KG4DI

W3C®

Tutorial on Knowledge Graph Construction

David Chaves-Fraga (KULeuven)
Dylan Van Assche (Ghent U.)
Christophe Debruyne (ULiège)

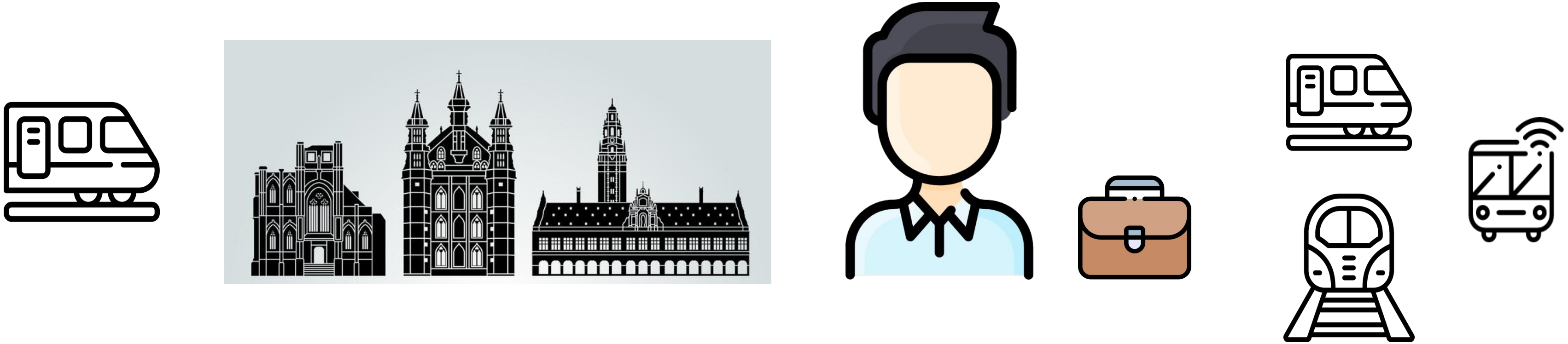
✉ david.chaves@kuleuven.be

@dchavesf

Agenda

- Introduction to KG construction
- Easy Mapping Languages: YARRRML
- Constructing KGC in Python Notebooks: Morph-KGC
- Reasoning over KGs

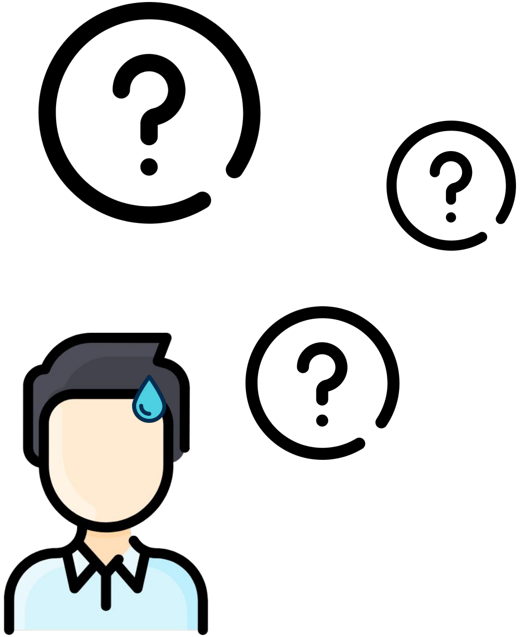
We arrive in Leuven, and as data engineers, we are in charge of handling Belgian's **transport data** to enhance sustainable mobility



* Adapted from KGC with Declarative Mapping Rules Tutorial @ ISWC 2020 (Iglesias-Molina et al.)

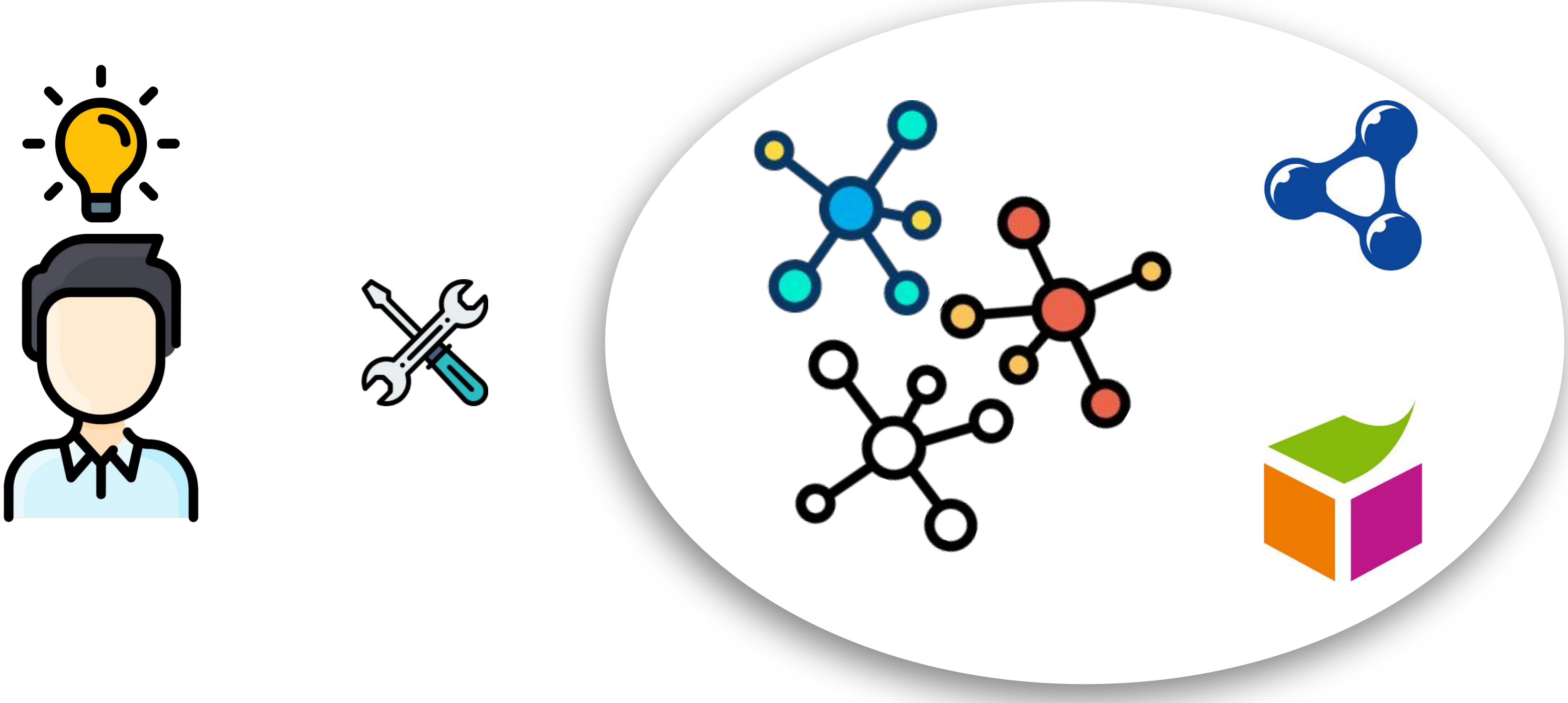
However, real transport data is **tricky**:

- **Heterogeneity**
- **Complexity**
- Data **not clean** and **normalized**
- Values, such as dates, have to be **correctly interpreted**
- Has to be **understandable** by anyone



* Adapted from KGC with Declarative Mapping Rules Tutorial @ ISWC 2020 (Iglesias-Molina et al.)

So he decided to build a **Knowledge Graph**, the solution to represent clearly the heterogeneous and make it **clear**, **accessible** and **queriable**



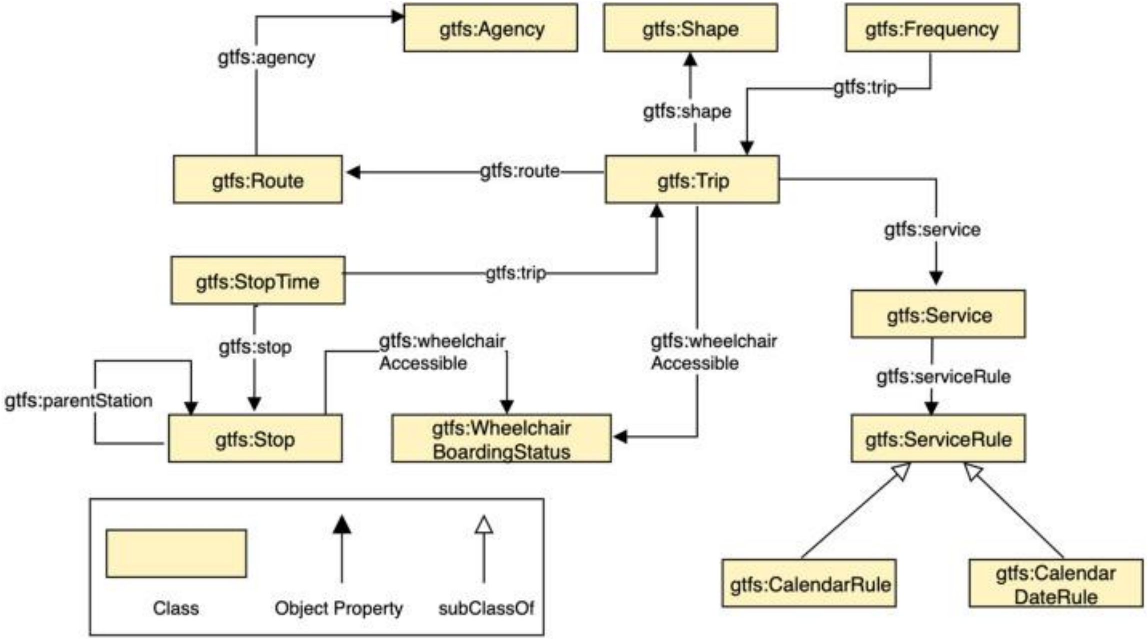
* Adapted from KGC with Declarative Mapping Rules Tutorial @ ISWC 2020 (Iglesias-Molina et al.)

