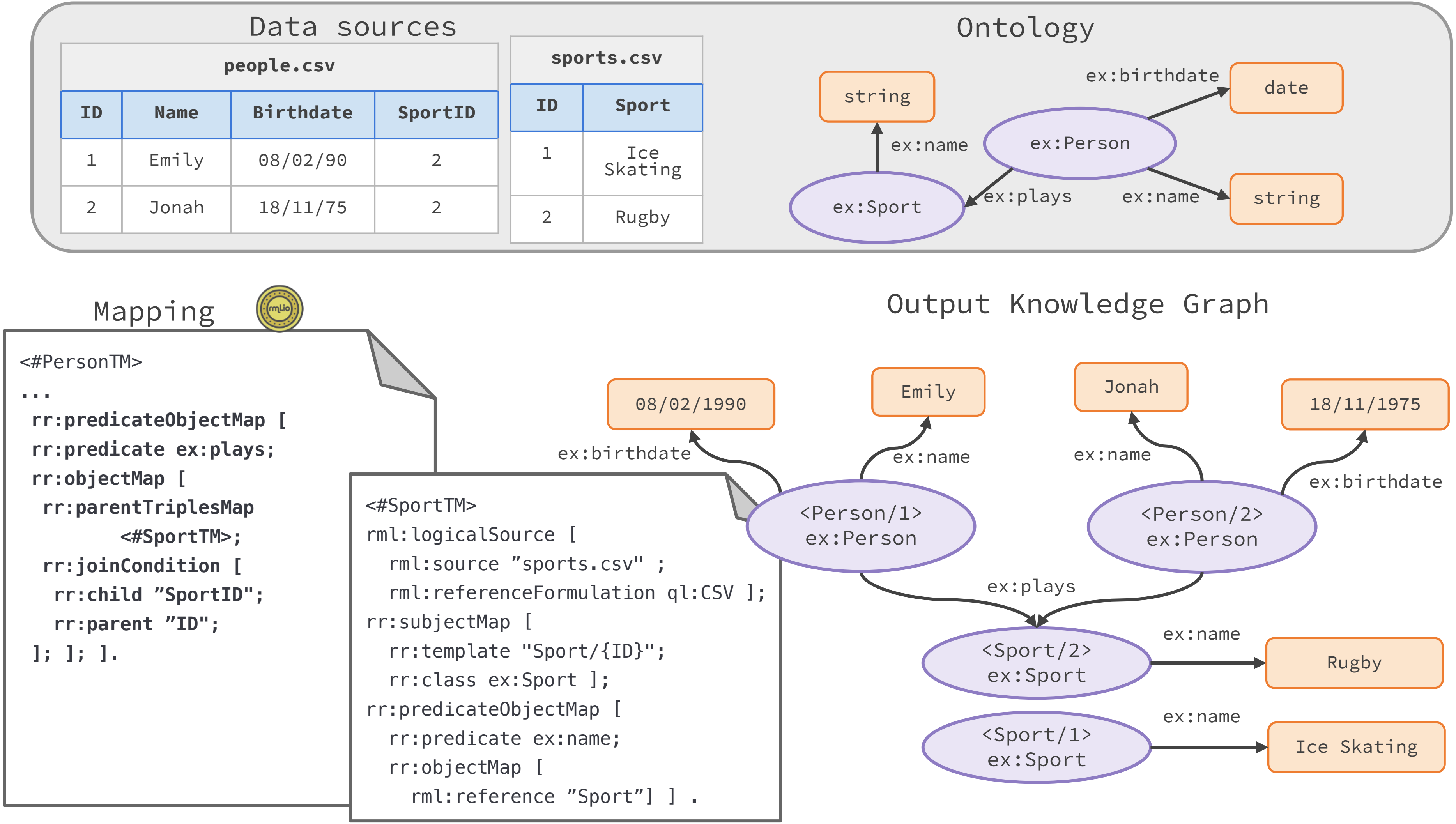
Ontology

Mapping

```
<#PersonTM>
rml:logicalSource [
  rml:source ”people.csv” ;
  rml:referenceFormulation ql:CSV ];
rr:subjectMap [
  rr:template "Person/{ID}";
  rr:class ex:Person ];
rr:predicateObjectMap [
  rr:predicate ex:name;
  rr:objectMap [
    rml:reference ”Name”] ];
rr:predicateObjectMap [
  rr:predicate ex:birthdate;
  rr:objectMap [
    rml:reference ” Birthdate’
...
```

```
<#SportTM>
rml:logicalSource [
  rml:source ”sports.csv” ;
  rml:referenceFormulation ql:CSV ];
rr:subjectMap [
  rr:template "Sport/{ID}";
  rr:class ex:Sport ];
rr:predicateObjectMap [
  rr:predicate ex:name;
  rr:objectMap [
    rml:reference ”Sport”] ] .
```

Output Knowledge Graph



Data sources

people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

Mapping

```
<#PersonTM>
...
 rr:predicateObjectMap [
 rr:predicate ex:plays;
 rr:objectMap [
  rr:parentTriplesMap
         <#SportTM>;
  rr:joinCondition [
   rr:child ”SportID";
   rr:parent ”ID";
 ]; ]; ].
```

```
<#SportTM>
rml:logicalSource [
  rml:source ”sports.csv" ;
  rml:referenceFormulation ql:CSV ];
rr:subjectMap [
  rr:template "Sport/{ID}";
  rr:class ex:Sport ];
rr:predicateObjectMap [
  rr:predicate ex:name;
  rr:objectMap [
    rml:reference ”Sport”] ] .
```

User-friendly approaches to write mappings

User-friendly approaches to write mappings

- [R2]RML mappings are written as Turtle files:
  - Risk of syntax errors
  - Too many language constructs
- Approaches developed to aid the mapping writing:
  - User-friendly syntax:
    - YARRRML ➡️ RML
    - XRM ➡️ R2RML, CSVW, RML (Zazuko)
    - SMS2 ➡️ R2RML (Stardog)
    - obda ➡️ R2RML (Ontop)
  - Editors
    - RMLEditor ➡️ RML
    - Mapeathor ➡️ R2RML, RML, YARRRML

YARRRML

User-friendly serialization for RML based on YAML syntax:

- Compact syntax
- Widely adopted in real-life projects
  - Entity Reconciliation API on Google Specification
  - https://w3id.org/kg-construct/yarrrml
- Online service
  - https://rml.io/yarrrml/matey/

Mapping example

—

YARRRML

Data sources

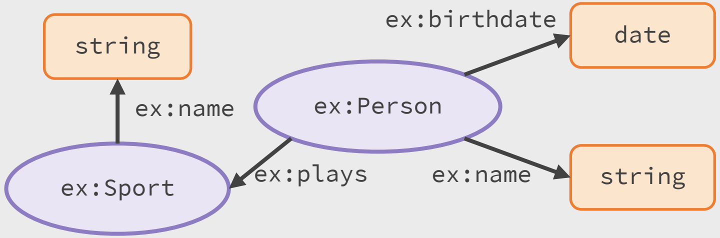
people.csv

ID	**Name**	**Birthdate**	**SportID**
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	**Sport**
1	Ice Skating
2	Rugby

Ontology



Mapping

```
mappings:
 **PERSON**:
```

Group of
transformation rules

Output Knowledge Graph

Data sources

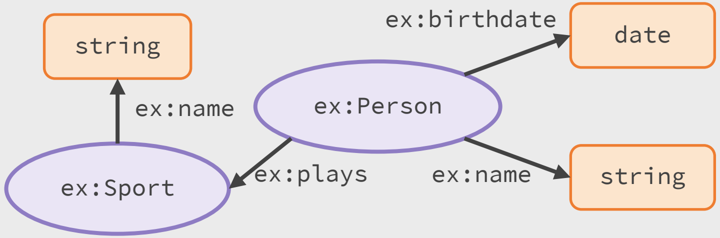
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

Ontology



Mapping

```
mappings:
 PERSON:
  sources:
   - ['data/people.csv~csv']
```

Output Knowledge Graph

Data sources

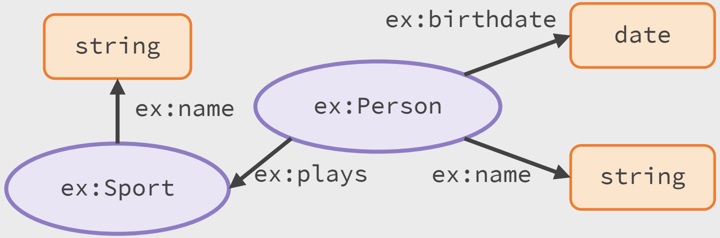
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

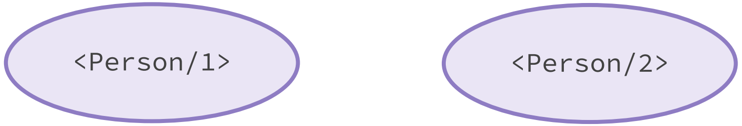
Ontology



Mapping

```
mappings:
 PERSON:
  sources:
   - ['data/people.csv~csv']
  s: http://ex.com/Person/$(ID)
```

Output Knowledge Graph



Data sources

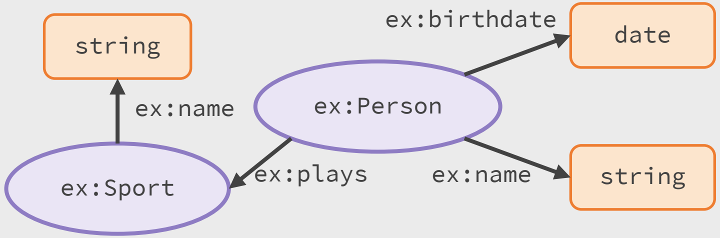
people.csv

ID	**Name**	**Birthdate**	**SportID**
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	**Sport**
1	Ice Skating
2	Rugby

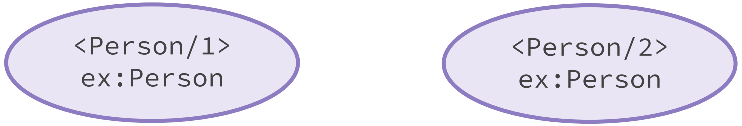
Ontology



Mapping

```
mappings:
 PERSON:
  sources:
   - ['data/people.csv~csv']
  s: http://ex.com/Person/$(ID)
  po:
   - [a, ex:Person]
```

Output Knowledge Graph



Data sources

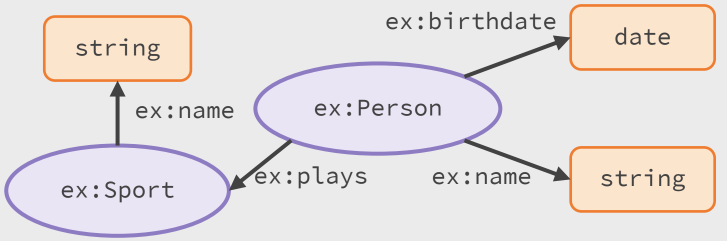
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

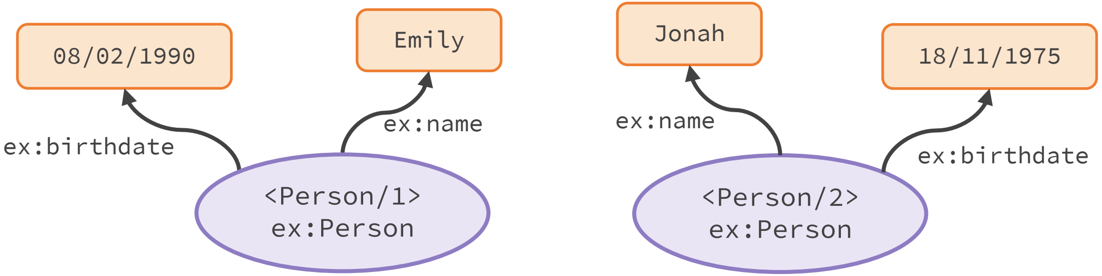
Ontology



Mapping

```
mappings:
 PERSON:
  sources:
   - ['data/people.csv~csv']
  s: http://ex.com/Person/$(ID)
  po:
   - [a, ex:Person]
   - [ex:name, $(Name)]
   - [ex:birthdate, $(Birthdate)]
```

Output Knowledge Graph



Data sources

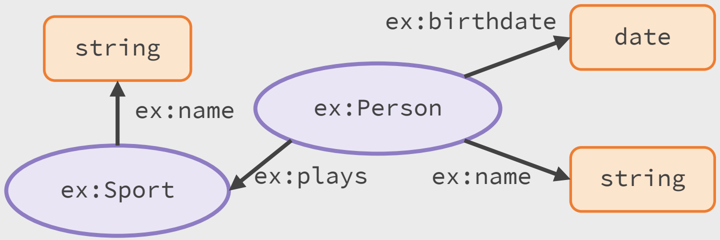
people.csv

ID	Name	Birthdate	SportID
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv

ID	Sport
1	Ice Skating
2	Rugby

Ontology

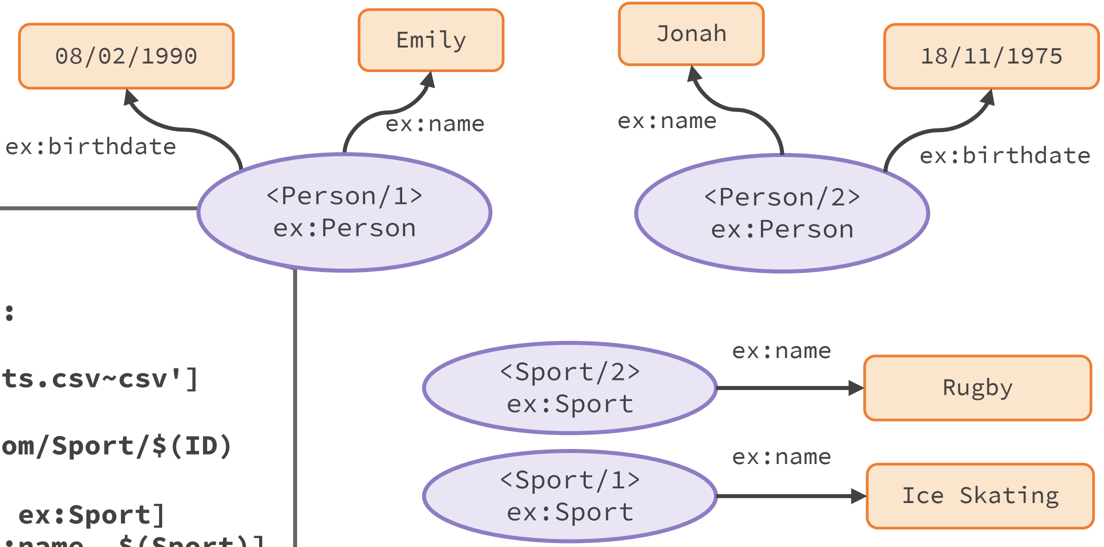


Mapping

```
mappings:
 PERSON:
  sources:
   - ['data/people.csv~csv']
  s: http://ex.com/Person/$(ID)
  po:
   - [a, ex:Person]
   - [ex:name, $(Name)]
   - [ex:birthdate, $(Birthdate)
```

```
mappings:
 SPORT:
     sources:
       -
['data/sports.csv~csv']
     s:
http://ex.com/Sport/$(ID)
     po:
       - [a, ex:Sport]
       - [ex:name, $(Sport)]
```