The **requirements** needed for this data integration pipeline are:

- **Ontology** that models the transport domain
- **Standard declarative mapping rules**:
  - Flexibility
  - Adaptability
  - Maintainability
  - Readability
  - Reproducibility
- Ensure **materialized** KG
- **Avoid** at maximum ad-hoc and **manual** steps
- **Efficient** generation

* Adapted from KGC with Declarative Mapping Rules Tutorial @ ISWC 2020 (Iglesias-Molina et al.)

He has been given the data, so it's time to choose a suitable ontology: the **Linked General Transit Feed Specification (LinkedGTFS)**



- To be aligned with the GTFS spec[1] (de-facto standard for open transport data)
- 13 input sources (in different potential formats)
- No information about data size, join selectivity, etc.

[1] https://developers.google.com/transit/gtfs

GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain Chaves-Fraga et al. JoWS 2020

KG4DI

Introduction to Declarative Knowledge Graph Construction

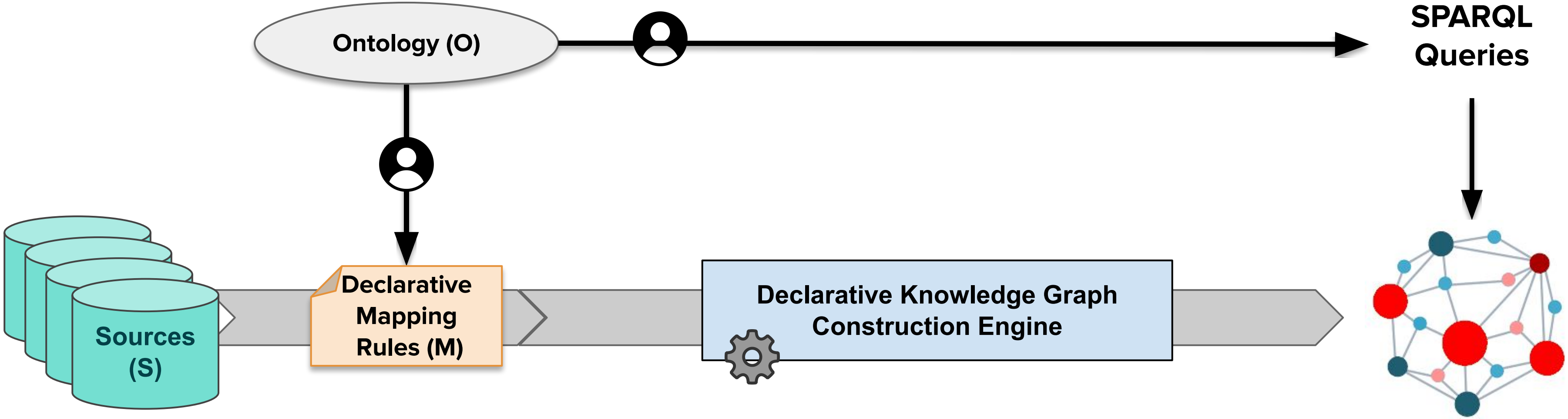
Dylan Van Assche

✉ dylan.vanassche@ugent.be

@dylanvanassche@fosstodon.org

https://dylanvanassche.be

KG Construction with Mapping Rules



Ontology (O)

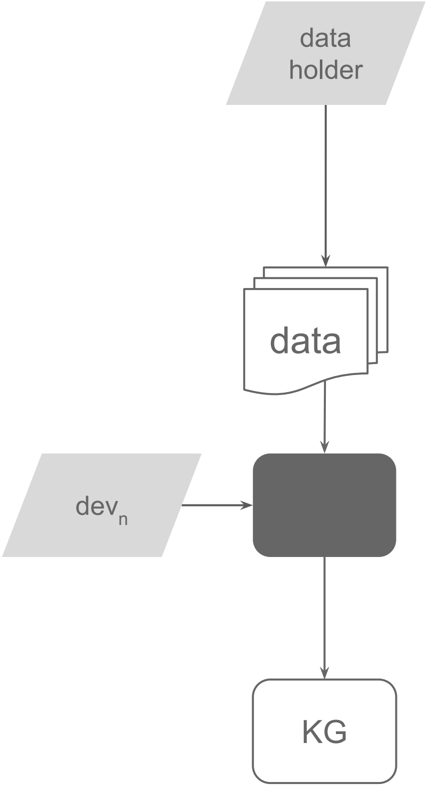
SPARQL Queries

Sources (S)

Declarative Mapping Rules (M)

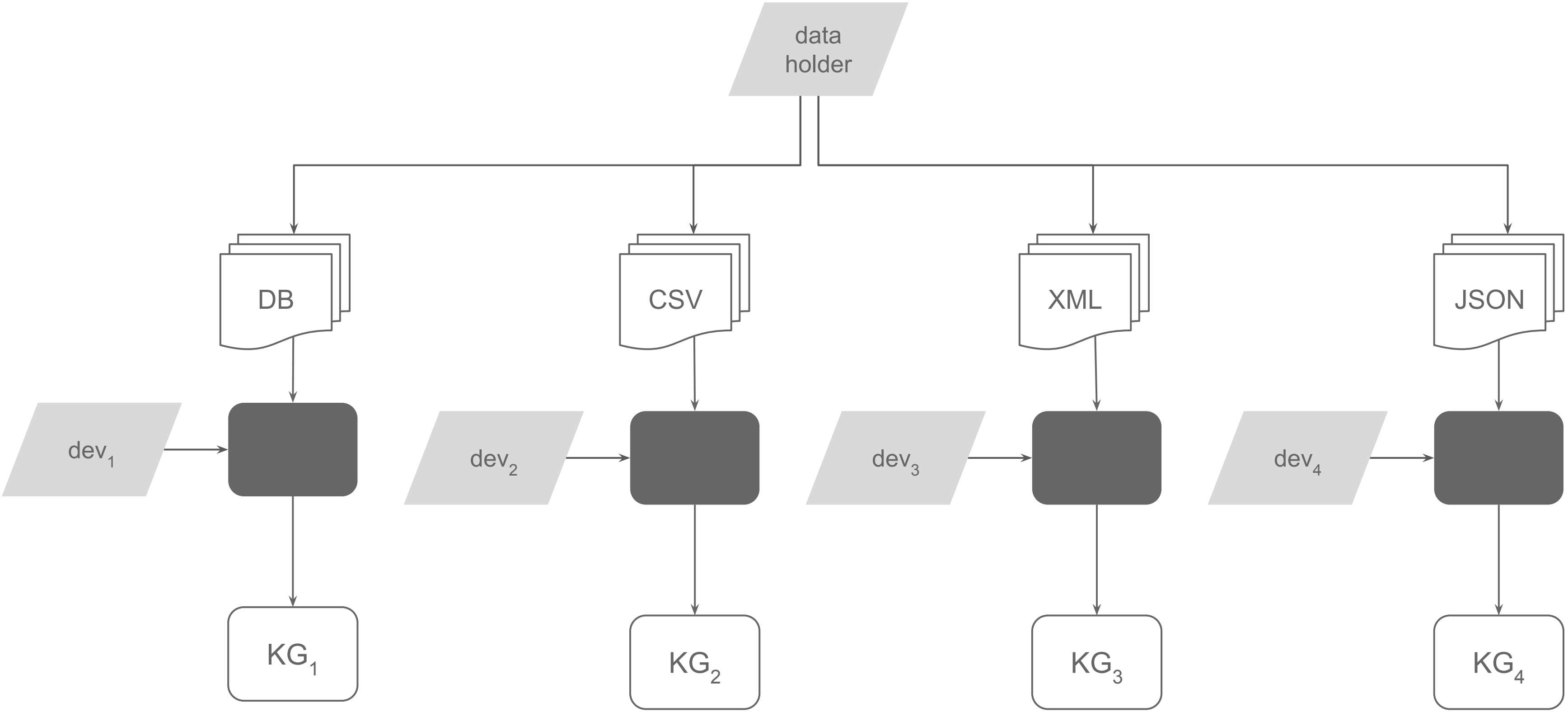
Declarative Knowledge Graph Construction Engine