Output Knowledge Graph

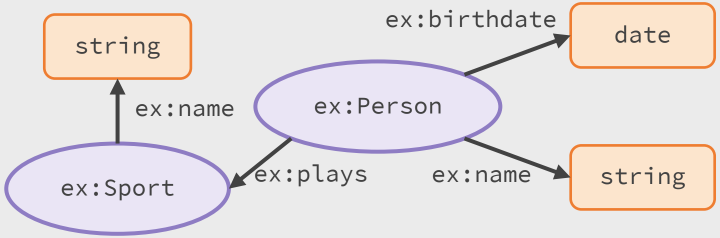


Data sources

people.csv			
ID	**Name**	**Birthdate**	**SportID**
1	Emily	08/02/90	2
2	Jonah	18/11/75	2

sports.csv	
ID	**Sport**
1	Ice Skating
2	Rugby

Ontology

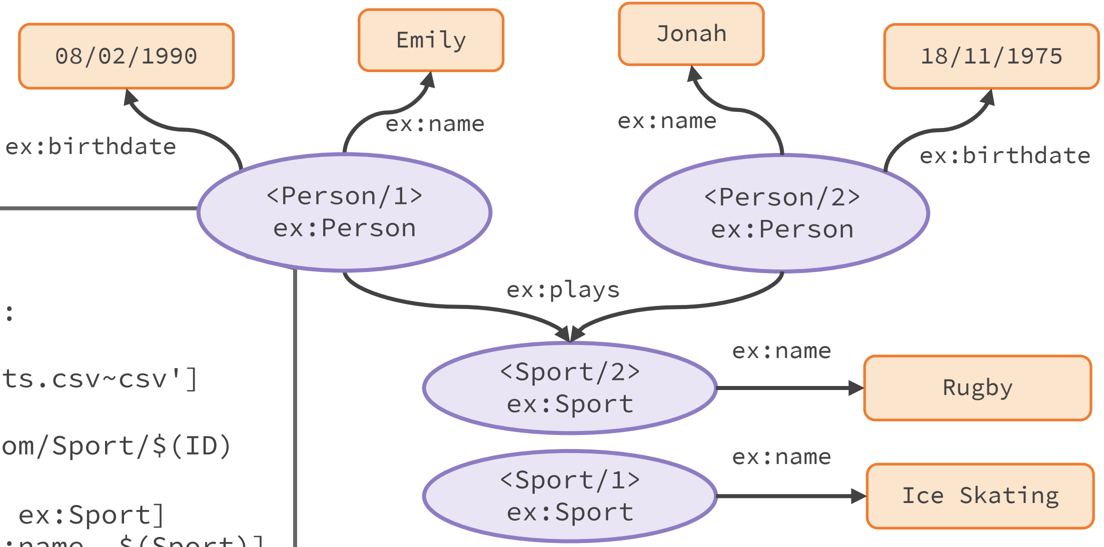


Mapping

```
mappings:
 PERSON:
  sources:
   - ['data/people.csv~csv']
  s: http://ex.com/Person/$(ID)
  po:
   - [a, ex:Person]
   - [ex:name, $(Name)]
   - [ex:birthdate, $(Birthdate)
   - p: ex:plays
     o:
      - mapping: SPORT
        condition:
          function: equal
          parameters:
           - [str1, $(SportID)]
           - [str2, $(ID)]
```

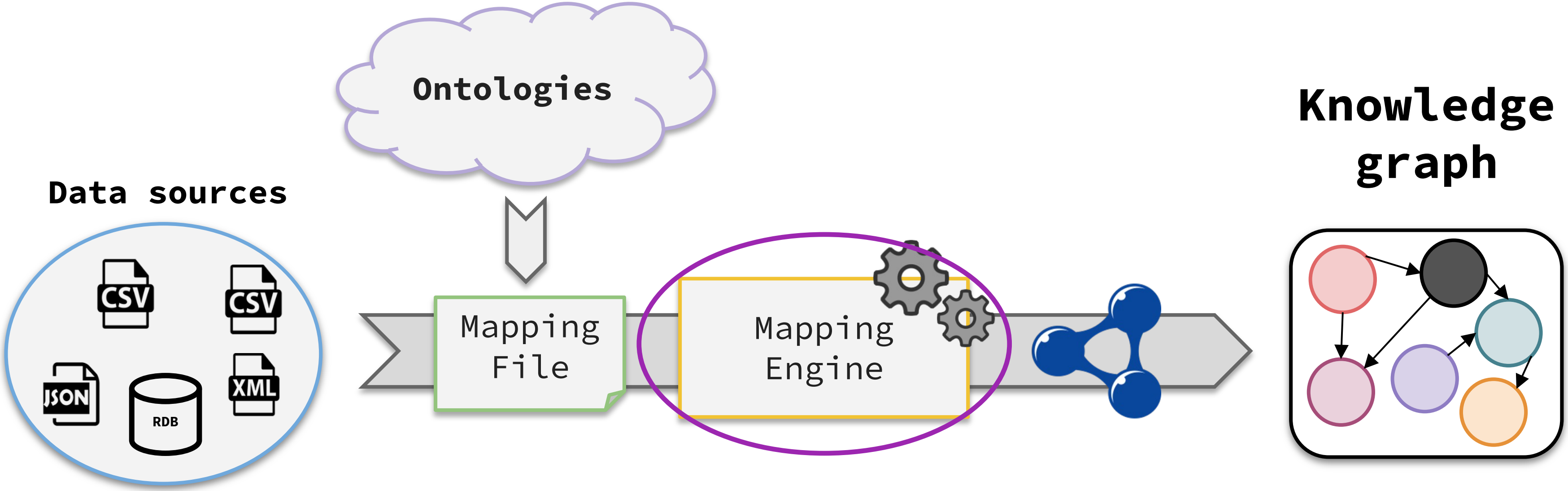
```
mappings:
 SPORT:
    sources:
      -
['data/sports.csv~csv']
    s:
http://ex.com/Sport/$(ID)
    po:
      - [a, ex:Sport]
      - [ex:name, $(Sport)]
```

Output Knowledge Graph



RML compliant systems

Declarative Knowledge Graph Construction



RMLMapper

Download at: https://github.com/RMLio/rmlmapper-java

(jar files at releases: https://github.com/RMLio/rmlmapper-java/releases)

Features:

- Good parser in terms on Data Quality
- Java based
- Really simple to run (-d is for removing duplicates):

  ```
  java -jar rmlmapper.jar -m mapping.rml -o output.nt -d
  ```

- It produces empty objects for Literals (s p “”^^xsd:string) that have to be removed after the transformation
- Not very efficient when you have many joins or duplicates

SDM-RDFizer

Download at: https://pypi.org/project/rdfizer/ (pip install rdfizer)

Github website: https://github.com/SDM-TIB/SDM-RDFizer

Features:

- Very efficient for large datasets
- Efficient execution of joins and duplicates removal
- Python based
- Config file needed to be run
- It uses RDFlib

RocketRML

Download at: https://www.npmjs.com/package/rocketrml (npm install rocketrml)

Github website: https://github.com/semantifyit/RocketRML

Features:

- Very efficient for small and medium data (more than SDM-RDFizer)
- Node based
- Optimizations for XML files
- A bit difficult to run (index.js has to be programmed)
- It uses RDFlib (but in JS)

Matey - YARRRML parser

Use at: https://rml.io/yarrrml/matey/

Github website: https://github.com/RMLio/yarrrml-parser
Can be used together with a YAML validator: http://www.yamllint.com/

Features:

- Translate YARRRML to RML
- Node based
- Can be installed locally or used through the website
- It uses RDFlib (but in JS)

Yatter

Download at: https://pypi.org/project/yatter/ (pip install yatter)

Github website: https://github.com/oeg-upm/yatter

Features:

- Easy to read outputs (following Turtle RDF syntax with BN)
- Translation from YARRRML to R2RML/RML and viceversa
- Continuous integration/development of test-cases
- Code coverage of ~85% of with the test-cases
- PyPi module (easy to install)
- Follows Open Science good practices (GitHub + Zenodo)

Morph-KGC

Download at https://github.com/oeg-upm/Morph-KGC

Website: https://morph.oeg.fi.upm.es/

Features:

- Python-based construction
- Supports R2RML, RML and YARRRML
- Input data formats:
  - Relational databases: MySQL, PostgreSQL, Oracle, Microsoft SQL Server, MariaDB, SQLite
  - Tabular files: CSV, TSV, Excel, Parquet, Feather, ORC, Stata, SAS, SPSS
  - Hierarchical files: JSON, XML
- Output RDF serializations: N-Triples, N-Quads
- Runs on Linux, Windows and macOS systems
- Optimized to materialize large knowledge graphs
- Multiple configuration options

Running Morph-KGC

Command line execution:

- Installation with PIP
- Takes as input a config file

```
pip install morph_kgc
python -m morph_kgc config.ini
```

config.ini

```
[CONFIGURATION]
output_file=result

[DataSource1]
mappings=/Users/user/example/mappings/mapping.rml.ttl
```

Hands-on exercise

Hands-on exercise

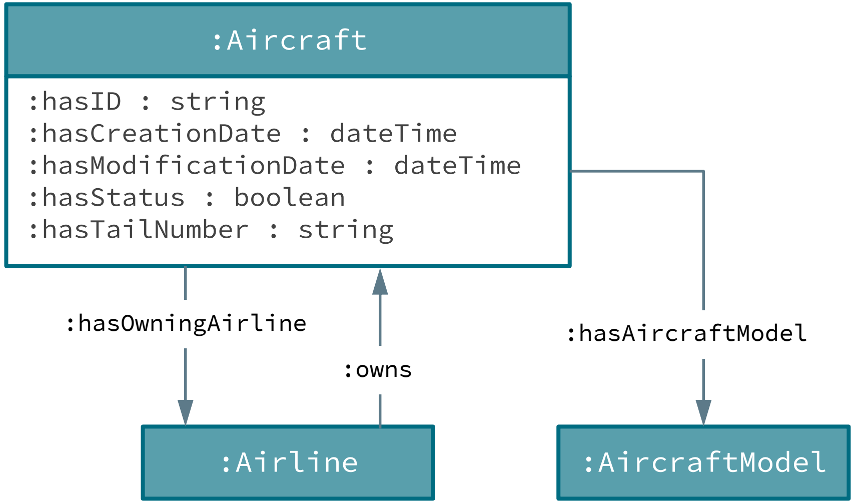
- Creating a KG from a CSV file using YARRRML

Hands-on exercise

- Creating a KG from a CSV file using YARRRML:
  1. Access the resources in: https://bit.ly/3T3Ysr0
  2. Create the mapping completing the file ‘yarrrml_template.yml’
  3. Can be tested using Matey or Yatter
  4. Run the mapping and the data with Morph-KGC to create the KG

Hands-on exercise

- Creating a KG from a CSV file using YARRRML



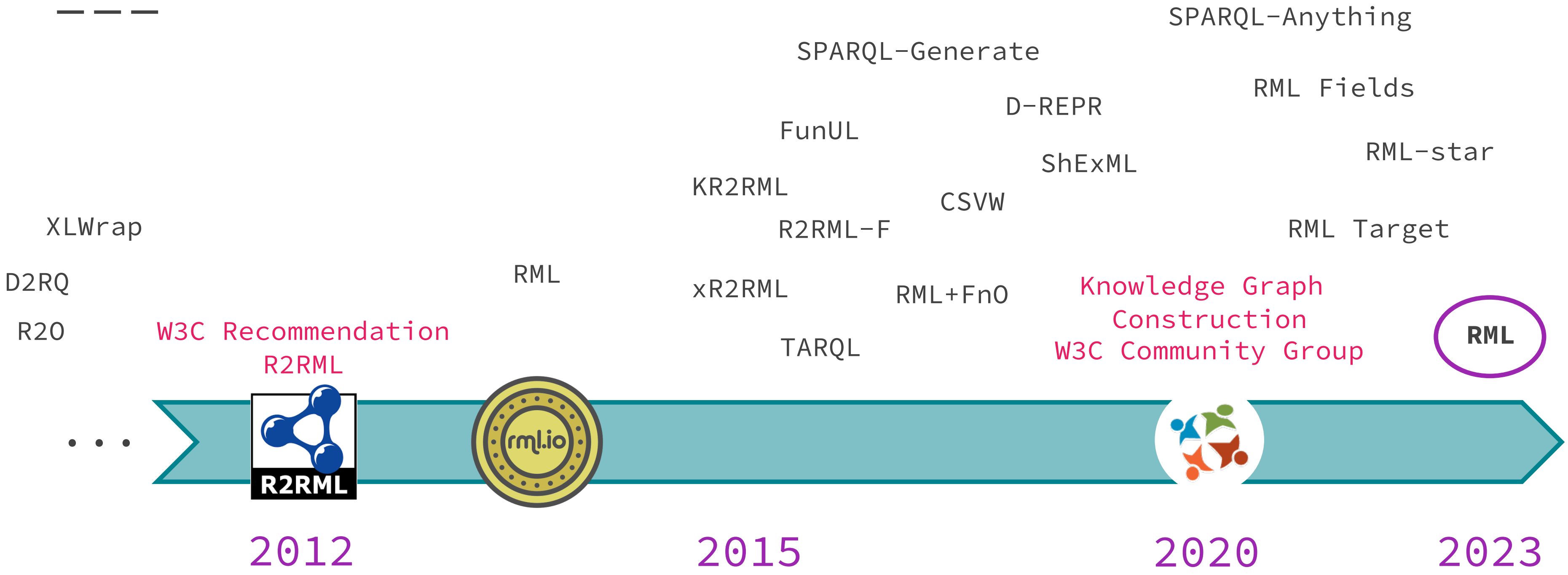
csv **sfo-aircraft-tail-numbers-and-models.csv**

- aircraft_id
- aircraft_model
- airline
- creation_date
- modification_date
- status
- tail_number

Resources → https://bit.ly/3T3Ysr0

The new RML release

The history so far



…

D2RQ

XLWrap

R2O

W3C Recommendation
R2RML

R2RML

2012

RML

rml.io

KR2RML

xR2RML

R2RML-F

FunUL

SPARQL-Generate

TARQL

2015

CSVW

RML+FnO

D-REPR

ShExML

Knowledge Graph
Construction
W3C Community Group

2020

SPARQL-Anything

RML Fields

RML-star

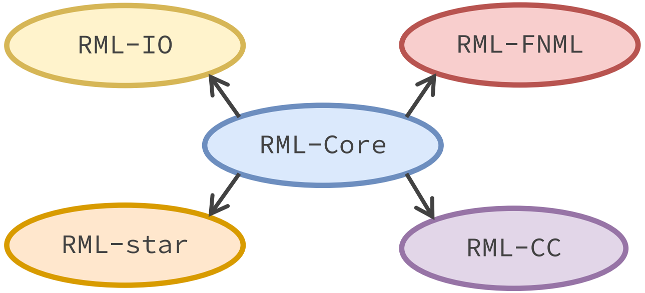
RML Target

RML

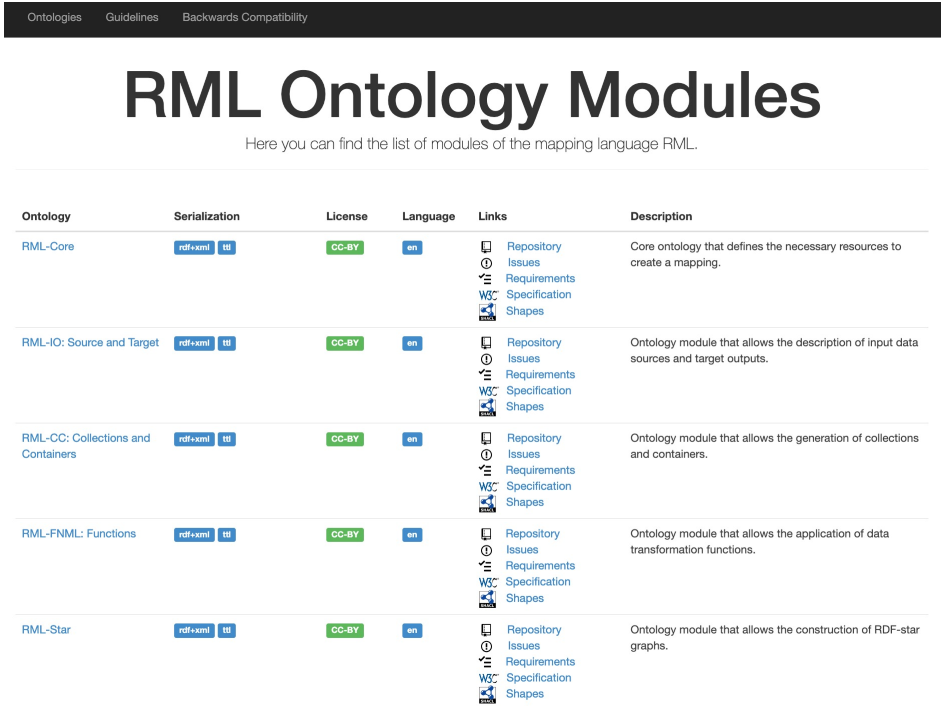
2023

The RML Ontology: Overview

- **Reference** ontology specification addressing **identified challenges**
- **Modular** design
- Developed following the **Linked Open Terms (LOT) methodology**
- Maintaining **backwards compatibility** with R2RML
  - **Semantic correspondences** with previous resources
- All resources in all modules use **same namespace and prefix**
  - **rml:** https://w3id.org/rml/



The RML Ontology: Modules



Ontologies Guidelines Backwards Compatibility

RML Ontology Modules

Here you can find the list of modules of the mapping language RML.

Ontology	Serialization	License	Language	Links	Description
RML-Core	rdf+xml ttl	CC-BY	en	Repository Issues Requirements Specification Shapes	Core ontology that defines the necessary resources to create a mapping.
RML-IO: Source and Target	rdf+xml ttl	CC-BY	en	Repository Issues Requirements Specification Shapes	Ontology module that allows the description of input data sources and target outputs.
RML-CC: Collections and Containers	rdf+xml ttl	CC-BY	en	Repository Issues Requirements Specification Shapes	Ontology module that allows the generation of collections and containers.
RML-FNML: Functions	rdf+xml ttl	CC-BY	en	Repository Issues Requirements Specification Shapes	Ontology module that allows the application of data transformation functions.
RML-Star	rdf+xml ttl	CC-BY	en	Repository Issues Requirements Specification Shapes	Ontology module that allows the construction of RDF-star graphs.