Efficient	Scalable	Maintainable	Robust	Reproducible
🙋‍♀️	🙋‍♀️	🙋‍♀️	🙋‍♀️	🙋‍♀️

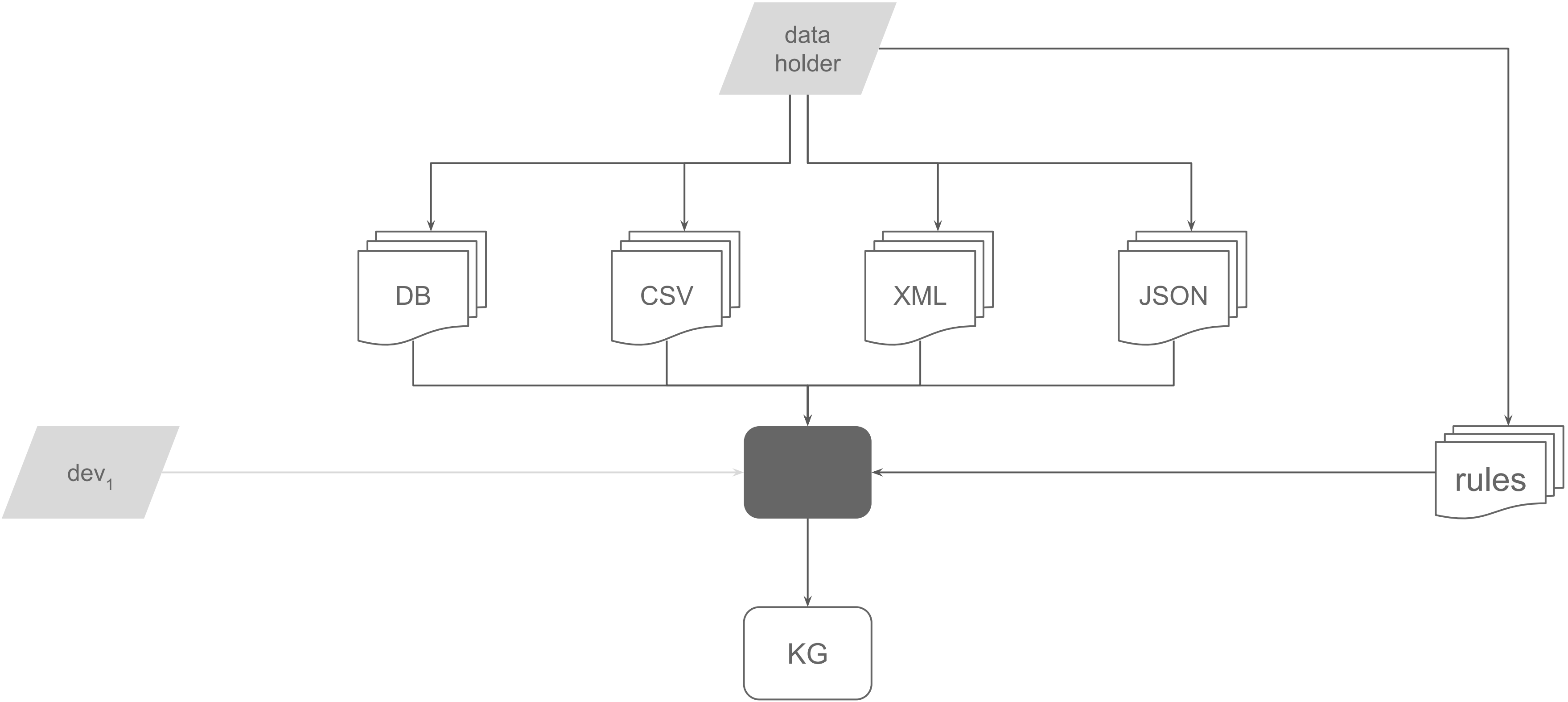


custom dedicated script for a data owner’s data

(-) new development cycle every time a modification is needed



dedicated tool for certain format
(+) great solution if a data owner has data only in a certain format
(-) learn and maintain multiple tools if a data owner has data in different formats
(-) post-processing step to integrate

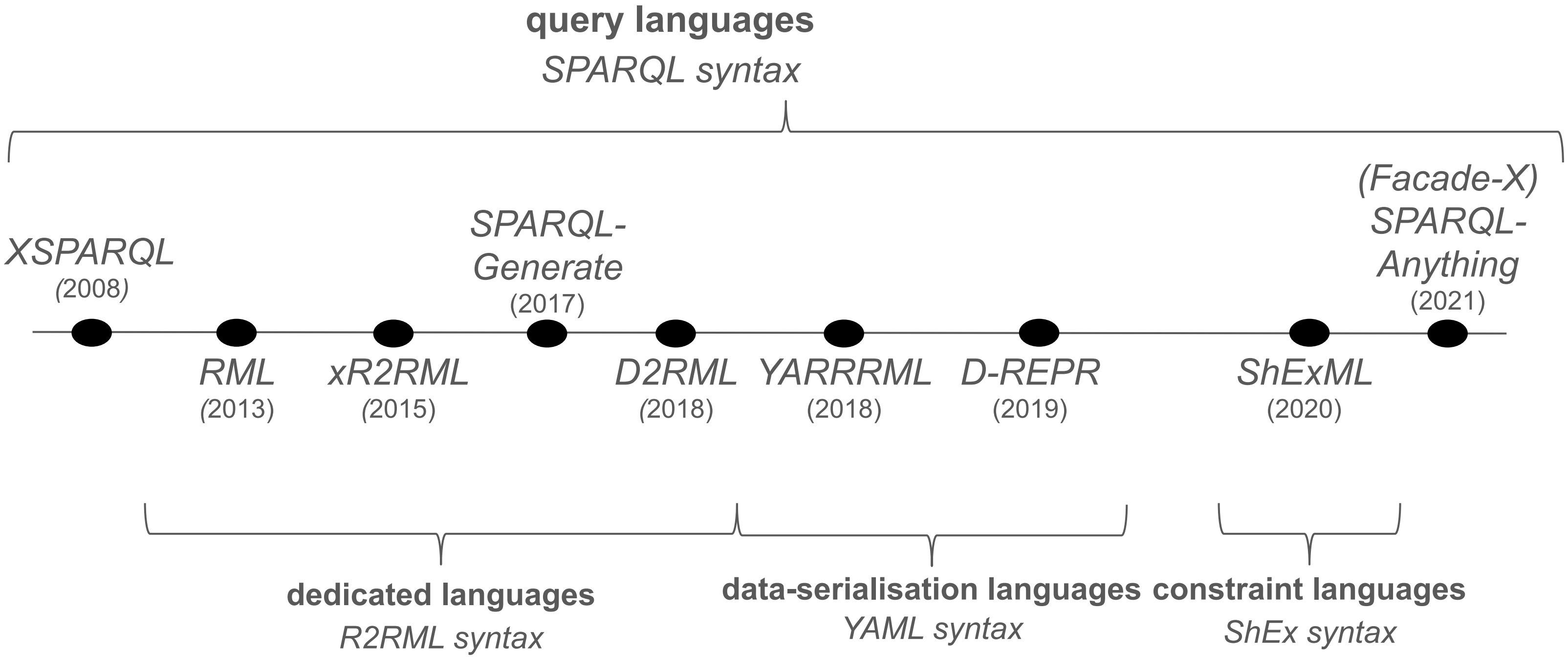


a tool for all data formats

(+) learn and maintain a single tool

(+) configure the rules that define how a KG is generated

Declarative mapping languages - *schema* transformations



query languages

SPARQL syntax

XSPARQL
(2008)

RML
(2013)

xR2RML
(2015)

SPARQL-Generate
(2017)

D2RML
(2018)

YARRRML
(2018)

D-REPR
(2019)

ShExML
(2020)

(Facade-X) SPARQL-Anything
(2021)

dedicated languages

R2RML syntax

data-serialisation languages

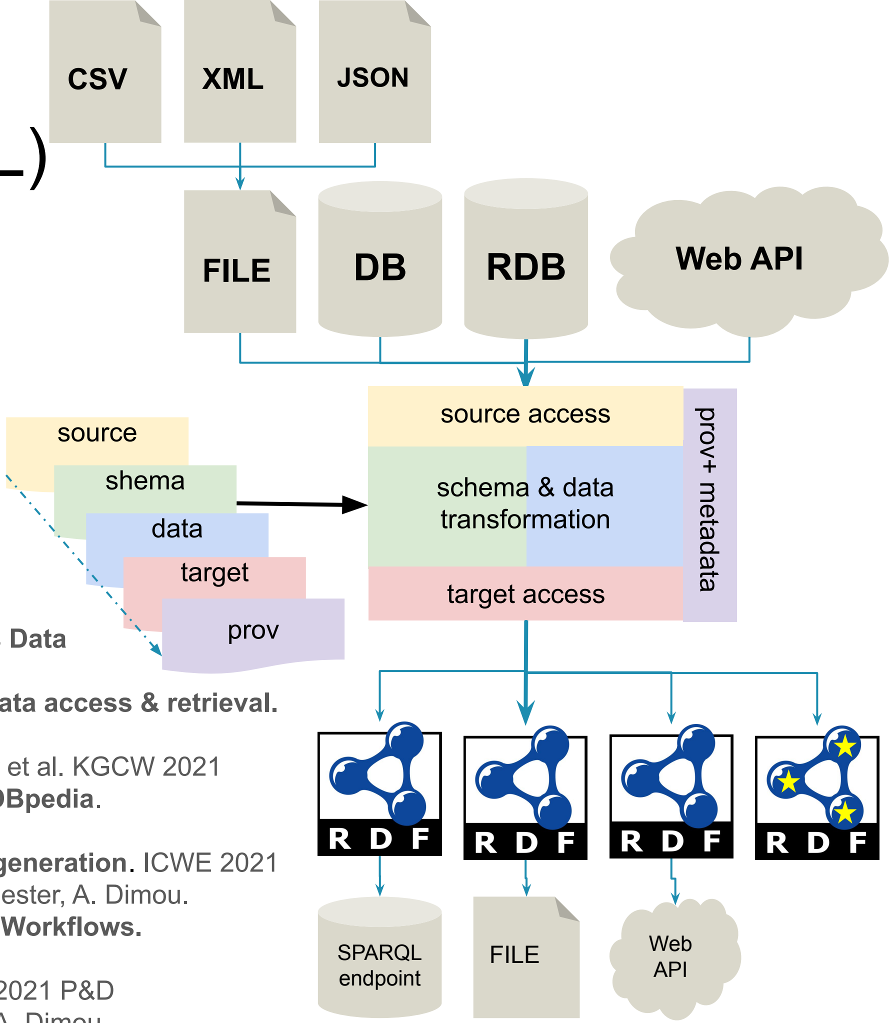
YAML syntax

constraint languages

ShEx syntax

Declarative RDF graph generation from heterogeneous (semi-)structured data: a Systematic Literature Review.
D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou. (JWS prepress)

RDF Mapping Language (RML)