Composed of **5 modules:**

- **RML-Core:** schema transformations
- **RML-IO:** source and target data
- **RML-FNML:** data transformation functions
- **RML-CC:** collections and containers
- **RML-star:** RDF-star

w3id.org/rml/portal

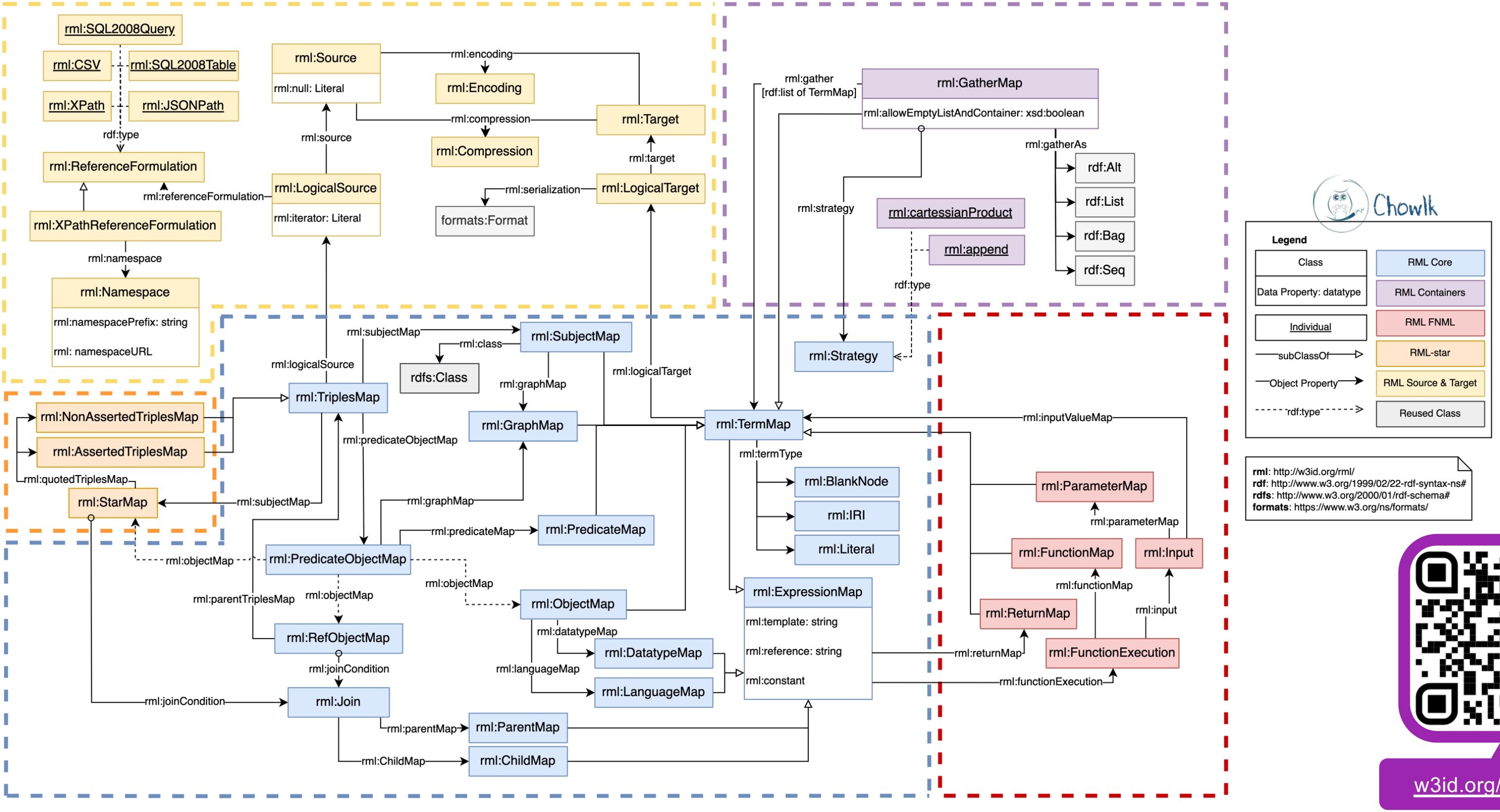
Network of Resources

Each module is composed of:

- **Ontology** w3id.org/rml/core w3id.org/rml/io w3id.org/rml/fnml w3id.org/rml/cc w3id.org/rml/star
  - Implemented in OWL2: Definition of concepts, relationships (w/ domain and range)
  - Available in multiple serializations and a human readable documentation
- **Specification** w3id.org/rml/core/spec w3id.org/rml/io/spec w3id.org/rml/fnml/spec w3id.org/rml/cc/spec w3id.org/rml/star/spec
  - Feature details, examples of use and implementation instructions
- **SHACL shapes** w3id.org/rml/core/shapes w3id.org/rml/io/shapes w3id.org/rml/fnml/shapes w3id.org/rml/cc/shapes w3id.org/rml/star/shapes
  - Restrictions of module constructs
  - Usable to test correctness of mappings
- **Test cases (WIP)**
  - Language coverage by processing engines

w3id.org/rml/portal

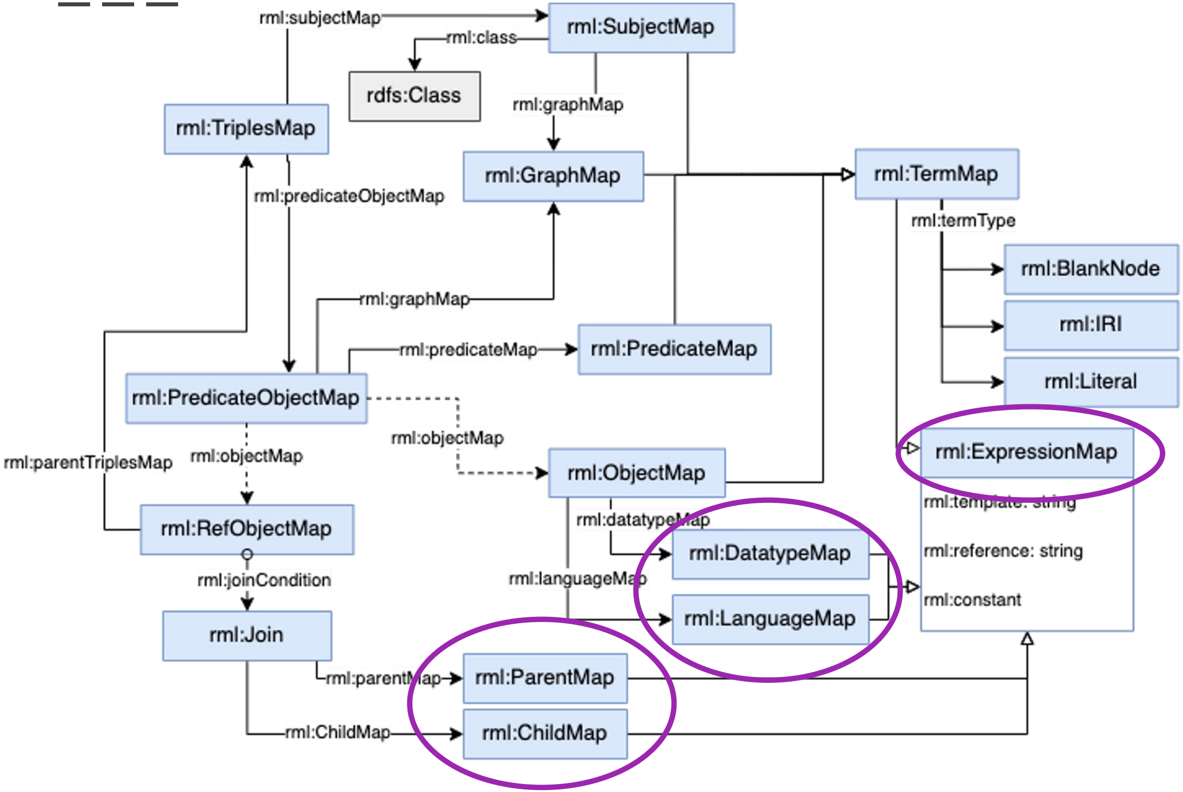
The RML Ontology: Overview



rml: http://w3id.org/rml/
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs: http://www.w3.org/2000/01/rdf-schema#
formats: https://www.w3.org/ns/formats/

w3id.org/rml/portal

RML-Core: Schema transformations



- Maintains **R2RML** basic structure
- **Dynamic** generation of:
  - Language tags
  - Data types
- Increased **flexibility** for join conditions

w3id.org/rml/core

w3id.org/rml/core/spec

w3id.org/rml/core/shapes

RML-Core: Example

```
<#NameTriplesMap> a rml:TriplesMap ;
  rml:logicalSource <#JSONSource> ;
  rml:subjectMap [
    a rml:SubjectMap, rml:ExpressionMap ;
    rml:template "{$.NAME}” ;
    rml:class ex:Athlete ;
    ] ;
  rml:predicateObjectMap [
    a rml:PredicateObjectMap ;
    rml:predicate ex:name ;
    rml:objectMap [
      a rml:ObjectMap, rml:ExpressionMap ;
      rml:reference "$.NAME” ;
      rml:languageMap [
        a rml:LanguageMap;
        rml:reference "$.COUNTRY"]
          ] ] .
```

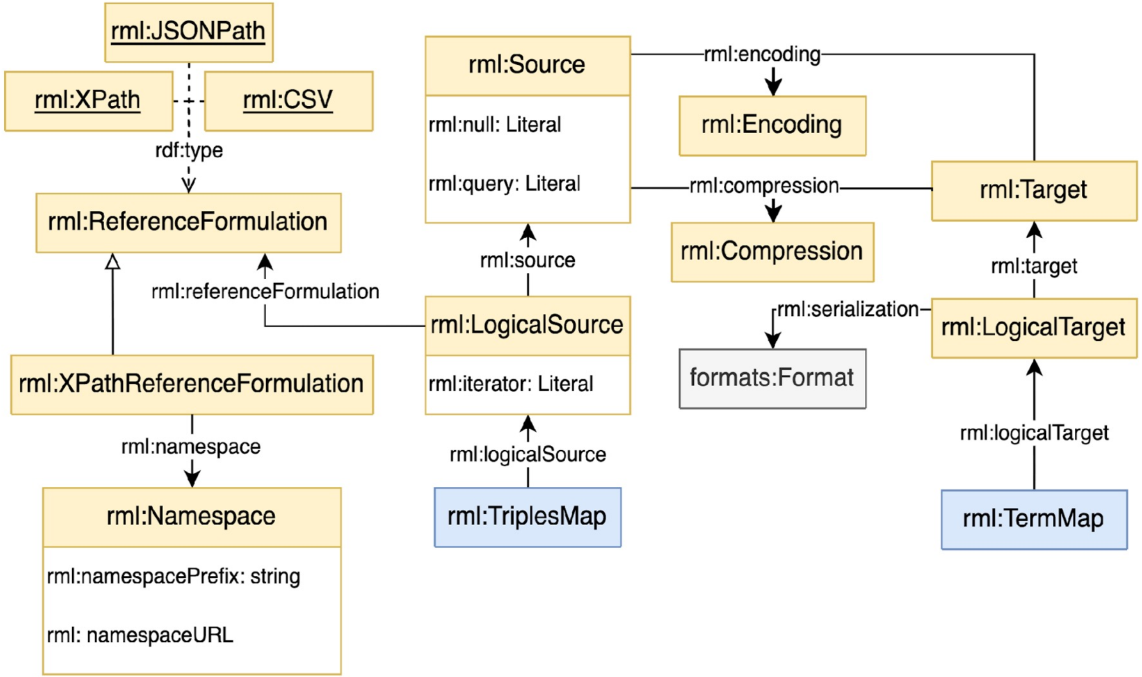
{JSON}

```
[ { "NAME": "Duplantis",
    "RANK": "1",
    "MARK": "6.22",
    "COUNTRY": "sv"
  },{
    "NAME": "Guttormsen",
    "RANK": "2",
    "MARK": "6.00",
    "COUNTRY": "no" } ]
```

RDF

```
:Duplantis a ex:Athlete ;
          ex:name ”Duplantis”@sv .
:Guttormsen a ex:Athlete ;
          ex:name ”Guttormsen”@no .
```

RML-IO: Data source and target



- **Extended input** data source description
- **Output data** description
- Leverage of existing **vocabularies** (SCAT, SPARQL-SD, VoID)

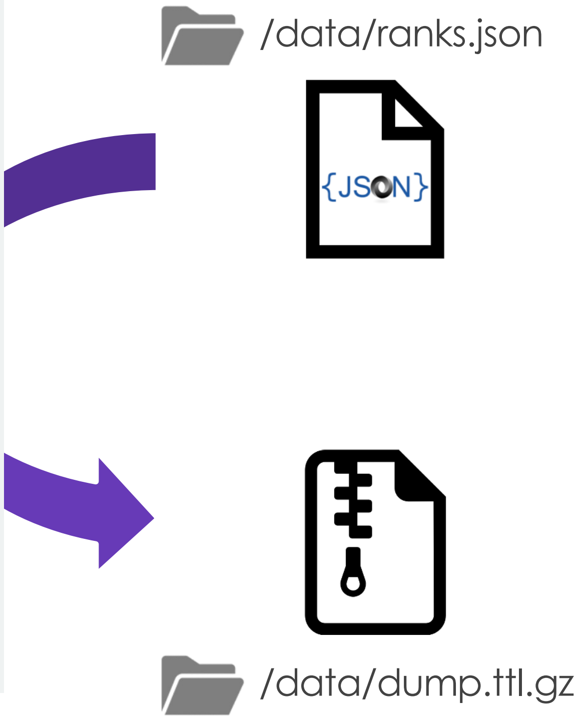
w3id.org/rml/io

w3id.org/rml/io/spec

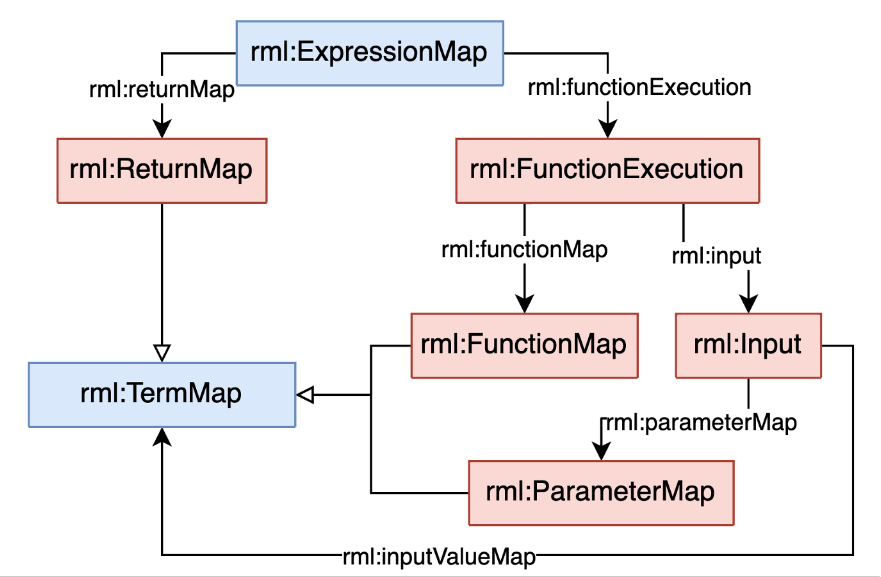
w3id.org/rml/io/shapes

RML-IO: Example

```
<#JSONSource> a rml:LogicalSource ;
  rml:source [ a rml:Source, dcat:Distribution ;
    dcat:accessURL <file:///data/ranks.json> ];
  rml:referenceFormulation rml:JSONPath ;
  rml:iterator "$.[*]"; .

<#FileTarget> a rml:LogicalTarget ;
  rml:target [ a rml:Target, void:Dataset ;
    void:dataDump <file:///data/dump.ttl.gz> ;
    rml:compression rml:gzip ;
    rml:encoding rml:UTF-8 ] ;
  rml:serialization formats:Turtle ; .

<#RankTriplesMap> a rml:TriplesMap ;
  rml:logicalSource <#JSONSource> ;
  rml:subjectMap <#RankSubjectMap> .

<#RankSubjectMap> rml:logicalTarget <#FileTarget>.
```



RML-FNML: Data Transformations



- Refines **RML+FnO** approach
- Reference connector between RML and the **Function Ontology** (FnO)

w3id.org/rml/fnml

w3id.org/rml/fnml/spec

w3id.org/rml/fnml/shapes

RML-FNML: Example

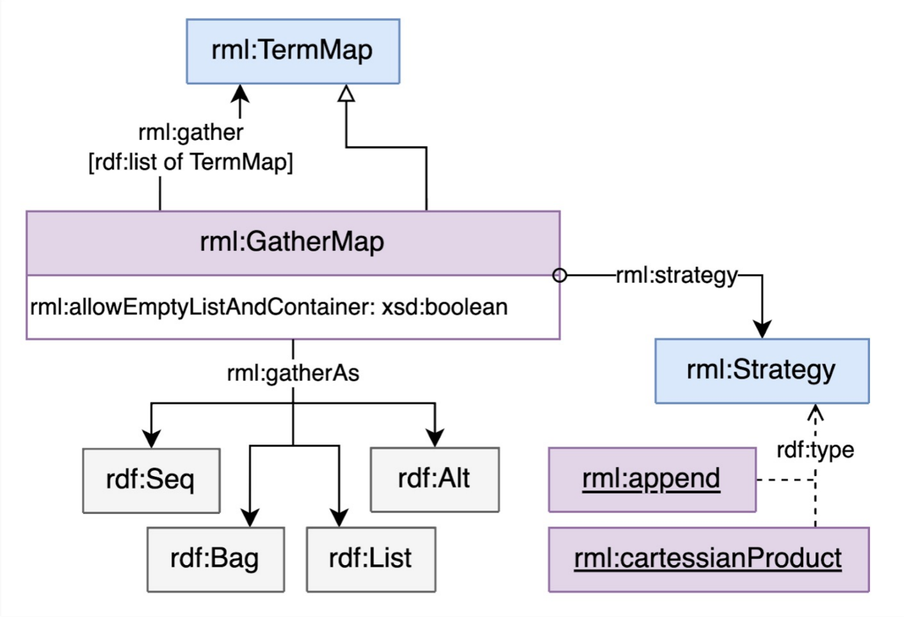
```
<#DateTriplesMap> a rml:TriplesMap ;
 rml:logicalSource <#JSONSource> ;
 rml:subjectMap  <#RankSubjectMap> ;
 rml:predicateObjectMap [
  rml:predicate ex:jumpOnDate ;
  rml:objectMap [
   rml:functionExecution <#Execution> ;
   rml:return ex:dateOut ] ] .

<#Execution> a rml:FunctionExecution ;
 rml:function ex:parseDate ;
 rml:input
     [ a rml:Input ;
  rml:parameter ex:valueParam;
  rml:inputValueMap [
      rml:reference "$.DATE" ] ] ,
  [ a rml:Input ;
  rml:parameter ex:dateFormatParam ;
  rml:inputValueMap [
      rml:constant ”DD-MM-YYYY" ] ] .
```

{JSON}

```
[ { "NAME": "Duplantis",
    "RANK": "1",
    "MARK": "6.22",
    "DATE": 02-25-2023
  },{
    "NAME": "Guttormsen",
    "RANK": "2",
    "MARK": "6.00",
    "DATE": 03-10-2023} ]
```

```
:Duplantis a ex:Athlete ;
 ex:jumpsOnDate “2023-25-02”.
:Guttormsen a ex:Athlete ;
 ex:jumpsOnDate “2023-10-03”.
```

RML-CC: Collections and Containers



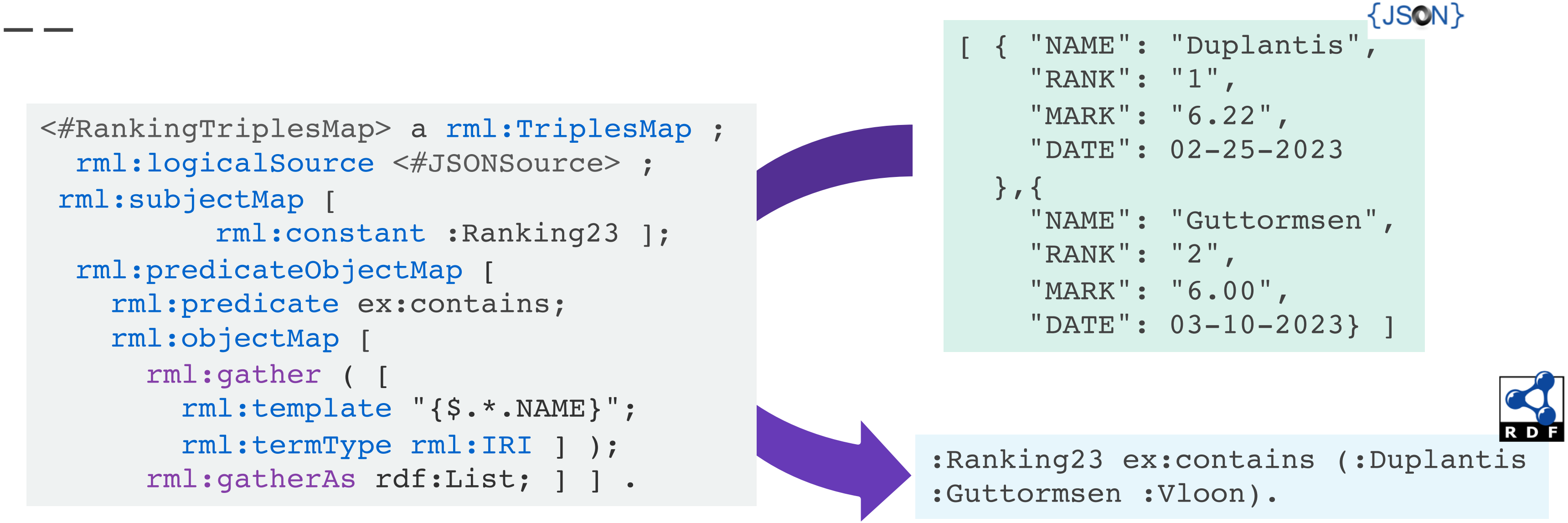
- Introduces generation of **collections and containers**
- Specifies how **gather terms** into a CC and **manage** them: to assign them a IRI or BN, manage empty CC, how the gathering is performed...

w3id.org/rml/fnml

w3id.org/rml/fnml/spec

w3id.org/rml/fnml/shapes

RML-CC: Example

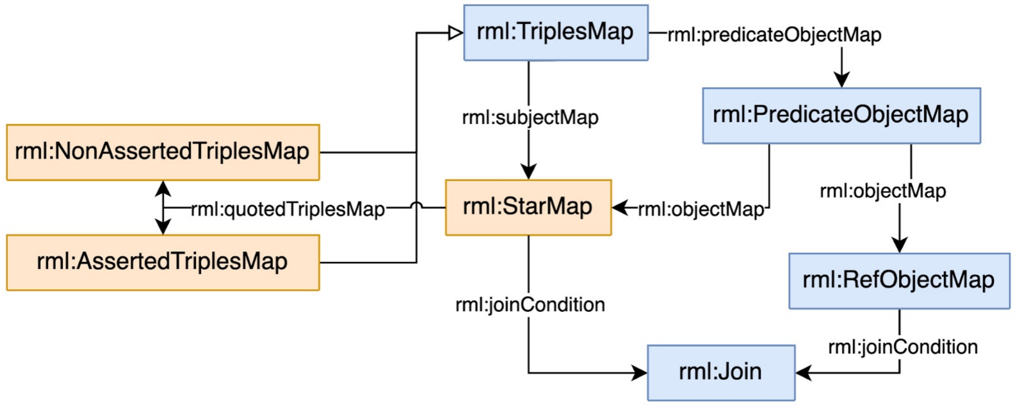


```
<#RankingTriplesMap> a rml:TriplesMap ;
  rml:logicalSource <#JSONSource> ;
 rml:subjectMap [
           rml:constant :Ranking23 ];
  rml:predicateObjectMap [
    rml:predicate ex:contains;
    rml:objectMap [
      rml:gather ( [
        rml:template "{$.*.NAME}";
        rml:termType rml:IRI ] );
      rml:gatherAs rdf:List; ] ] .
```

```
[ { "NAME": "Duplantis",
    "RANK": "1",
    "MARK": "6.22",
    "DATE": 02-25-2023
  },{
    "NAME": "Guttormsen",
    "RANK": "2",
    "MARK": "6.00",
    "DATE": 03-10-2023} ]
```

```
:Ranking23 ex:contains (:Duplantis
:Guttormsen :Vloon).
```

RML-star: RDF-star



- **Recursiveness** in mapping rules to generate quoted triples
- Applicable in **subject** and **object** position
- Asserted and non-asserted quoted triples

w3id.org/rml/fnml

w3id.org/rml/fnml/spec

w3id.org/rml/fnml/shapes

RML-star: Example

```
<#QuotedRankTM> a rml:AssertedTriplesMap ;
  rml:logicalSource <#JSONSource> ;
  rml:subjectMap <#RankSubjectMap> ;
  rml:predicateObjectMap [
          rml:predicate ex:mark ;
    rml:objectMap [
            rml:reference "$.MARK” ;
] ] .

<#DateTM> a rml:TriplesMap ;
  rml:logicalSource <#JSONSource> ;
  rml:subjectMap [
          rml:quotedTriplesMap
<#QuotedRankTM> ];
  rml:predicateObjectMap [
    rml:predicate ex:date ;
    rml:objectMap [
            rml:reference "$.DATE";
] ] .
```

{JSON}

```
[ { "NAME": "Duplantis",
    "RANK": "1",
    "MARK": "6.22",
    "DATE": 02-25-2023
  },{
    "NAME": "Guttormsen",
    "RANK": "2",
    "MARK": "6.00",
    "DATE": 03-10-2023} ]
```

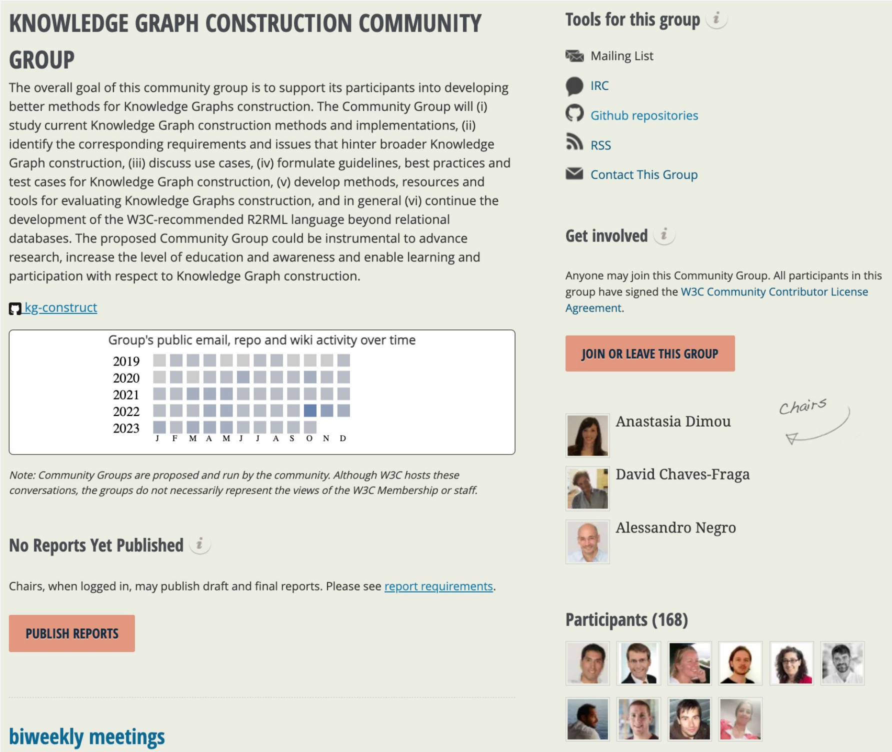
```
:Duplantis ex:mark “6.22” .
<< :Duplantis ex:mark “6.22” >>
         ex:date “02-25-2023” .
:Guttormsen ex:mark “6.00” .
<< :Guttormsen ex:mark “6.00” >>
         ex:date “03-10-2023” .
```

Summary of updates

R2RML	
Schema Transformations	Static and dynamic subject, predicate and object
	Static datatypes and language tags
	Join conditions
Data Source Description	Relational Databases
Function Support	SQL functions
RDF Terms	IRI
	Blank Node
	Literal

RML	
Schema Transformations	Static and dynamic subject, predicate and object
	Static and dynamic predicate and object
	More flexible join conditions
Data Source Description	(Non-)Relational databases, tabular and hierarchical data
	Format, encoding and compression description
	Target data description
Function Support	SQL functions for RDBs
	Function Ontology (FnO)
RDF Terms	IRIs
	Blank Nodes
	Literals
	Collections and containers
	RDF-star triples

KGC W3C Community Group



KNOWLEDGE GRAPH CONSTRUCTION COMMUNITY GROUP

The overall goal of this community group is to support its participants into developing better methods for Knowledge Graphs construction. The Community Group will (i) study current Knowledge Graph construction methods and implementations, (ii) identify the corresponding requirements and issues that hinter broader Knowledge Graph construction, (iii) discuss use cases, (iv) formulate guidelines, best practices and test cases for Knowledge Graph construction, (v) develop methods, resources and tools for evaluating Knowledge Graphs construction, and in general (vi) continue the development of the W3C-recommended R2RML language beyond relational databases. The proposed Community Group could be instrumental to advance research, increase the level of education and awareness and enable learning and participation with respect to Knowledge Graph construction.

kg-construct

Note: Community Groups are proposed and run by the community. Although W3C hosts these conversations, the groups do not necessarily represent the views of the W3C Membership or staff.

No Reports Yet Published

Chairs, when logged in, may publish draft and final reports. Please see report requirements.

PUBLISH REPORTS

biweekly meetings

Tools for this group

- Mailing List
- IRC
- Github repositories
- RSS
- Contact This Group

Get involved

Anyone may join this Community Group. All participants in this group have signed the W3C Community Contributor License Agreement.

JOIN OR LEAVE THIS GROUP

Chairs

Anastasia Dimou

David Chaves-Fraga

Alessandro Negro

Participants (168)

- Maintaining and developing RML new spec
- Towards Working Group in 2024
- 168 participants
  - ~25-30 active
- Bi-weekly general meetings
- Dedicated task-force meetings

http://w3id.org/kg-construct

http://github.com/kg-construct

...This is not the end, but the beginning

Continuously **looking for**:

- **New perspectives**:
  - Feedback on **open issues**
  - **Uncovered use cases**
- Venues:
  - 5th KGC **Workshop** proposal → **ESWC2024**
- Currently **evolving**:
  - **Ongoing** discussions on **joins** and **fields**
- **Tools implementing the new specs**
  - **Preliminar adoption** in e.g., Yatter, Morph-KGC, SDM-RDFizer, Mapeathor

Declarative Construction and Validation of Knowledge Graphs

Ana Iglesias-Molina and Xuemin Duan

Coffee Break!

We will be back at 16:00

Agenda

- 13:00 – 13:10 Introduction
- 13:10 – 15:00 Declarative Knowledge Graph Construction
- 15:00 – 15:30 Break
- 15:30 – 17:20 **Declarative Knowledge Graph Validation**
- 17:20 – 17:30 Conclusions

Knowledge Graph Validation

15:30 - 17:20

Outline:

- SHACL background
- Write SHACL by hand
- RML2SHACL

SHACL Background

SHACL and ShEx

- Shapes Constraint Language (SHACL)
    - developed by the W3C RDF Data Shapes Working Group with the goal to “produce a language for defining structural constraints on RDF graphs.”
    - the first public draft was published in 2015
    - proposed as a W3C Recommendation in 2017
- Shapes Expression Language (ShEx)
    - developed in late 2013 with the goal to provide a human-readable syntax for OSLC Resource Shapes.