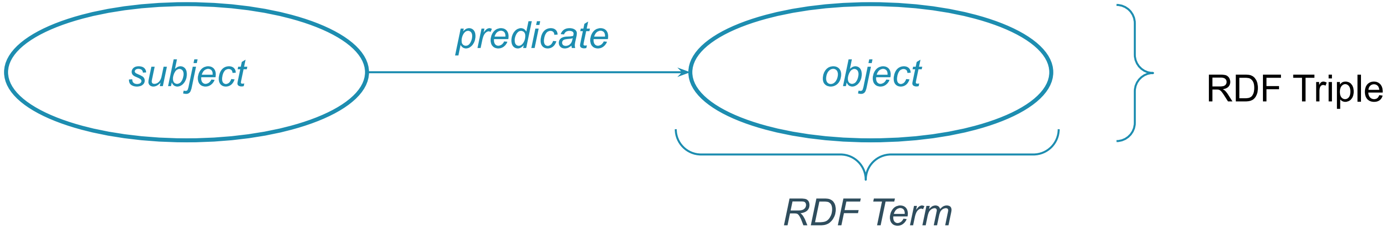


RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data
A. Dimou et al. LDOW 2014
Machine-interpretable dataset & service descriptions for heterogeneous data access & retrieval.
A. Dimou et al. SEMANTICS 2015
Mapping Spreadsheets to RDF: Supporting Excel in RML. Markus Schröder et al. KGCW 2021
Declarative data transformations for Linked Data generation: the case of DBpedia.
B.De Meester et al. ESWC 2017
Leveraging Web of Things W3C recommendations for knowledge graphs generation. ICWE 2021
D. Van Assche, G.Haesendonck, G. De Mulder, T. Delva, P. Heyvaert, B. De Meester, A. Dimou.
Automated Metadata Generation for Linked Data Generation & Publishing Workflows.
A. Dimou et al. LDOW 2016
RML-star: A declarative mapping language for RDF-star generation ISWC 2021 P&D
T. Delva, J. Arenas-Guerrero, A. Iglesias-Molina, O. Corcho, D. Chaves-Fraga, A. Dimou.

rank	name	nationality	mark	notes
1	Anzhelika Sidorova	Russia	4.95	WL,PB
2	Sandi Morris	USA	4.90	SB
3	Katerina Stefanidi	Greece	4.85	SB
4	Holly Bradshaw	UK	4.80	-
5	Alysha Newman	Canada	4.80	-
6	Angelica Bengtsson	Sweden	4.80	NR

Input:
- (semi-)structured raw data
- ontology/vocabulary terms
- mapping rules

Output:
- RDF graphs



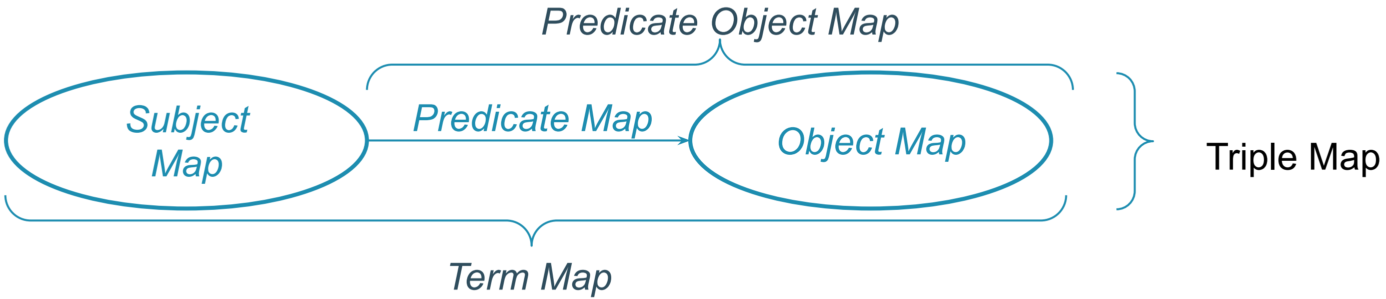
subject

predicate

object

RDF Triple

RDF Term



Predicate Object Map

Subject Map

Predicate Map

Object Map

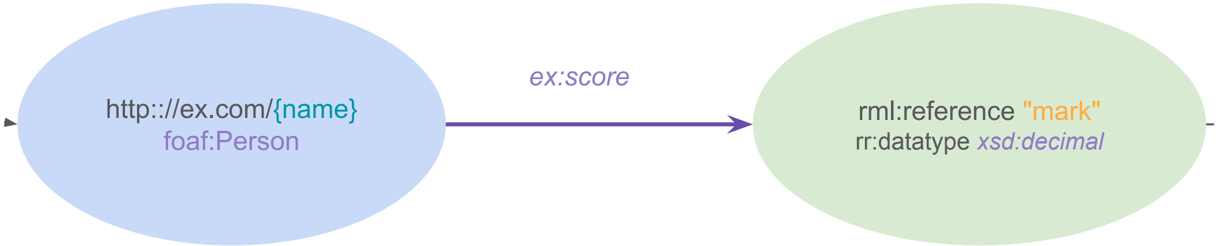
Triple Map

Term Map

rank	name	nationality	mark	notes
1	Anzhelika Sidorova	Russia	4.95	WL,PB
2	Sandi Morris	USA	4.90	SB
3	Katerina Stefanidi	Greece	4.85	SB
4	Holly Bradshaw	UK	4.80	-
5	Alysha Newman	Canada	4.80	-
6	Angelica Bengtsson	Sweden	4.80	NR

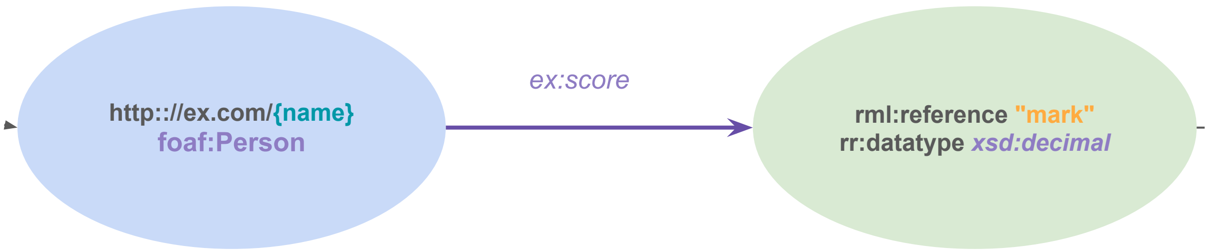
```
<#TriplesMap_1> [
  rr:subjectMap [...];
  rr:predicateObjectMap [
    rr:predicateMap [...];
    rr:objectMap [...]; ] ].
```

```
<http://ex.com/Anzhelika%20Sidorova> ex:score “4.95”^^xsd:decimal.
<http://ex.com/Sandi%20Morris> ex:score “4.90”^^xsd:decimal.
<http://ex.com/Katerina%20Stefanidi> ex:score “4.85”^^xsd:decimal.
<http://ex.com/Holly%20Bradshaw> ex:score “4.80”^^xsd:decimal.
<http://ex.com/Alysha%20Newman> ex:score “4.80”^^xsd:decimal.
<http://ex.com/Angelica%20Bengtsson> ex:score “4.80”^^xsd:decimal.
```



rank	name	nationality	mark	notes
1	Anzhelika Sidorova	Russia	4.95	WL,PB
2	Sandi Morris	USA	4.90	SB
3	Katerina Stefanidi	Greece	4.85	SB
4	Holly Bradshaw	UK	4.80	-
5	Alysha Newman	Canada	4.80	-
6	Angelica Bengtsson	Sweden	4.80	NR

```
<#TriplesMap_1> [
  rr:subjectMap [
    rr:template "http:://ex.com/{name}";
    rr:class foaf:Person; ];
  rr:predicateObjectMap [
    rr:predicateMap [rr:constant ex:score];
    rr:objectMap [ rml:reference "mark";
                   rr:datatype xsd:decimal]; ] ].
```

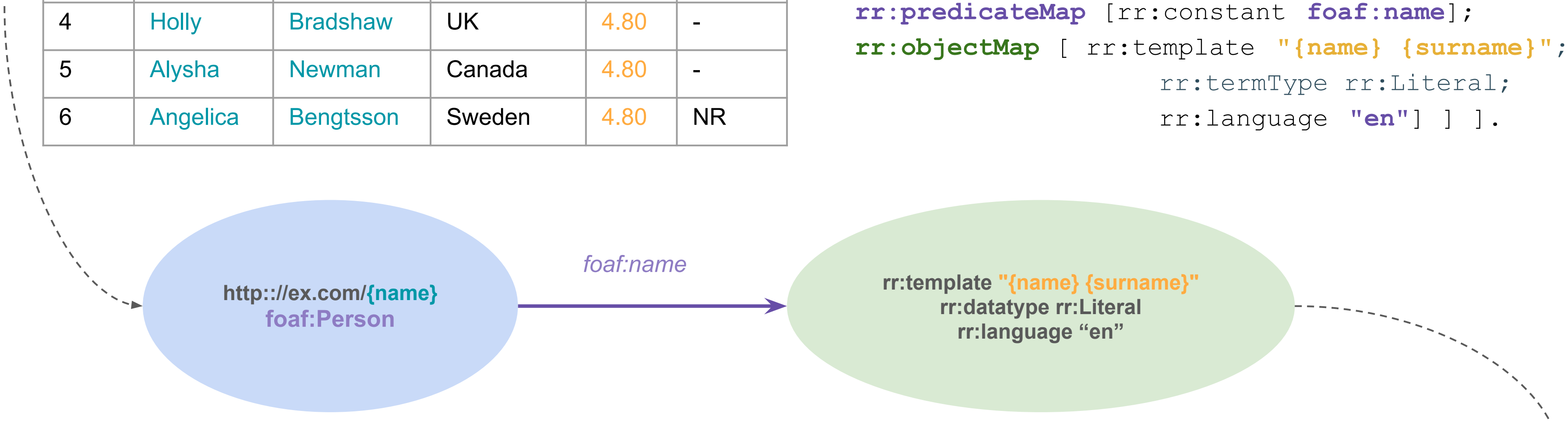


<http://ex.com/Anzhelika%20Sidorova> a foaf:Person.
<http://ex.com/Sandi%20Morris> a foaf:Person.
<http://ex.com/Katerina%20Stefanidi> a foaf:Person.
<http://ex.com/Holly%20Bradshaw> a foaf:Person.
<http://ex.com/Alysha%20Newman> a foaf:Person.
<http://ex.com/Angelica%20Bengtsson> a foaf:Person.