SHACL

- validating RDF graphs against a set of constraints in shapes expressed in terms of RDF

- SHACL processor validates whether the nodes in RDF satisfies the constraints and return validation report

SHACL
validate
R D F

SHACL Concepts

Data Graph; Shapes Graph; Node Shape; Property Shape; target declarations; focus node; property path, property value; constraints

Shapes Graph and Data Graph

- an RDF graph containing shapes defining constriants
- RDF graphs that are validated

```
:StudentShape a sh:NodeShape ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string ;
  sh:minCount 1.
```

Shapes Graph and Data Graph

- an RDF graph containing shapes defining constriants
- RDF graphs that are validated

```
:r001 a :Student ;    #passes
  :name “Alice” .

:r002 a :Student ;    #passes
  :name “Bob” ;
  :id “r002” .

:r003 a :Student ;    #fails
  :id “r003” .
```

Shapes Graph and Data Graph

- an RDF graph containing shapes defining constriants
- RDF graphs that are validated

```
:StudentShape a sh:NodeShape ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string ;
  sh:minCount 1.
```

```
:r001 a :Student ;    #passes
  :name “Alice” .

:r002 a :Student ;    #passes
  :name “Bob” ;
  :id “r002” .

:r003 a :Student ;    #fails
  :id “r003” .
```

Node Shape, Target Declaration, and Focus Node

— — —

- specify constraints that need to be met with respect to focus nodes declared by target

```
:StudentShape a sh:NodeShape ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string ;
  sh:minCount 1.
```

```
:r001 a :Student ;    #passes
  :name “Alice” .

:r002 a :Student ;    #passes
  :name “Bob” ;
  :id “r002” .

:r003 a :Student ;    #fails
  :id “r003” .
```

Target Declaration and Focus Node

---

- sh:targetClass

```
:StudentShape a sh:NodeShape ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI.
```

```
:r001 a :Student ;#passes
  :name “Alice” .
```

- sh:targetNode

```
:StudentShape a sh:NodeShape ;
  sh:targetNode :r001 ;
  sh:nodeKind sh:IRI.
```

```
:r001 a :Student ;#passes
  :name “Alice” .
```

- sh:targetSubjectsOf

```
:StudentShape a sh:NodeShape ;
  sh:targetSubjectsOf :name ;
  sh:nodeKind sh:IRI.
```

```
:r001 a :Student ;#passes
  :name “Alice” .
```

- sh:targetObjectsOf

```
:StudentShape a sh:NodeShape ;
  sh:targetObjectsOf :name ;
  sh:nodeKind sh:IRI.
```

```
:r001 a :Student ; #fails
  :name “Alice” .
```

Property Shape, Property Path, and Property Value

- primarily apply to the property value

- reached by focus node vis property path

```
:StudentShape a sh:NodeShape ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string ;
  sh:minCount 1.
```

```
:r001 a :Student ;   #passes
  :name "Alice" .

:r002 a :Student ;   #passes
  :name "Bob" ;
  :id "r001" .

:u001 a :Teacher ;   #passes
  :name "Carol" .
```

No Target No Validation?

```
:StudentShape a sh:NodeShape ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI .
```

```
:StudentShape a sh:NodeShape ;

  sh:nodeKind sh:IRI .
```

```
:StudentShape a sh:NodeShape ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string ;
  sh:minCount 1.
```

```
:StudentShape a sh:NodeShape ;

  sh:nodeKind sh:IRI ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string ;
  sh:minCount 1.
```

SHACL Constraints

- can be divided into
  - **SHACL Core**
  - **SHACL-SPARQL**

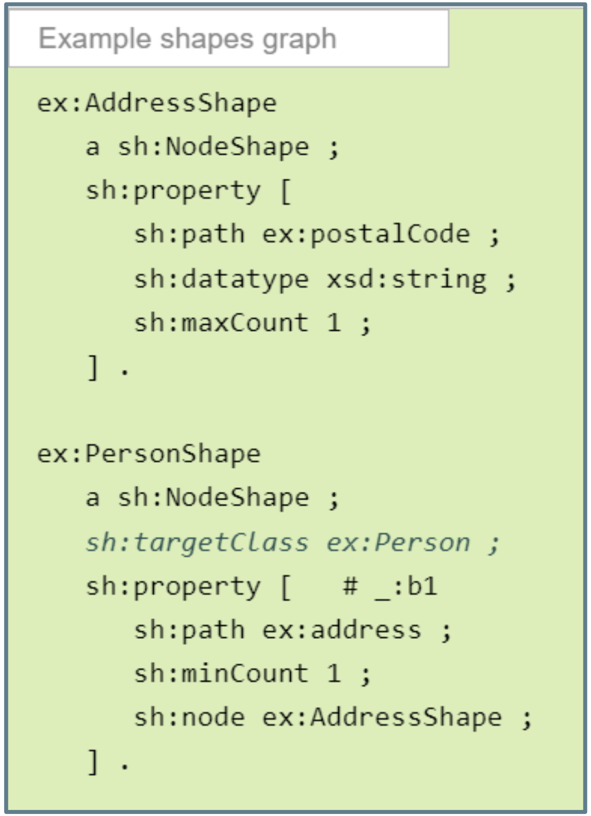
  Some works extend SHACL with (a) advanced features such as rules and complex expressions (called SHACL-Javascript).

Built-in SHACL Core Constraints

Components	**Parameters**
Value types	sh:class, sh:datatype, sh:nodeKind
Cardinality	sh:minCount, sh:maxCount
Value range	sh:minExclusive, sh:minInclusive, sh:minExclusive, sh:minInclusive
String-based	sh:minLength, sh:maxLength, sh:pattern, sh:languageIn,sh:uniqueLang
Property Pair	sh:equals, sh:disjoint, sh:lessThan, sh:lessThanOrEquals
Logical	sh:not, sh:and, sh:or, sh:xone
Shape-based	sh:node, sh:property, sh:qualifiedValueShape, sh:qualifiedMinCount, sh:qualifiedMaxCount
Other	sh:closed, sh:ignoredProperties, sh:hasValue, sh:in
Non-validating	sh:name, sh:description, sh:order, sh:group

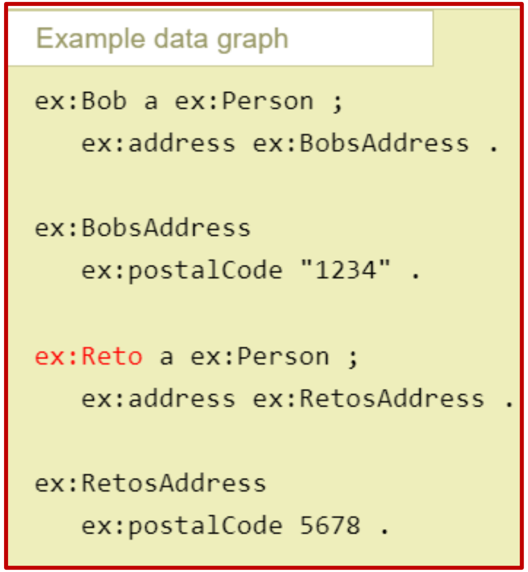
https://www.w3.org/TR/shacl/

sh:node & sh:property



Example shapes graph

```
ex:AddressShape
    a sh:NodeShape ;
    sh:property [
        sh:path ex:postalCode ;
        sh:datatype xsd:string ;
        sh:maxCount 1 ;
    ] .

ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [    # _:b1
        sh:path ex:address ;
        sh:minCount 1 ;
        sh:node ex:AddressShape ;
    ] .
```

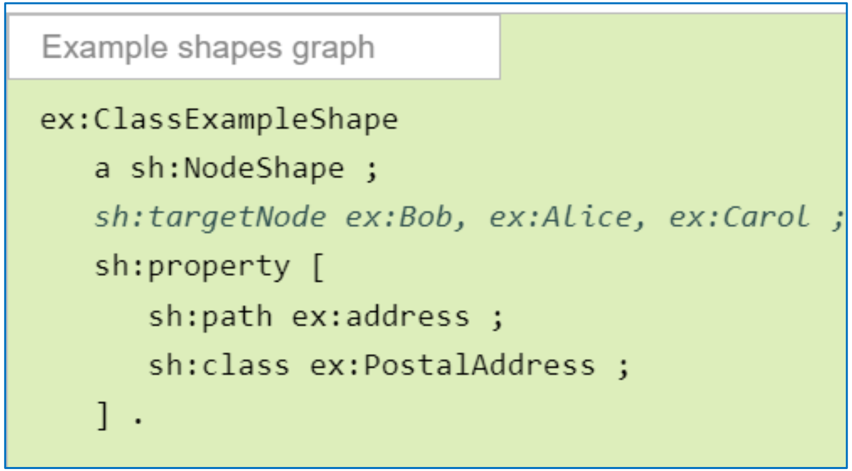


Example data graph

```
ex:Bob a ex:Person ;
    ex:address ex:BobsAddress .

ex:BobsAddress
    ex:postalCode "1234" .

ex:Reto a ex:Person ;
    ex:address ex:RetosAddress .

ex:RetosAddress
    ex:postalCode 5678 .
```

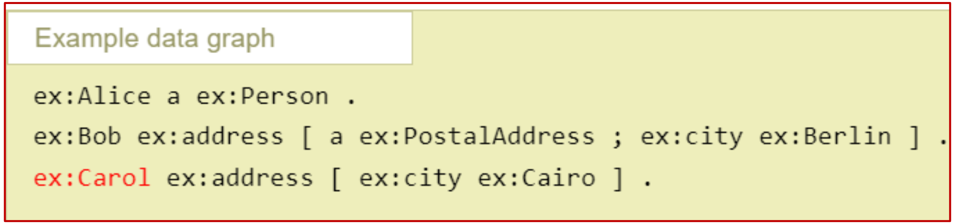
https://www.w3.org/TR/shacl/

sh:class



Example shapes graph

```
ex:ClassExampleShape
    a sh:NodeShape ;
    sh:targetNode ex:Bob, ex:Alice, ex:Carol ;
    sh:property [
        sh:path ex:address ;
        sh:class ex:PostalAddress ;
    ] .
```

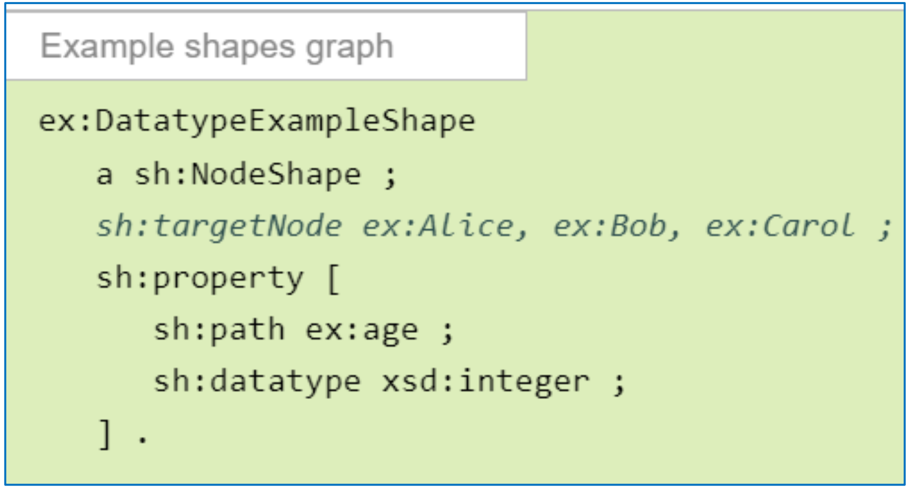


Example data graph

```
ex:Alice a ex:Person .
ex:Bob ex:address [ a ex:PostalAddress ; ex:city ex:Berlin ] .
ex:Carol ex:address [ ex:city ex:Cairo ] .
```

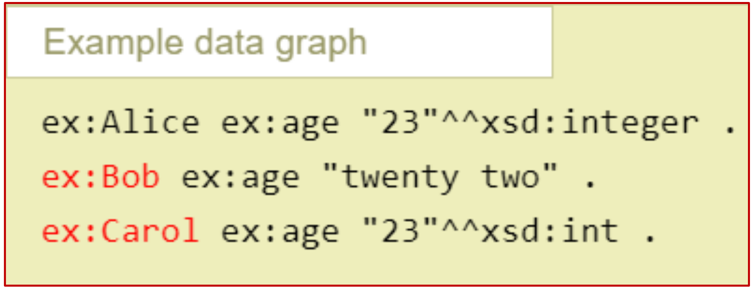
https://www.w3.org/TR/shacl/

sh:datatype



Example shapes graph

```
ex:DatatypeExampleShape
    a sh:NodeShape ;
    sh:targetNode ex:Alice, ex:Bob, ex:Carol ;
    sh:property [
        sh:path ex:age ;
        sh:datatype xsd:integer ;
    ] .
```

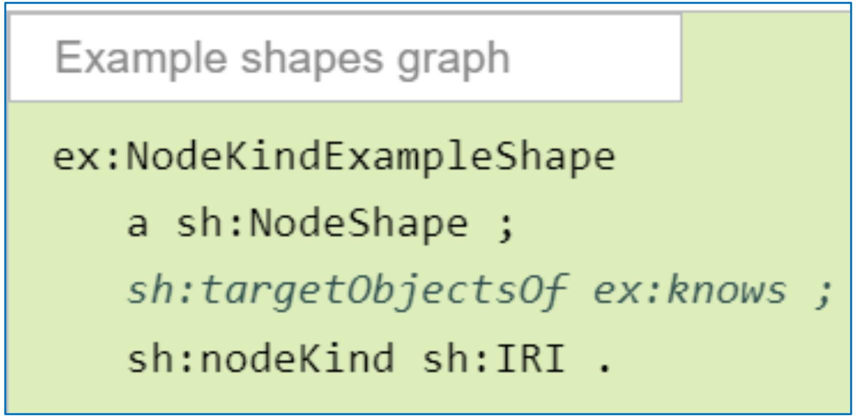


Example data graph

```
ex:Alice ex:age "23"^^xsd:integer .
ex:Bob ex:age "twenty two" .
ex:Carol ex:age "23"^^xsd:int .
```

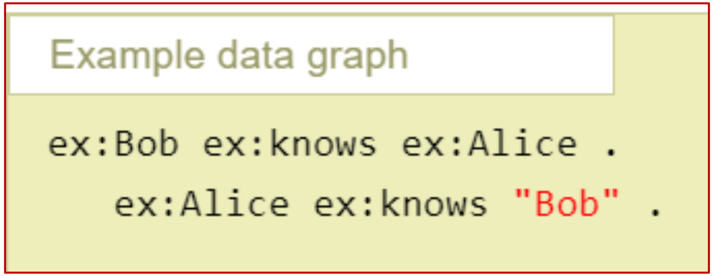
https://www.w3.org/TR/shacl/

sh:nodeKind



Example shapes graph

```
ex:NodeKindExampleShape
    a sh:NodeShape ;
    sh:targetObjectsOf ex:knows ;
    sh:nodeKind sh:IRI .
```

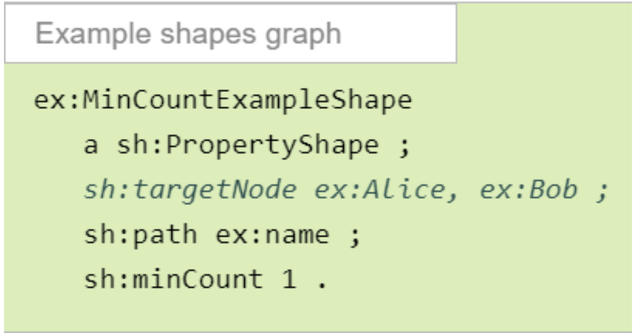


Example data graph

```
ex:Bob ex:knows ex:Alice .
   ex:Alice ex:knows "Bob" .
```

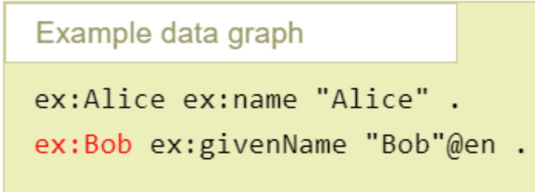
https://www.w3.org/TR/shacl/

sh:minCount & sh:maxCount



Example shapes graph

```
ex:MinCountExampleShape
    a sh:PropertyShape ;
    sh:targetNode ex:Alice, ex:Bob ;
    sh:path ex:name ;
    sh:minCount 1 .
```

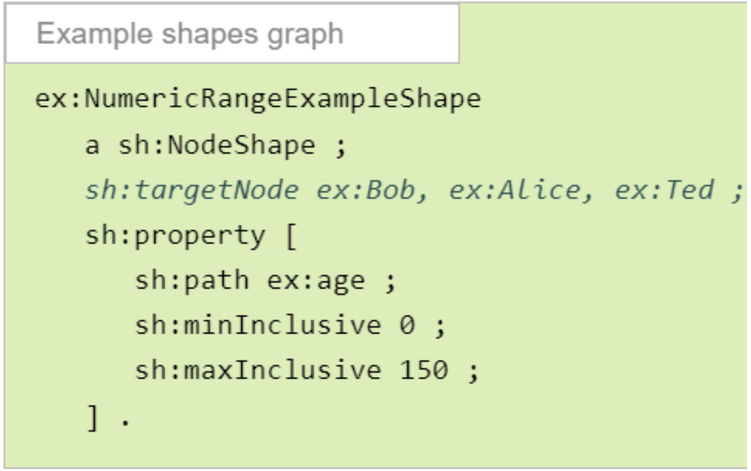


Example data graph

```
ex:Alice ex:name "Alice" .
ex:Bob ex:givenName "Bob"@en .
```