<http://ex.com/Anzhelika%20Sidorova> ex:score “4.95”^^xsd:decimal.
<http://ex.com/Sandi%20Morris> ex:score “4.90”^^xsd:decimal.
<http://ex.com/Katerina%20Stefanidi> ex:score “4.85”^^xsd:decimal.
<http://ex.com/Holly%20Bradshaw> ex:score “4.80”^^xsd:decimal.
<http://ex.com/Alysha%20Newman> ex:score “4.80”^^xsd:decimal.
<http://ex.com/Angelica%20Bengtsson> ex:score “4.80”^^xsd:decimal.

rank	name	surname	nationality	mark	notes
1	Anzhelika	Sidorova	Russia	4.95	WL,PB
2	Sandi	Morris	USA	4.90	SB
3	Katerina	Stefanidi	Greece	4.85	SB
4	Holly	Bradshaw	UK	4.80	-
5	Alysha	Newman	Canada	4.80	-
6	Angelica	Bengtsson	Sweden	4.80	NR

```
<#TriplesMap_1> [
  rr:subjectMap [
    rr:template "http:://ex.com/{name}";
    rr:class foaf:Person; ];
  rr:predicateObjectMap [
    rr:predicateMap [rr:constant foaf:name];
    rr:objectMap [ rr:template "{name} {surname}";
                   rr:termType rr:Literal;
                   rr:language "en"] ] ].
```



```
<http://ex.com/Anzhelika%20Sidorova> a foaf:Person.
<http://ex.com/Sandi%20Morris> a foaf:Person.
<http://ex.com/Katerina%20Stefanidi> a foaf:Person.
<http://ex.com/Holly%20Bradshaw> a foaf:Person.
<http://ex.com/Alysha%20Newman> a foaf:Person.
<http://ex.com/Angelica%20Bengtsson> a foaf:Person.
```

```
<http://ex.com/Anzhelika%20Sidorova> foaf:name “Anzhelika Sidorova”@en.
<http://ex.com/Sandi%20Morris> foaf:name "Sandi Morris"@en.
<http://ex.com/Katerina%20Stefanidi> foaf:name "Katerina Stefanidi"@en.
<http://ex.com/Holly%20Bradshaw> foaf:name  "Holly Bradshaw"@en.
<http://ex.com/Alysha%20Newman> foaf:name "Alysha Newman"@en .
<http://ex.com/Angelica%20Bengtsson> foaf:name "Angelica Bengtsson"@en .
```

rank	name	surname	nationality	mark	notes
1	Anzhelika	Sidorova	Russia	4.95	WL,PB
2	Sandi	Morris	USA	4.90	SB
3	Katerina	Stefanidi	Greece	4.85	SB
4	Holly	Bradshaw	UK	4.80	-
5	Alysha	Newman	Canada	4.80	-
6	Angelica	Bengtsson	Sweden	4.80	NR

```
<countries>
  <country continent="Europe">
    <country_abb>GR</country_abb>
    <country_name country_language="en">Greece</country_name>
    <country_name country_language="nl">Griekenland</country_name>
  </country>
  <country continent="Europe">
    <country_abb>UK</country_abb>
    <country_name country_language="en">United Kingdom</country_name>
    <country_name country_language="nl">Verenigd Koninkrijk</country_name>
  </country>
  <country continent="America">
    <country_abb>CA</country_abb>
    <country_name country_language="en">Canada</country_name>
    <country_name country_language="nl">Canada</country_name>
  </country>
...
</countries>
```

```
<#TriplesMap_1> [
  rr:predicateObjectMap [
    rr:predicateMap [rr:constant ex:country];
    rr:objectMap [ rr:parentTriplesMap <#TriplesMap_2>;
                   rr:joinCondition [
                              rr:parent "country_name";
                              rr:child "nationality"]]]].
```

```
<#TriplesMap_2> [
  rml:logicalSource [
    rml:source "countries.xml";
    rml:referenceFormulation ql:XPath;
    rml:iterator "countries/country" ];
  rr:subjectMap [
    rr:template "http:://ex.com/{country_abb}"; ].
```

```
<http://ex.com/Anzhelika%20Sidorova> ex:country <http://ex.com/RU>.
<http://ex.com/Sandi%20Morris> ex:countrly <http://ex.com/US>.
<http://ex.com/Katerina%20Stefanidi> ex:country <http://ex.com/EL>.
<http:://ex.com/Holly%20Bradshaw> ex:country <http://ex.com/UK>.
<http:://ex.com/Alysha%20Newman> ex:country <http://ex.com/CA>.
<http:://ex.com/Angelica%20Bengtsson> ex:country <http://ex.com/SE>.
```

User Interfaces

Matey https://github.com/rmlio/matey

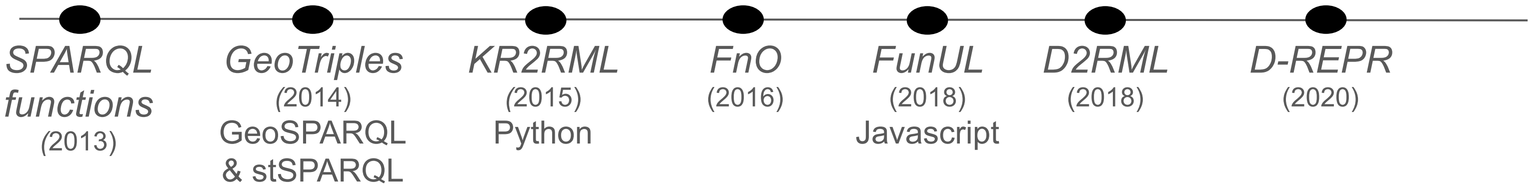
Mapeathor https://morph.oeg.fi.upm.es/tool/mapeathor, https://github.com/oeg-upm/morph-website

RMLEditor https://app.rml.io/rmleditor/, https://rml.io/tools/rmleditor/, https://github.com/RMLio/rmleditor-ce

Map-On http://semanco-tools.eu/map-on, https://github.com/arc-lasalle/Map-On

RMLx Visual Editor http://pebbie.org/mashup/rml

Declarative mapping languages - *data* transformations



SPARQL functions (2013)

GeoTriples (2014) GeoSPARQL & stSPARQL

KR2RML (2015) Python

FnO (2016)

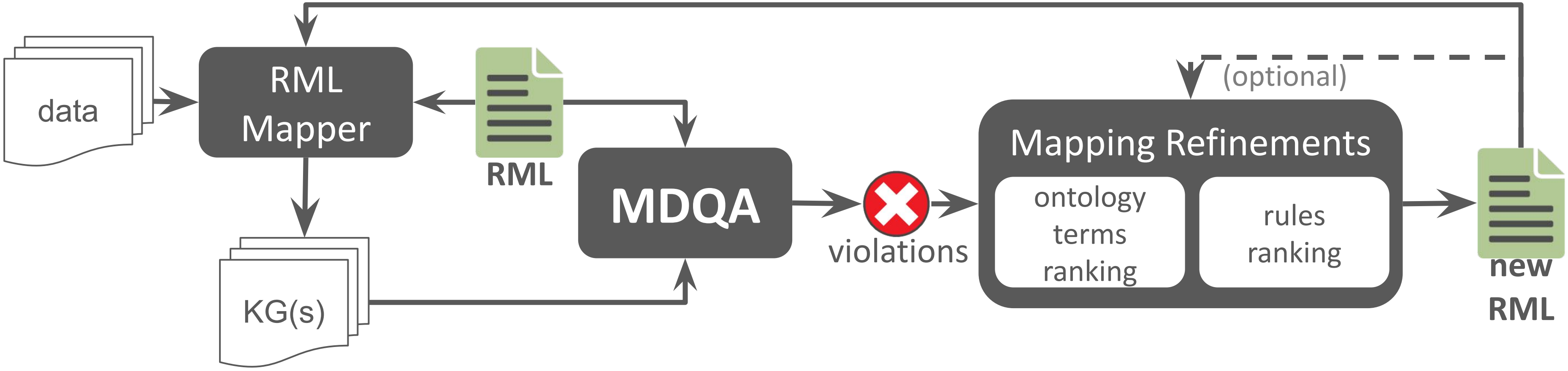
FunUL (2018) Javascript

D2RML (2018)

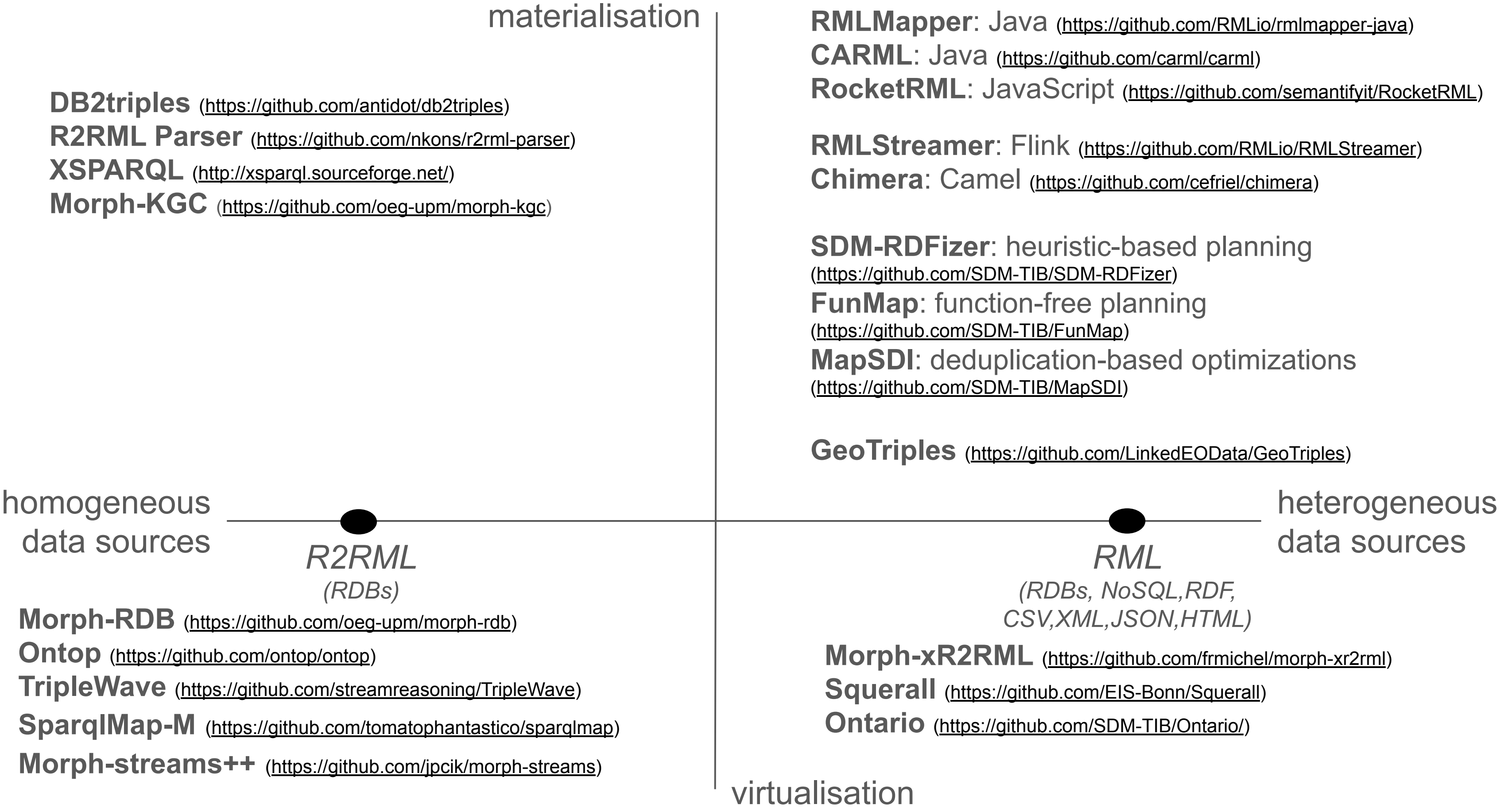
D-REPR (2020)

Declarative RDF graph generation from heterogeneous (semi-)structured data: a Systematic Literature Review.
D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou. (JWS prepress)

Validation



Assessing and Refining Mappings to RDF to Improve Dataset Quality. A. Dimou et al. ISWC 2015
Rule-driven inconsistency resolution for knowledge graph generation rules P. Heyvaert A. Dimou, B. De Meester, R. Verborgh. SWJ 2019



materialisation

virtualisation

homogeneous data sources

heterogeneous data sources

R2RML

(RDBs)

RML

(RDBs, NoSQL,RDF, CSV,XML,JSON,HTML)

DB2triples (https://github.com/antidot/db2triples)
R2RML Parser (https://github.com/nkons/r2rml-parser)
XSPARQL (http://xsparql.sourceforge.net/)
Morph-KGC (https://github.com/oeg-upm/morph-kgc)

RMLMapper: Java (https://github.com/RMLio/rmlmapper-java)
CARML: Java (https://github.com/carml/carml)
RocketRML: JavaScript (https://github.com/semantifyit/RocketRML)

RMLStreamer: Flink (https://github.com/RMLio/RMLStreamer)
Chimera: Camel (https://github.com/cefriel/chimera)

SDM-RDFizer: heuristic-based planning (https://github.com/SDM-TIB/SDM-RDFizer)
FunMap: function-free planning (https://github.com/SDM-TIB/FunMap)
MapSDI: deduplication-based optimizations (https://github.com/SDM-TIB/MapSDI)

GeoTriples (https://github.com/LinkedEOData/GeoTriples)

Morph-RDB (https://github.com/oeg-upm/morph-rdb)
Ontop (https://github.com/ontop/ontop)
TripleWave (https://github.com/streamreasoning/TripleWave)
SparqlMap-M (https://github.com/tomatophantastico/sparqlmap)
Morph-streams++ (https://github.com/jpcik/morph-streams)

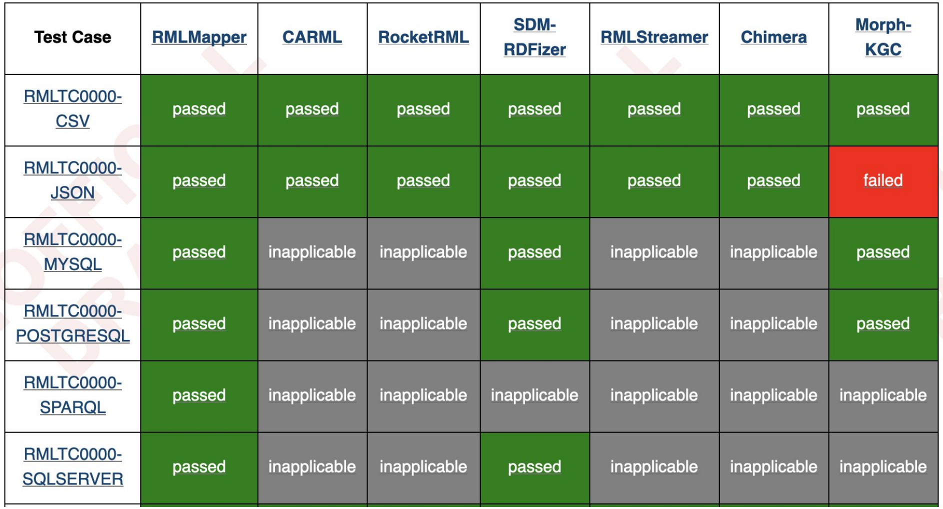
Morph-xR2RML (https://github.com/frmichel/morph-xr2rml)
Squerall (https://github.com/EIS-Bonn/Squerall)
Ontario (https://github.com/SDM-TIB/Ontario/)

Declarative RDF graph generation from heterogeneous (semi-)structured data: a Systematic Literature Review.
D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou. (JWS under review after major revision)

Test Cases - Implementation Reports - Benchmarks

Choose yourself the best tool for your needs!

http://rml.io/test-cases/
http://rml.io/implementation-report/



Test Case	RMLMapper	CARML	RocketRML	SDM-RDFizer	RMLStreamer	Chimera	Morph-KGC
RMLTC0000-CSV	passed	passed	passed	passed	passed	passed	passed
RMLTC0000-JSON	passed	passed	passed	passed	passed	passed	failed
RMLTC0000-MYSQL	passed	inapplicable	inapplicable	passed	inapplicable	inapplicable	passed
RMLTC0000-POSTGRESQL	passed	inapplicable	inapplicable	passed	inapplicable	inapplicable	passed
RMLTC0000-SPARQL	passed	inapplicable	inapplicable	inapplicable	inapplicable	inapplicable	inapplicable
RMLTC0000-SQLSERVER	passed	inapplicable	inapplicable	passed	inapplicable	inapplicable	inapplicable

Conformance test-cases for the RDF Mapping Language. P. Heyvaert, D. Chaves-Fraga, F. Priyatna, O. Corcho, E. Mannens, R. Verborgh, A. Dimou. KGSWC2019

Benchmarks:

GTFS: evaluate tools generating RDF graphs with RML mapping rules (https://github.com/oeg-upm/gtfs-bench)

RODI: test the quality of (semi-)automatically generated mapping rules (https://github.com/chrpin/rodi)

RODI: Benchmarking Relational-to-Ontology Mapping Generation Quality. Pinkel et al. SWJ 2016
GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain Chaves-Fraga et al. JWS 2020

KG4DI

YARRRML

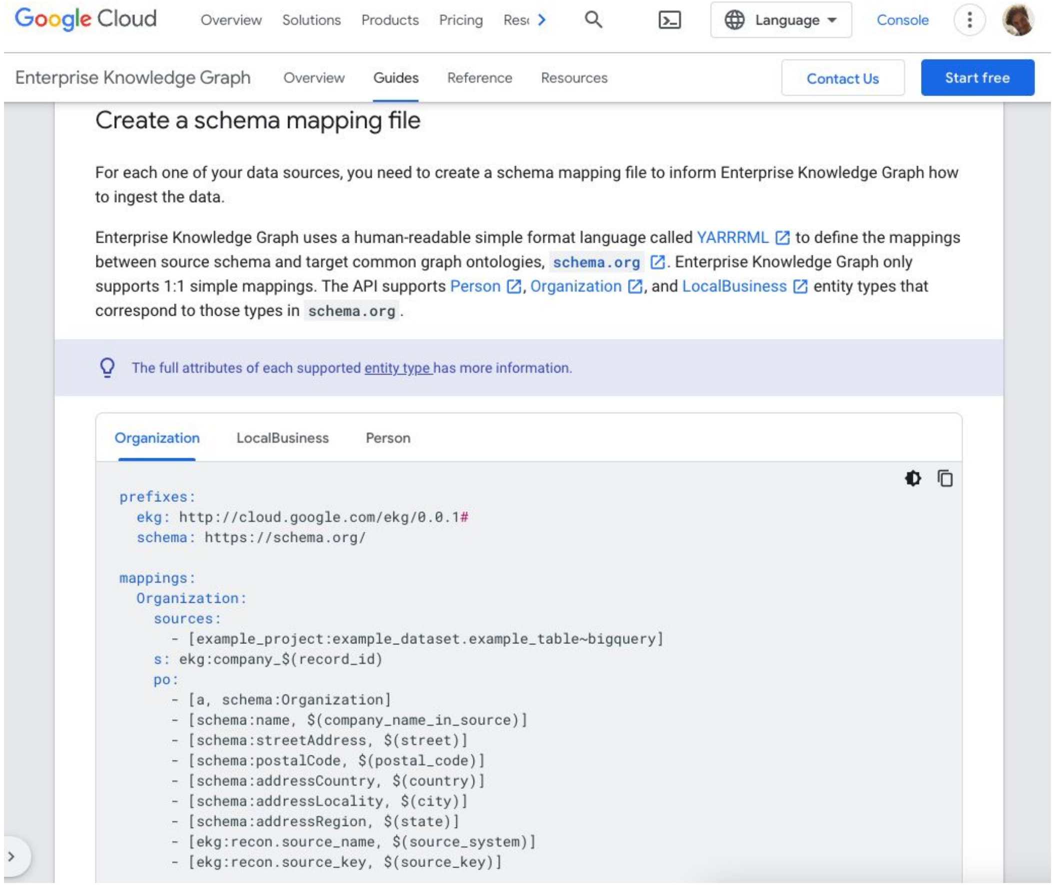
Dylan Van Assche

✉ dylan.vanassche@ugent.be

@dylanvanassche@fosstodon.org

https://dylanvanassche.be

Why you should involve YARRRML in your KG pipeline?