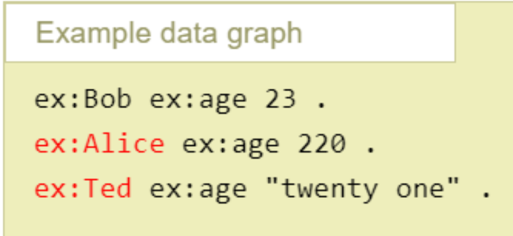
https://www.w3.org/TR/shacl/

sh:minExclusive & sh:minInclusive & sh:minExclusive & sh:minInclusive



Example shapes graph

```
ex:NumericRangeExampleShape
   a sh:NodeShape ;
   sh:targetNode ex:Bob, ex:Alice, ex:Ted ;
   sh:property [
      sh:path ex:age ;
      sh:minInclusive 0 ;
      sh:maxInclusive 150 ;
   ] .
```

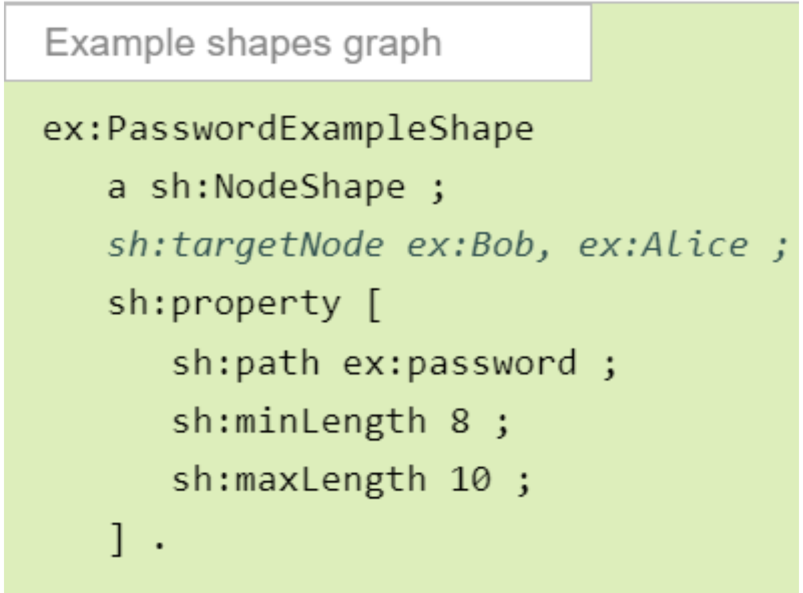


Example data graph

```
ex:Bob ex:age 23 .
ex:Alice ex:age 220 .
ex:Ted ex:age "twenty one" .
```

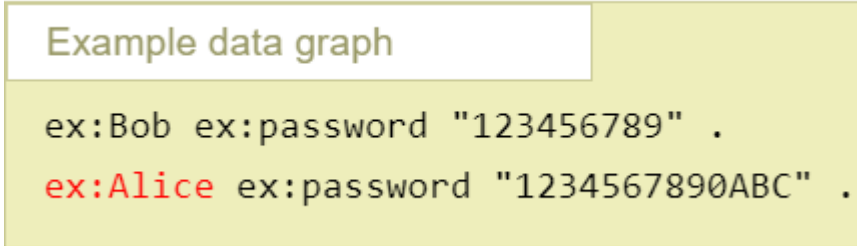
https://www.w3.org/TR/shacl/

sh:minLength & sh:maxLength



Example shapes graph

```
ex:PasswordExampleShape
   a sh:NodeShape ;
   sh:targetNode ex:Bob, ex:Alice ;
   sh:property [
      sh:path ex:password ;
      sh:minLength 8 ;
      sh:maxLength 10 ;
   ] .
```

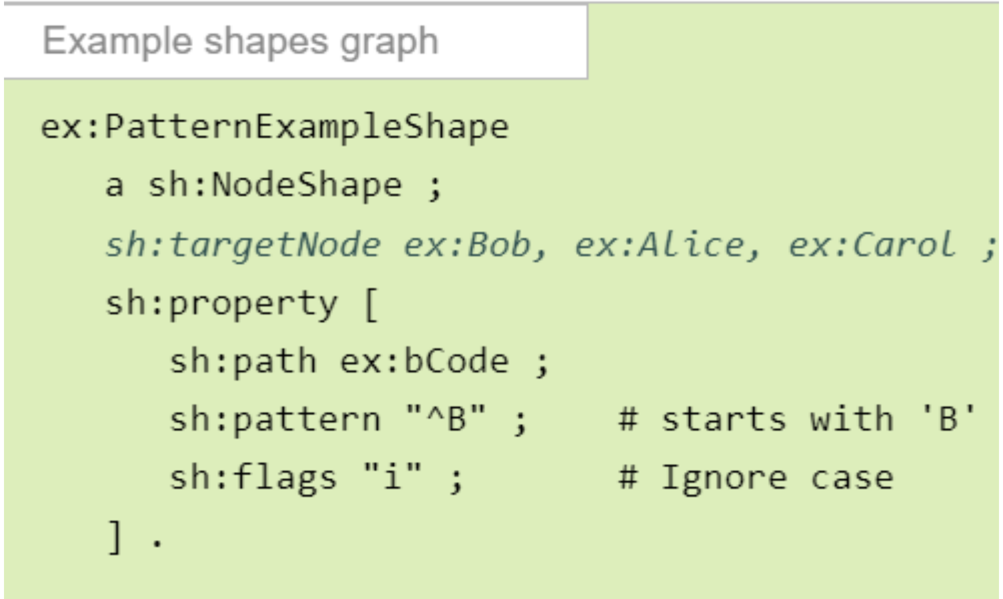


Example data graph

```
ex:Bob ex:password "123456789" .
ex:Alice ex:password "1234567890ABC" .
```

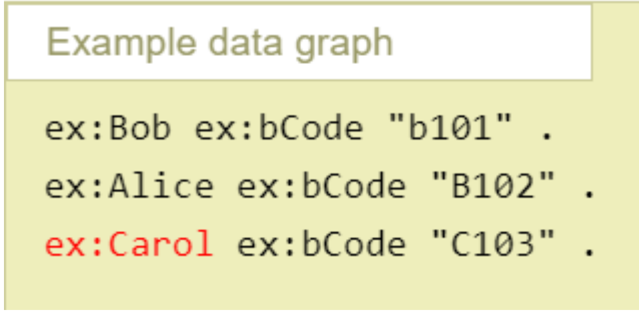
https://www.w3.org/TR/shacl/

sh:pattern



Example shapes graph

```
ex:PatternExampleShape
    a sh:NodeShape ;
    sh:targetNode ex:Bob, ex:Alice, ex:Carol ;
    sh:property [
        sh:path ex:bCode ;
        sh:pattern "^B" ;     # starts with 'B'
        sh:flags "i" ;        # Ignore case
    ] .
```

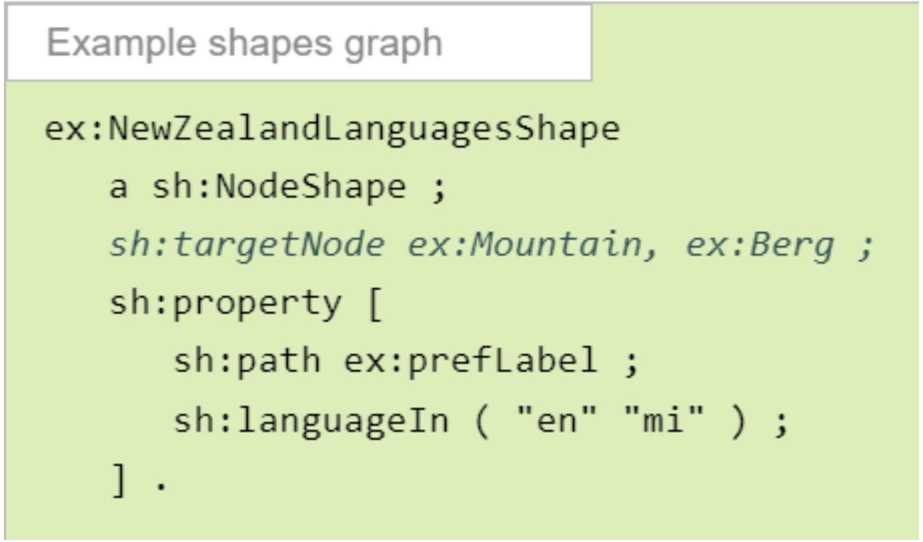


Example data graph

```
ex:Bob ex:bCode "b101" .
ex:Alice ex:bCode "B102" .
ex:Carol ex:bCode "C103" .
```

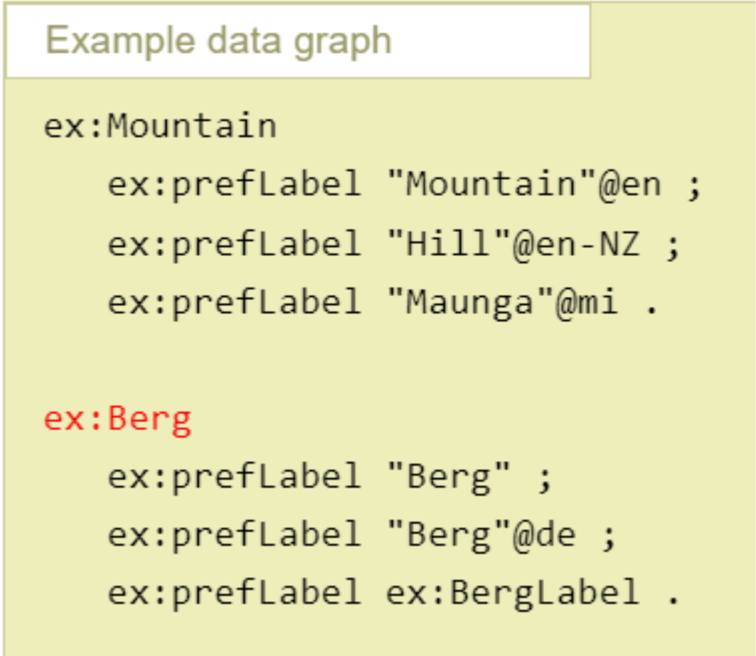
https://www.w3.org/TR/shacl/

sh:languageIn & sh:uniqueLang



Example shapes graph

```
ex:NewZealandLanguagesShape
    a sh:NodeShape ;
    sh:targetNode ex:Mountain, ex:Berg ;
    sh:property [
        sh:path ex:prefLabel ;
        sh:languageIn ( "en" "mi" ) ;
    ] .
```

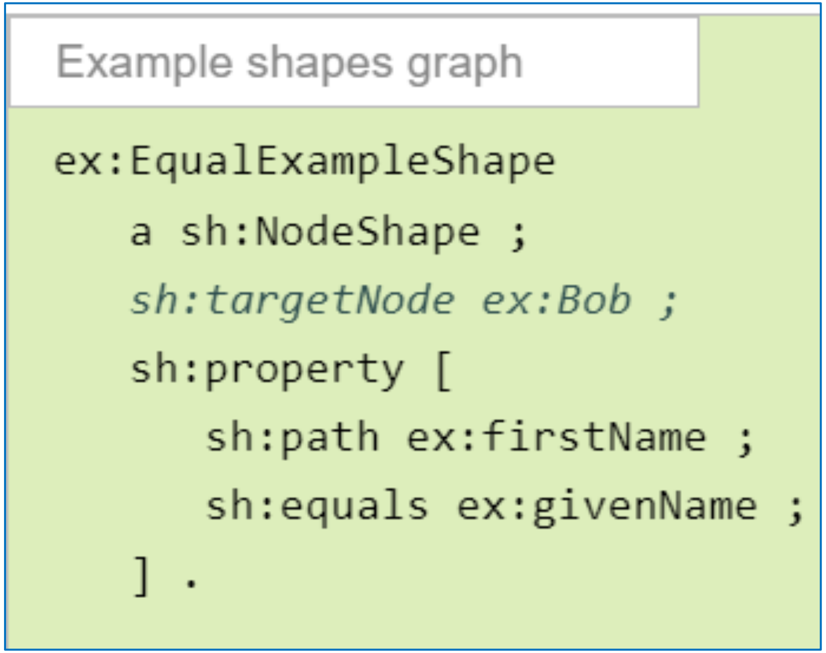


Example data graph

```
ex:Mountain
    ex:prefLabel "Mountain"@en ;
    ex:prefLabel "Hill"@en-NZ ;
    ex:prefLabel "Maunga"@mi .

ex:Berg
    ex:prefLabel "Berg" ;
    ex:prefLabel "Berg"@de ;
    ex:prefLabel ex:BergLabel .
```

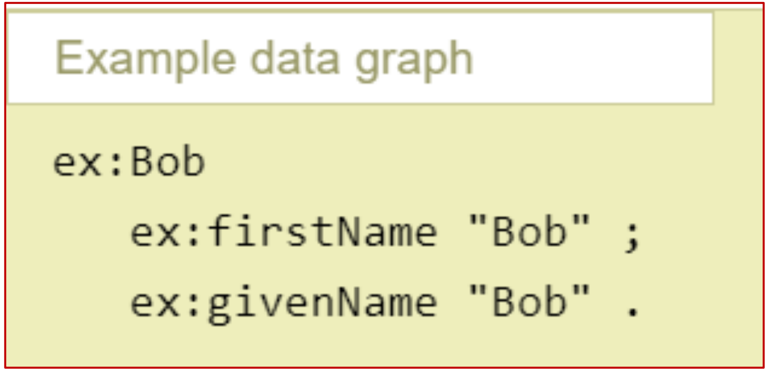
https://www.w3.org/TR/shacl/

sh:equals & disjoint



Example shapes graph

```
ex:EqualExampleShape
    a sh:NodeShape ;
    sh:targetNode ex:Bob ;
    sh:property [
        sh:path ex:firstName ;
        sh:equals ex:givenName ;
    ] .
```

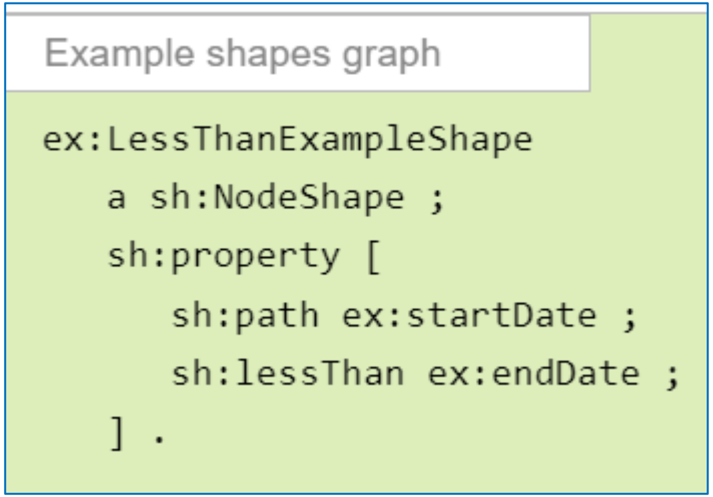


Example data graph

```
ex:Bob
    ex:firstName "Bob" ;
    ex:givenName "Bob" .
```

https://www.w3.org/TR/shacl/

sh:lessThan & sh:lessThanOrEquals



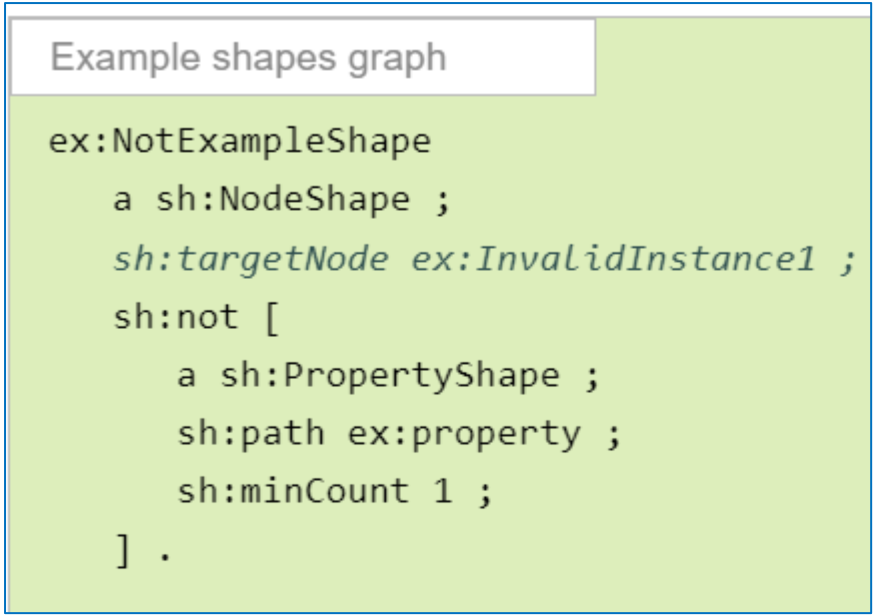
Example shapes graph

```
ex:LessThanExampleShape
	a sh:NodeShape ;
	sh:property [
		sh:path ex:startDate ;
		sh:lessThan ex:endDate ;
	] .
```

```
:r001 a :Example ;                    #passes
    ex:startDate "2017-04-20T20:00:00"^^xsd:dateTime ;
    ex:endDate "2017-04-20T21:30:00"^^xsd:dateTime ;.
```