Create a schema mapping file

For each one of your data sources, you need to create a schema mapping file to inform Enterprise Knowledge Graph how to ingest the data.

Enterprise Knowledge Graph uses a human-readable simple format language called YARRRML to define the mappings between source schema and target common graph ontologies, `schema.org`. Enterprise Knowledge Graph only supports 1:1 simple mappings. The API supports Person, Organization, and LocalBusiness entity types that correspond to those types in `schema.org`.

The full attributes of each supported entity type has more information.

```
prefixes:
  ekg: http://cloud.google.com/ekg/0.0.1#
  schema: https://schema.org/

mappings:
  Organization:
    sources:
      - [example_project:example_dataset.example_table~bigquery]
    s: ekg:company_$(record_id)
    po:
      - [a, schema:Organization]
      - [schema:name, $(company_name_in_source)]
      - [schema:streetAddress, $(street)]
      - [schema:postalCode, $(postal_code)]
      - [schema:addressCountry, $(country)]
      - [schema:addressLocality, $(city)]
      - [schema:addressRegion, $(state)]
      - [ekg:recon.source_name, $(source_system)]
      - [ekg:recon.source_key, $(source_key)]
```

Mapping rules are not human-friendly

RDF is great for machines but too verbose for humans

A mistake in your RDF syntax is hard to spot if you are not familiar with RDF

Harder to read than JSON or YAML

```
@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 transit: <http://vocab.org/transit/terms/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix wgs84_pos:
<http://www.w3.org/2003/01/geo/wgs84_pos#>.
@base <http://example.com/ns#>.

<#AirportMapping> a rr:TriplesMap;
  rml:logicalSource [
    rml:source "Airport.csv" ;
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
    rr:template "http://airport.example.com/{id}";
    rr:class transit:Stop
  ];
  rr:predicateObjectMap [
    rr:predicate transit:route;
    rr:objectMap [
      rml:reference "stop";
      rr:datatype xsd:int
    ]
  ];
.
```

More human-friendly mapping rules?

YAML is widely known among humans and is integrated in developer tools

YAML is easier to write than RDF

https://yaml.org

RML mapping rules in RDF Turtle

RML is widely used, but hard to write if not familiar with Turtle & RML

https://rml.io/spec

YARRRML

https://rml.io/yarrrml/spec

YARRRML is a human-friendly representation of mapping rules

YAML-based

YARRRML is a subset of YAML

Human-friendly representation

Human-friendly representation of RML mapping rules

Specification

YARRRML has its own specification

https://rml.io/yarrrml/spec/

YARRRML

https://rml.io/yarrrml/

```
<#TriplesMap_1> [
  rml:logicalSource [
    rml:source "poleVaulters.csv";
    rml:referenceFormulation ql:CSV; ]; ]
  rr:subjectMap [
    rr:template "http:://ex.com/{name}"; ];
  rr:predicateObjectMap [
    rr:predicateMap [rr:constant ex:score];
    rr:objectMap [ rml:reference "mark";
                        rr:datatype xsd:decimal]; ];
  rr:predicateObjectMap [
    rr:predicateMap [rr:constant foaf:name];
    rr:objectMap [ rml:reference "name"; rr:language "en"]; ];
  rr:predicateObjectMap [
    rr:predicateMap [rr:constant ex:country];
    rr:objectMap [ rr:parentTriplesMap <#TriplesMap_2>;
                   rr:joinCondition [
                         rr:parent "country_name";
                         rr:child "nationality"] ]; ] ].
  <#TriplesMap_2> [
  rml:logicalSource [
    rml:source "countries.xml";
    rml:referenceFormulation ql:XPath;
    rml:iterator "countries/country" ];
  rr:subjectMap [
    rr:template "http:://ex.com/{country_abb}";
    rr:graphMap [ rr:constant ex:CountryGraph ]; ].
```

RML

```
mapping:
  person:
    sources:
      - [poleVaulters.csv~csv]
    subjects:
     - value: "http:://ex.com/{name}"
    predicateobjects:
      - [ex:score, $(mark), xsd:decimal]
      - [foaf:name, $(name), en~lang]
      - [foaf:name, $(name) $(surname), en~lang]
      - predicates: ex:country
        objects:
          - mapping: country
            condition:
              function: equal
              parameters:
                - [str1, $(nationality), s]
                - [str2, $(country_name), o]
  country:
    subjects: http:://ex.com/{country_abb}
```

YARRRML

YARRRML is a human-friendly representation of mapping rules

Tooling
yarrrml-parser converts YARRRML
from and to RML. Matey provides a browser editor

Compatible
Works with any RML processor such as RMLMapper,
RMLStreamer, SDM-RDFizer, RocketRML, etc.

Battle-tested
Used in several projects, with more 1000 YARRRML mappings

https://github.com/rmlio/yarrrml-parser
https://rml.io/yarrrml/matey/

Matey is a browser application to write & test YARRRML rules

YARRRML editor

Matey is a YARRRML editor in your browser

Same tooling

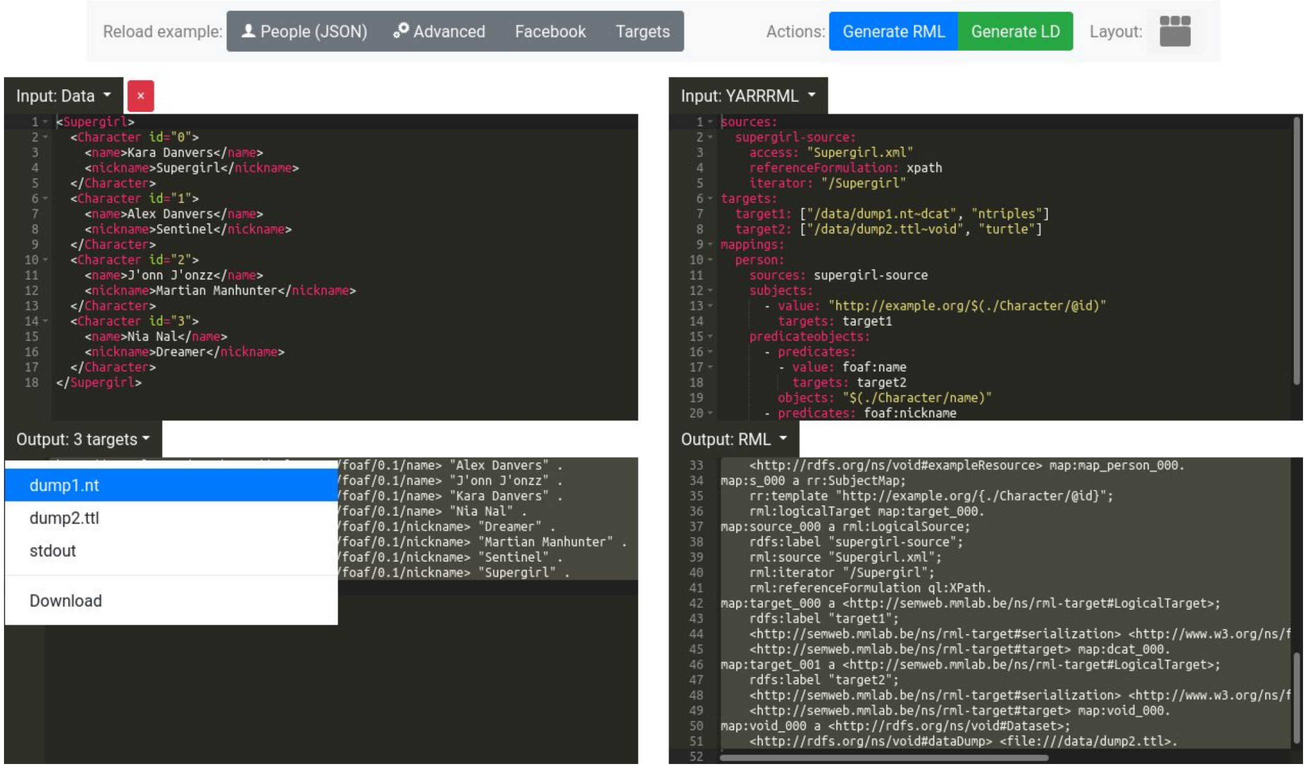
Uses yarrrml-parser and RMLMapper behind the scenes, no surprises in production!