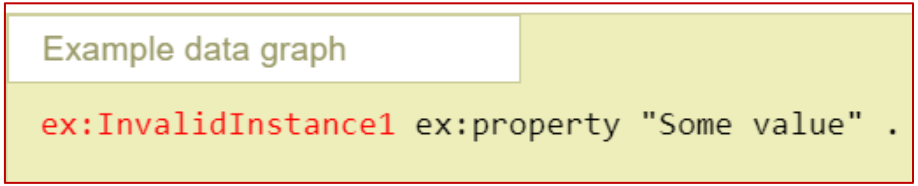
https://www.w3.org/TR/shacl/

sh:not & sh:and & sh:or & sh:xone



Example shapes graph

```
ex:NotExampleShape
    a sh:NodeShape ;
    sh:targetNode ex:InvalidInstance1 ;
    sh:not [
        a sh:PropertyShape ;
        sh:path ex:property ;
        sh:minCount 1 ;
    ] .
```

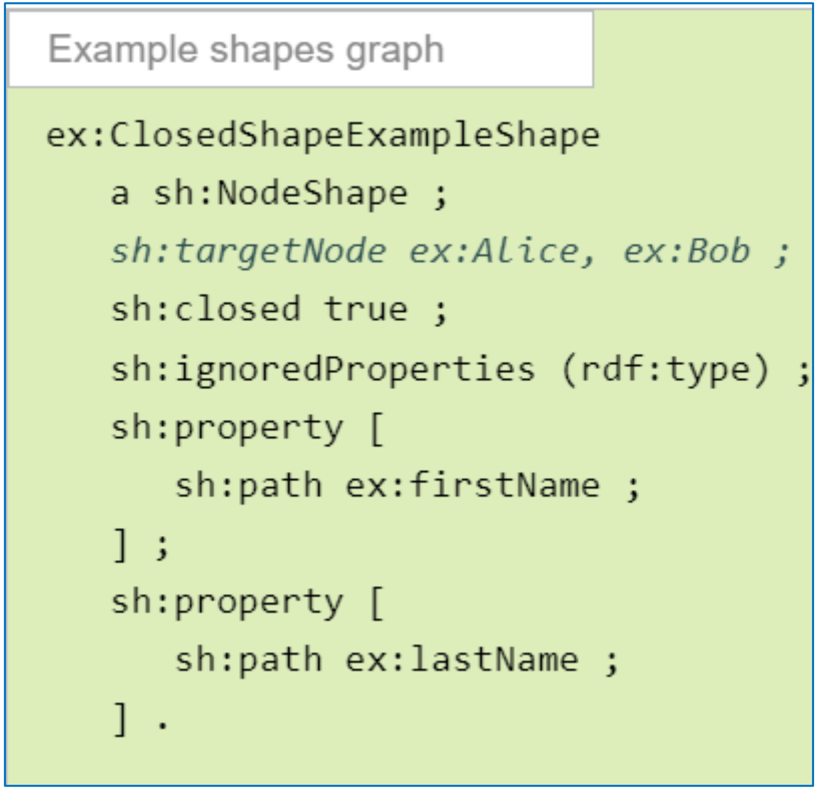


Example data graph

```
ex:InvalidInstance1 ex:property "Some value" .
```

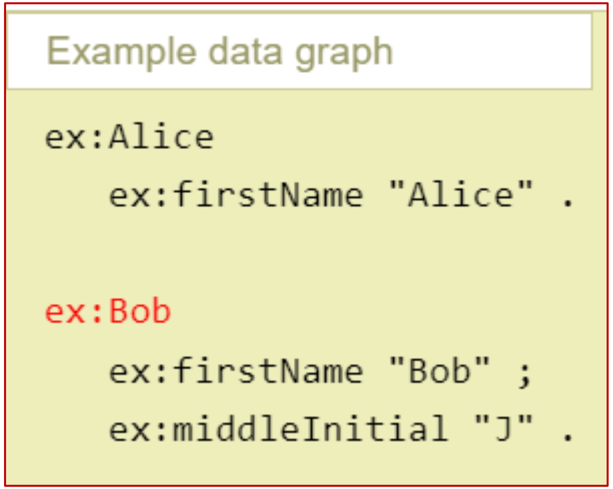
https://www.w3.org/TR/shacl/

sh:close & sh:ignoredProperties



Example shapes graph

```
ex:ClosedShapeExampleShape
    a sh:NodeShape ;
    sh:targetNode ex:Alice, ex:Bob ;
    sh:closed true ;
    sh:ignoredProperties (rdf:type) ;
    sh:property [
        sh:path ex:firstName ;
    ] ;
    sh:property [
        sh:path ex:lastName ;
    ] .
```

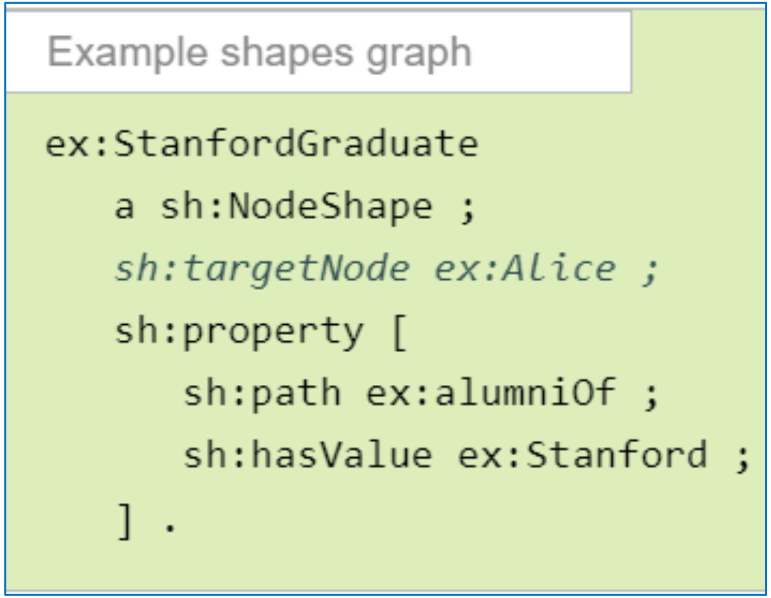


Example data graph

```
ex:Alice
    ex:firstName "Alice" .

ex:Bob
    ex:firstName "Bob" ;
    ex:middleInitial "J" .
```

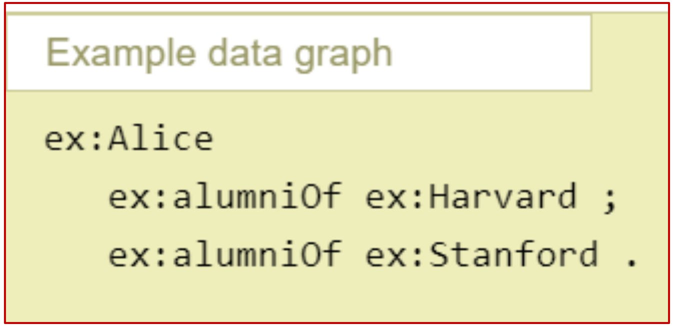
https://www.w3.org/TR/shacl/

sh:hasValue & sh:in



Example shapes graph

```
ex:StanfordGraduate
	a sh:NodeShape ;
	sh:targetNode ex:Alice ;
	sh:property [
		sh:path ex:alumniOf ;
		sh:hasValue ex:Stanford ;
	] .
```



Example data graph

```
ex:Alice
	ex:alumniOf ex:Harvard ;
	ex:alumniOf ex:Stanford .
```

SHACL syntax (SHACL-SHACL)

- Target declarations are optional for node shape and property shape

- Property shape must have exactly one property path (i.e. sh:path)

- Each shape has at most one sh:datatype, sh:nodeKind , …

- Each shape can have multiple sh:class, sh:and, …

Validation Report

- identified by sh:ValidationReport
- has exactly one sh:conforms (true/false)
- optional has sh:result if False

```
:report a sh:ValidationReport ;
  sh:conforms true .
```

```
:report a sh:ValidationReport ;
  sh:conforms false ;
  sh:result
    [a sh:ValidationResult;
     sh:resultSeverity sh:Violation ;
     sh:sourceConstraintComponent
sh:DatatypeConstraintComponent ;
     sh:sourceShape ... ;
     sh:focusNode :r001 ;
     sh:value 2000;
     sh:resultPath :name ;
     sh:resultMessage "Value does not
have datatype xsd:string" ],
```

Hands-on exercise

Write SHACL shapes by hand

SHACL validation.ipynb
https://colab.research.google.com/drive/1n8EAjS7Yq022JF8z067hIA6CCXHIni7T?usp=drive_link

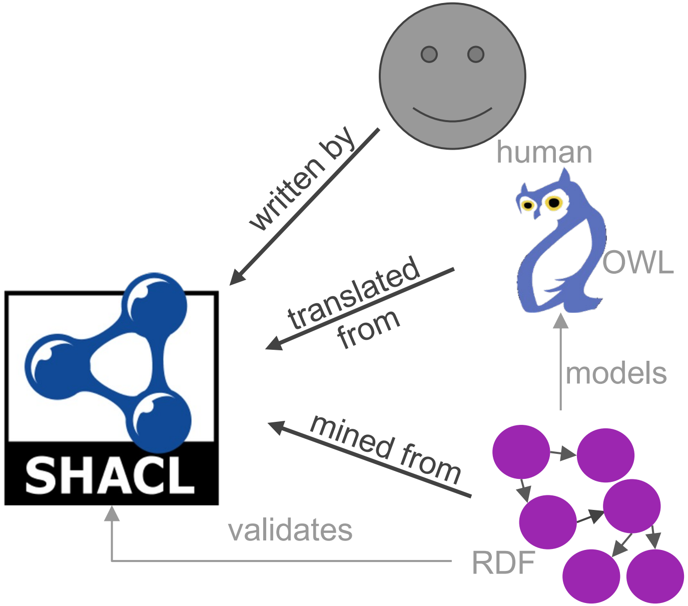
- Play with several simple tasks

- Learn to use SHACL-SHACL to validate whether the created SHACL shapes are well-formed

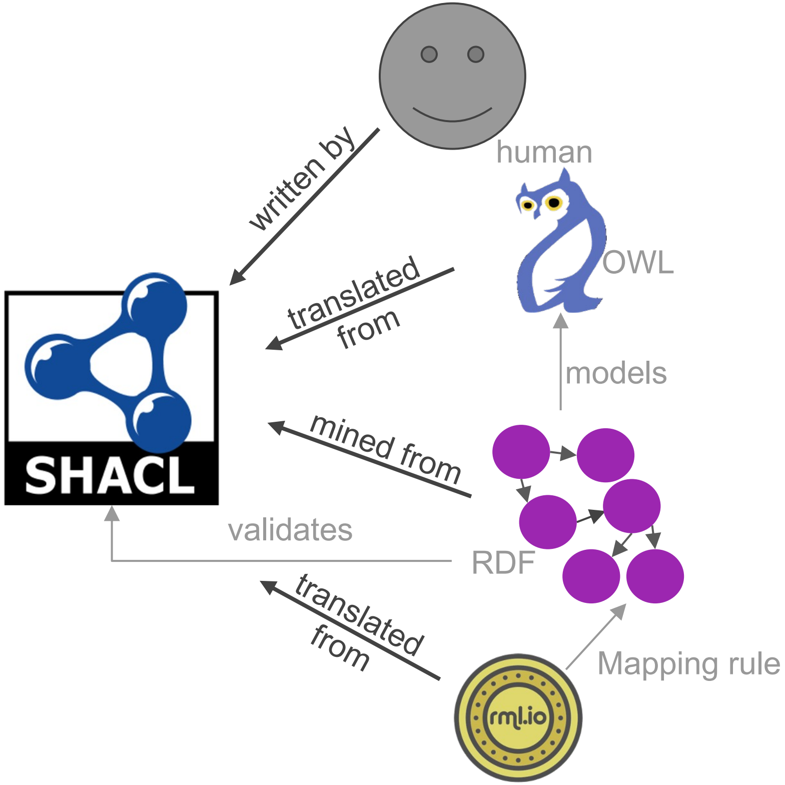
- Validate RDF graphs using created SHACL shapes

RML2SHACL

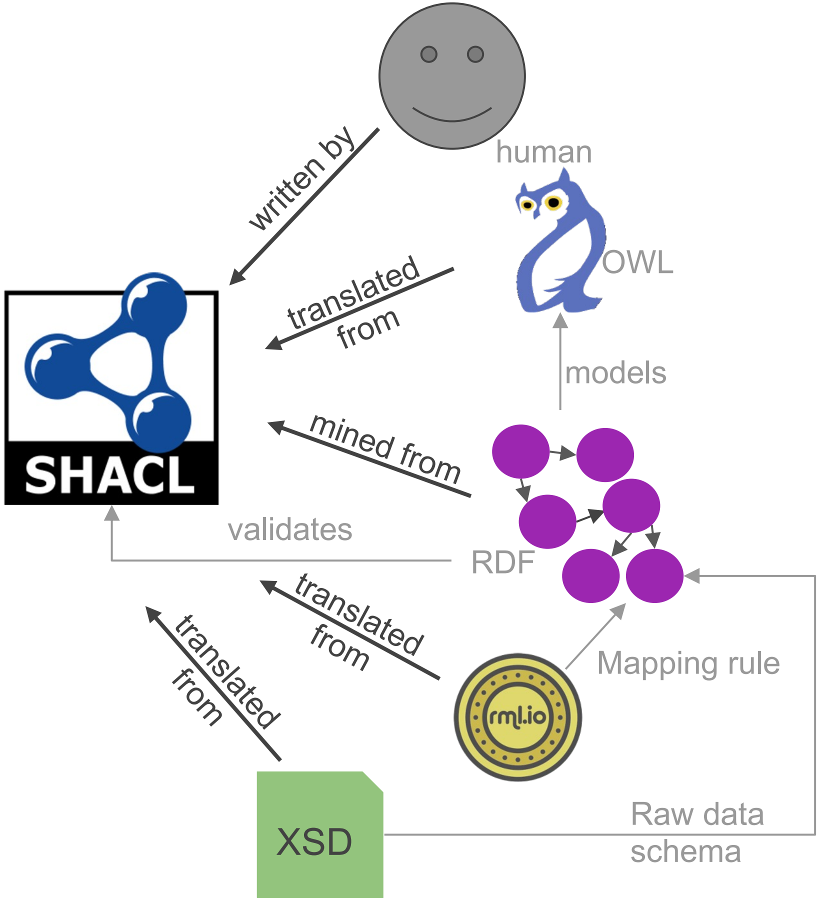
Automatic SHACL shapes extraction



Automatic SHACL shapes extraction

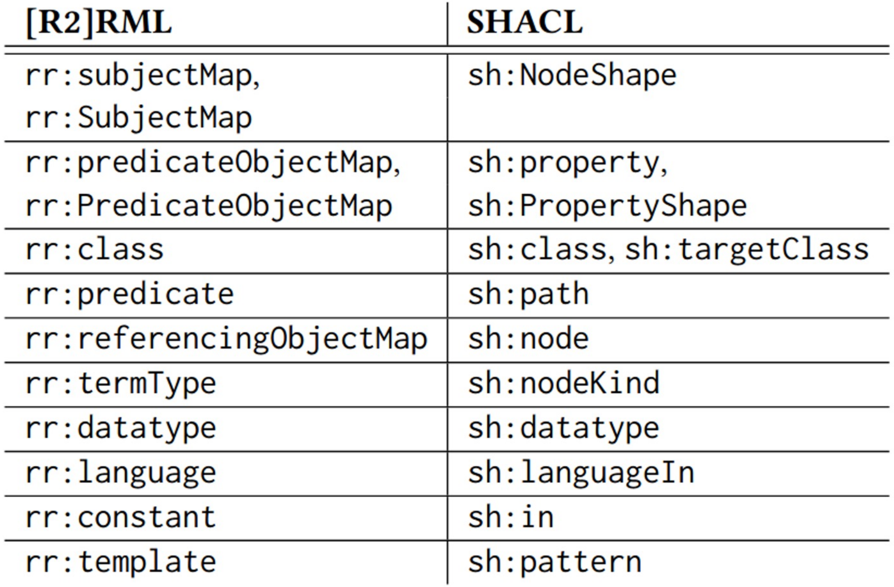


Automatic SHACL shapes extraction



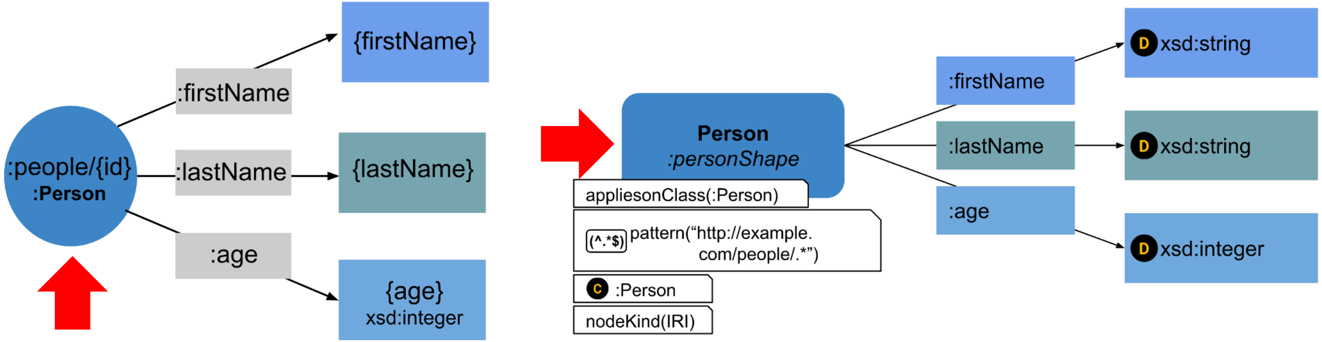
RML2SHACL

- Translate RML mapping rules to SHACL shapes
- Generate shapes that validate the output of RML mapping rules



[R2]RML	SHACL
rr:subjectMap, rr:SubjectMap	sh:NodeShape
rr:predicateObjectMap, rr:PredicateObjectMap	sh:property, sh:PropertyShape
rr:class	sh:class, sh:targetClass
rr:predicate	sh:path
rr:referencingObjectMap	sh:node
rr:termType	sh:nodeKind
rr:datatype	sh:datatype
rr:language	sh:languageIn
rr:constant	sh:in
rr:template	sh:pattern

Correspondence (1/7)



rr:subjectMap, rr:SubjectMap	sh:NodeShape

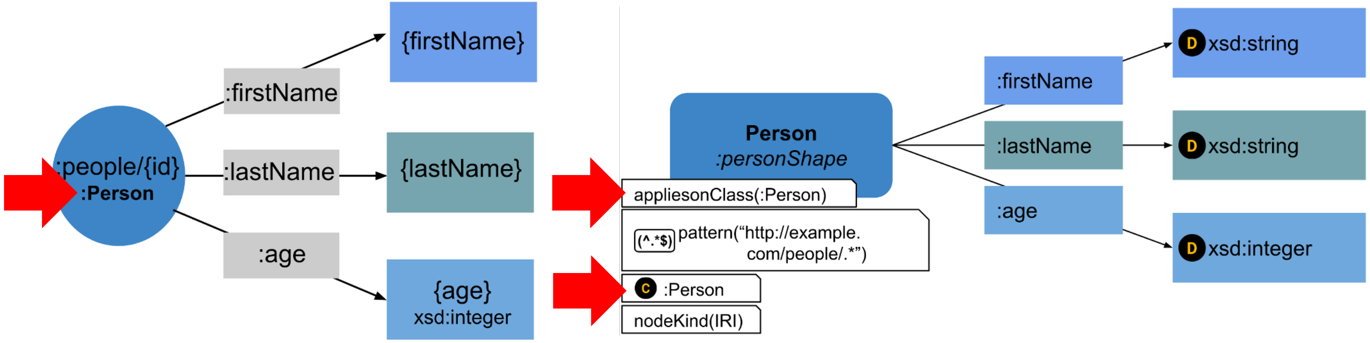
Correspondence (1/7)

```
:StudentMapping a rr:TriplesMap;

rr:subjectMap [
    rr:template "http://example.com/{id}";
    rr:class :Student
    rr:termType rr:IRI ];

  rr:predicateObjectMap [
    rr:predicate :name;
    rr:objectMap [
      rml:reference “name";
      rr:datatype xsd:string ] ].
```

```
:StudentShape a sh:NodeShape .
```

rr:subjectMap, rr:SubjectMap	sh:NodeShape

Correspondence (2/7)

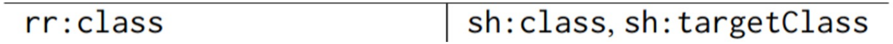


rr:class	sh:class, sh:targetClass

Correspondence (2/7)

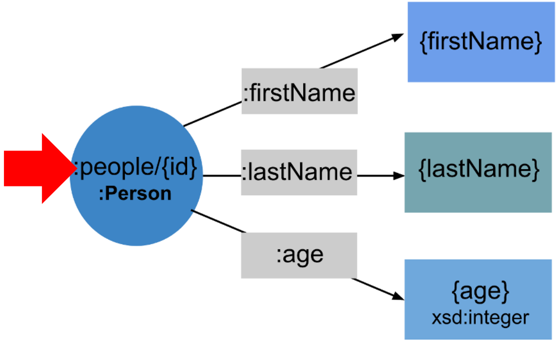
```
:StudentMapping a rr:TriplesMap;

rr:subjectMap [
    rr:template "http://example.com/{id}";
    rr:class :Student ;
    rr:termType rr:IRI ];

  rr:predicateObjectMap [
    rr:predicate :name;
    rr:objectMap [
      rml:reference “name";
      rr:datatype xsd:string ] ].
```

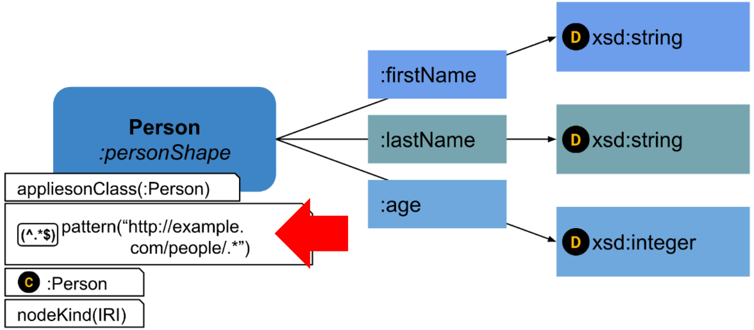
```
:StudentShape a sh:NodeShape ;
  sh:class :Student ;
  sh:targetClass :Student ;
```



rr:class	sh:class, sh:targetClass

Correspondence (3/7)





`rr:template` | `sh:pattern`

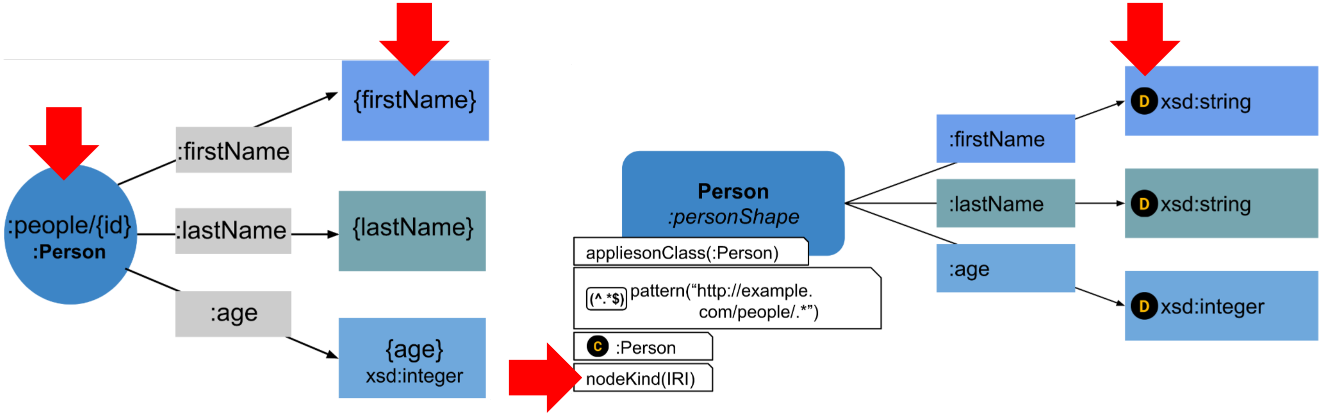
Correspondence (3/7)

```
:StudentMapping a rr:TriplesMap;

rr:subjectMap [
    rr:template "http://example.com/{id}";
    rr:class :Student ;
    rr:termType rr:IRI ];

  rr:predicateObjectMap [
    rr:predicate :name;
    rr:objectMap [
      rml:reference “name";
      rr:datatype xsd:string ] ].
```

```
:StudentShape a sh:NodeShape ;
  sh:class :Student ;
  sh:targetClass :Student ;
  sh:pattern
"http://example.com/.*" .
```

rr:template | sh:pattern

Correspondence (4/7)



rr:termType | sh:nodeKind

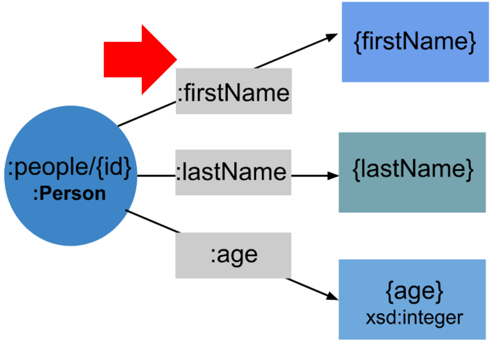
Correspondence (4/7)

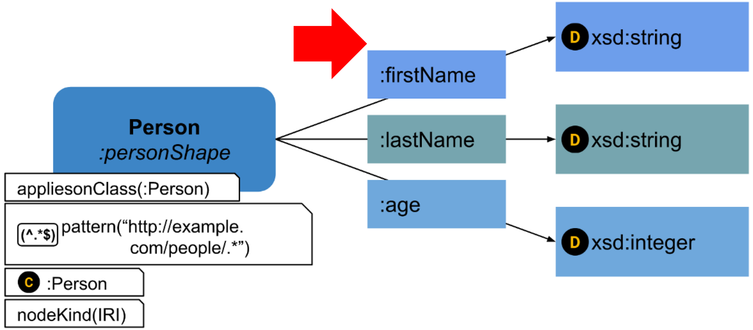
```
:StudentMapping a rr:TriplesMap;

rr:subjectMap [
    rr:template "http://example.com/{id}";
    rr:class :Student ;
    rr:termType rr:IRI];

  rr:predicateObjectMap [
    rr:predicate :name;
    rr:objectMap [
      rml:reference “name";
      rr:datatype xsd:string ] ].
```

```
:StudentShape a sh:NodeShape ;
  sh:class :Student ;
  sh:targetClass :Student ;
  sh:pattern +++ ;
  sh:nodeKind sh:IRI .
```

rr:termType	sh:nodeKind

Correspondence (5/7)





`rr:predicateObjectMap,`
`rr:PredicateObjectMap`
	`sh:property,`
	`sh:PropertyShape`

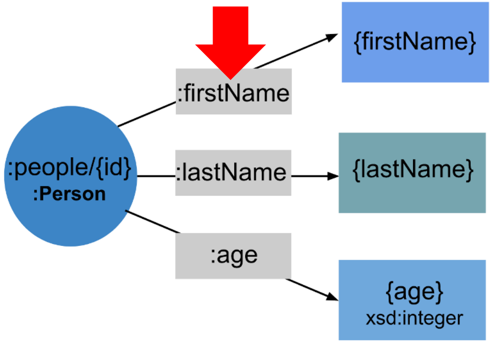
Correspondence (5/7)

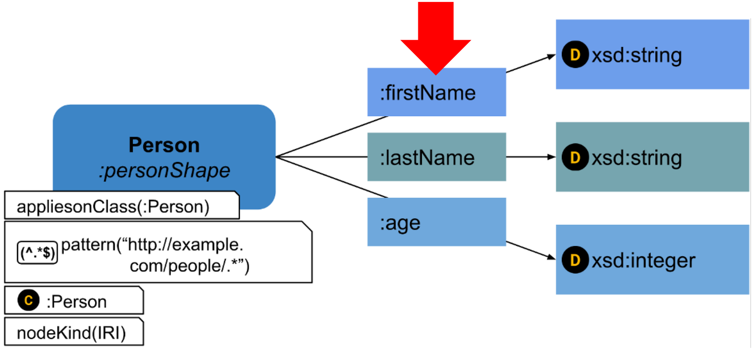
```
:StudentMapping a rr:TriplesMap;

rr:subjectMap [
    rr:template "http://example.com/{id}";
    rr:class :Student
    rr:termType rr:IRI];

  rr:predicateObjectMap [
    rr:predicate :name;
    rr:objectMap [
      rml:reference “name";
      rr:datatype xsd:string ] ].
```

```
:StudentShape a sh:NodeShape ;
  sh:class :Student ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI ;
  sh:pattern
"http://example.com/.*" ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string .
```

rr:predicateObjectMap, rr:PredicateObjectMap	sh:property, sh:PropertyShape

Correspondence (6/7)





`rr:predicate` | `sh:path`

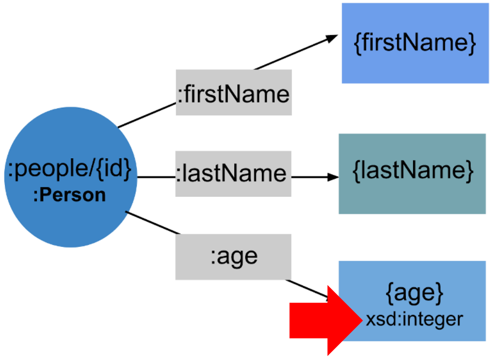
Correspondence (6/7)

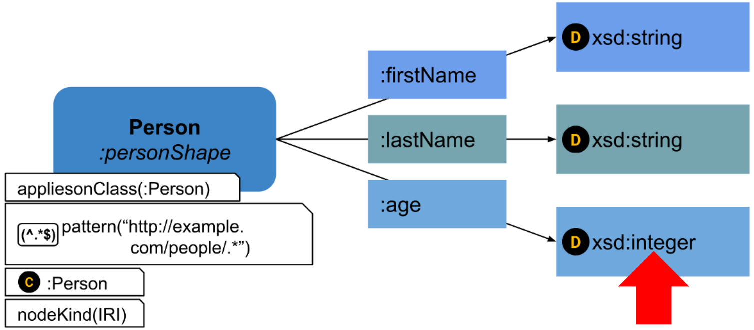
```
:StudentMapping a rr:TriplesMap;

rr:subjectMap [
    rr:template "http://example.com/{id}";
    rr:class :Student ];

  rr:predicateObjectMap [
    rr:predicate :name;
    rr:objectMap [
      rml:reference “name";
      rr:datatype xsd:string ] ].
```

```
:StudentShape a sh:NodeShape ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI ;
  sh:pattern
"http://example.com/.*" ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string .
```

rr:predicate | sh:path

Correspondence (7/7)





rr:datatype | sh:datatype

Correspondence (7/7)

```
:StudentMapping a rr:TriplesMap;

rr:subjectMap [
    rr:template "http://example.com/{id}";
    rr:class :Student ;
    rr:termType rr:IRI];

  rr:predicateObjectMap [
    rr:predicate :name;
    rr:objectMap [
      rml:reference “name";
      rr:datatype xsd:string ] ].
```

```
:StudentShape a sh:NodeShape ;
  sh:class :Student ;
  sh:targetClass :Student ;
  sh:nodeKind sh:IRI ;
  sh:pattern
"http://example.com/.*" ;
  sh:property :NameShape .

:NameShape a sh:PropertyShape ;
  sh:path :name ;
  sh:datatype xsd:string .
```

rr:datatype | sh:datatype

Try RML2SHACL

Try RML2SHACL

SHACL validation.ipynb
https://colab.research.google.com/drive/1n8EAjS7Yq022JF8z067hIA6CCXHIni7T?usp=drive_link

- generate SHACL shapes from RML mapping rules

- validate RDF graphs using RML-driven shapes

Wrapping up...

Conclusions

- Declarative pipeline for KG creation and validation
  - From heterogeneous data sources
  - Relying on standards (or in process to be)
  - Maintenance, reproducibility, understandability
  - Mature ecosystem of compliant systems
- Struggle in adoption (but progressing)
  - Some approaches are too verbose
  - Requires learning curve despite user-friendly developments
  - Manual effort to create mappings and shapes
    - How can LLMs help?

As for K-CAP…

Session 6
Thursday 7th at 14:10

Re-Construction Impact on Metadata Representation Models

Ana Iglesias-Molina, Jhon Toledo, Oscar Corcho and David Chaves-Fraga

Session 6
Thursday 7th at 14:50

XSD2SHACL: Capturing RDF Constraints from XML Schema

Xuemin Duan, David Chaves-Fraga and Anastasia Dimou

Declarative Construction and Validation of Knowledge Graphs

Ana Iglesias-Molina and Xuemin Duan

Thanks for attending!