Quickly prototyping

Matey executes YARRRML rules on your data.
The generated Linked Data is directly shown in your Matey

https://github.com/rmlio/matey
https://rml.io/yarrrml/matey/

Matey UI: input data, YARRRML rules, output, and RML rules



```
<Supergirl>
  <Character id="0">
    <name>Kara Danvers</name>
    <nickname>Supergirl</nickname>
  </Character>
  <Character id="1">
    <name>Alex Danvers</name>
    <nickname>Sentinel</nickname>
  </Character>
  <Character id="2">
    <name>J'onn J'onzz</name>
    <nickname>Martian Manhunter</nickname>
  </Character>
  <Character id="3">
    <name>Nia Nal</name>
    <nickname>Dreamer</nickname>
  </Character>
</Supergirl>
```

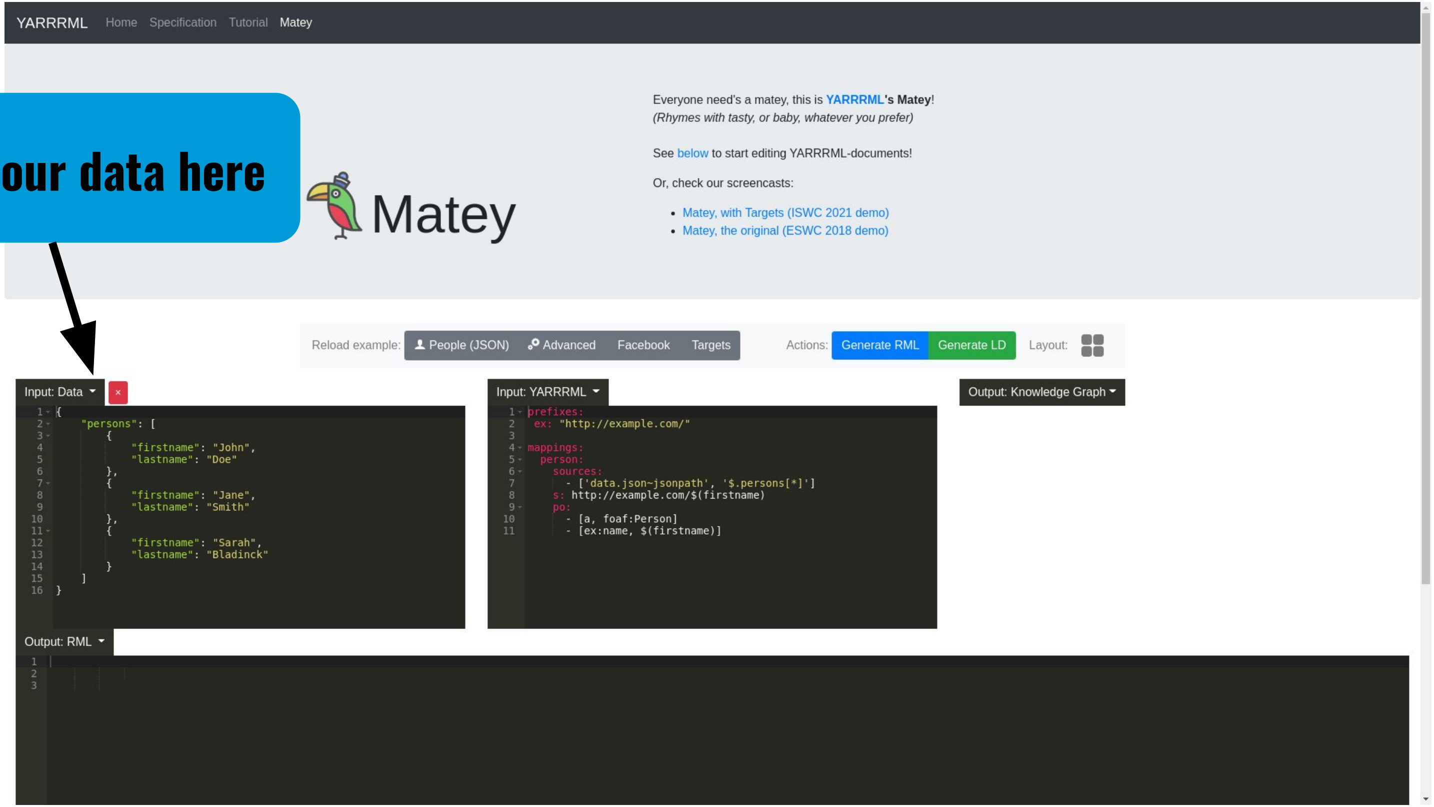
```
sources:
  supergirl-source:
    access: "Supergirl.xml"
    referenceFormulation: xpath
    iterator: "/Supergirl"
targets:
  target1: ["/data/dump1.nt~dcat", "ntriples"]
  target2: ["/data/dump2.ttl~void", "turtle"]
mappings:
  person:
    sources: supergirl-source
    subjects:
      - value: "http://example.org/$(./Character/@id)"
        targets: target1
    predicateobjects:
      - predicates:
          - value: foaf:name
            targets: target2
        objects: "$(./Character/name)"
      - predicates: foaf:nickname
```

```
/foaf/0.1/name> "Alex Danvers" .
/foaf/0.1/name> "J'onn J'onzz" .
/foaf/0.1/name> "Kara Danvers" .
/foaf/0.1/name> "Nia Nal" .
/foaf/0.1/nickname> "Dreamer" .
/foaf/0.1/nickname> "Martian Manhunter" .
/foaf/0.1/nickname> "Sentinel" .
/foaf/0.1/nickname> "Supergirl" .
```

```
    <http://rdfs.org/ns/void#exampleResource> map:map_person_000.
map:s_000 a rr:SubjectMap;
    rr:template "http://example.org/{./Character/@id}";
    rml:logicalTarget map:target_000.
map:source_000 a rml:LogicalSource;
    rdfs:label "supergirl-source";
    rml:source "Supergirl.xml";
    rml:iterator "/Supergirl";
    rml:referenceFormulation ql:XPath.
map:target_000 a <http://semweb.mmlab.be/ns/rml-target#LogicalTarget>;
    rdfs:label "target1";
    <http://semweb.mmlab.be/ns/rml-target#serialization> <http://www.w3.org/ns/f
    <http://semweb.mmlab.be/ns/rml-target#target> map:dcat_000.
map:target_001 a <http://semweb.mmlab.be/ns/rml-target#LogicalTarget>;
    rdfs:label "target2";
    <http://semweb.mmlab.be/ns/rml-target#serialization> <http://www.w3.org/ns/f
    <http://semweb.mmlab.be/ns/rml-target#target> map:void_000.
map:void_000 a <http://rdfs.org/ns/void#Dataset>;
    <http://rdfs.org/ns/void#dataDump> <file:///data/dump2.ttl>.
```

Hands-on: demo

1. Add your data here



Everyone need's a matey, this is **YARRRML's Matey**!
(Rhymes with tasty, or baby, whatever you prefer)

See below to start editing YARRRML-documents!

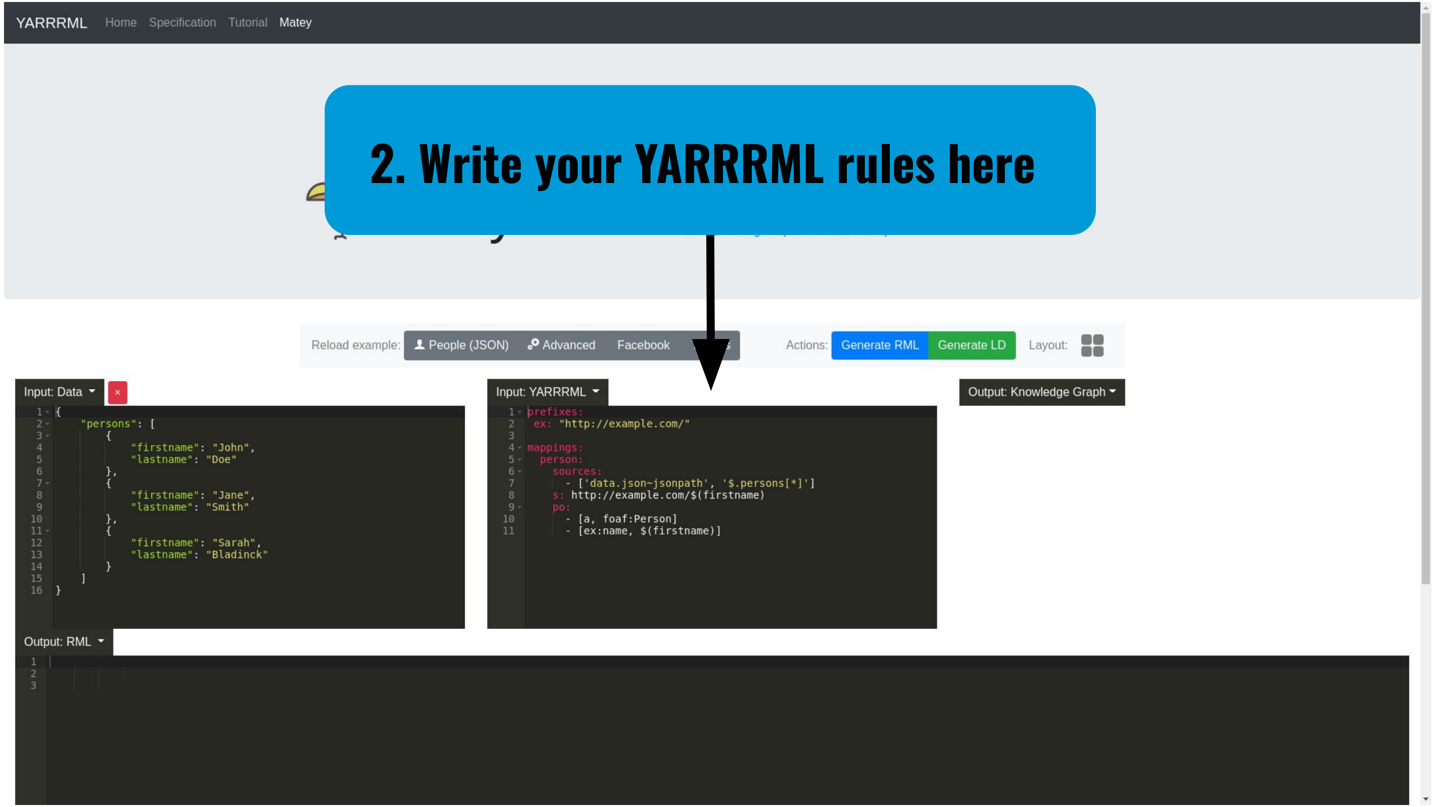
Or, check our screencasts:

- Matey, with Targets (ISWC 2021 demo)
- Matey, the original (ESWC 2018 demo)

```
{
    "persons": [
        {
            "firstname": "John",
            "lastname": "Doe"
        },
        {
            "firstname": "Jane",
            "lastname": "Smith"
        },
        {
            "firstname": "Sarah",
            "lastname": "Bladinck"
        }
    ]
}
```

```
prefixes:
 ex: "http://example.com/"

mappings:
  person:
    sources:
      - ['data.json~jsonpath', '$.persons[*]']
    s: http://example.com/$(firstname)
    po:
      - [a, foaf:Person]
      - [ex:name, $(firstname)]
```

Hands-on: demo

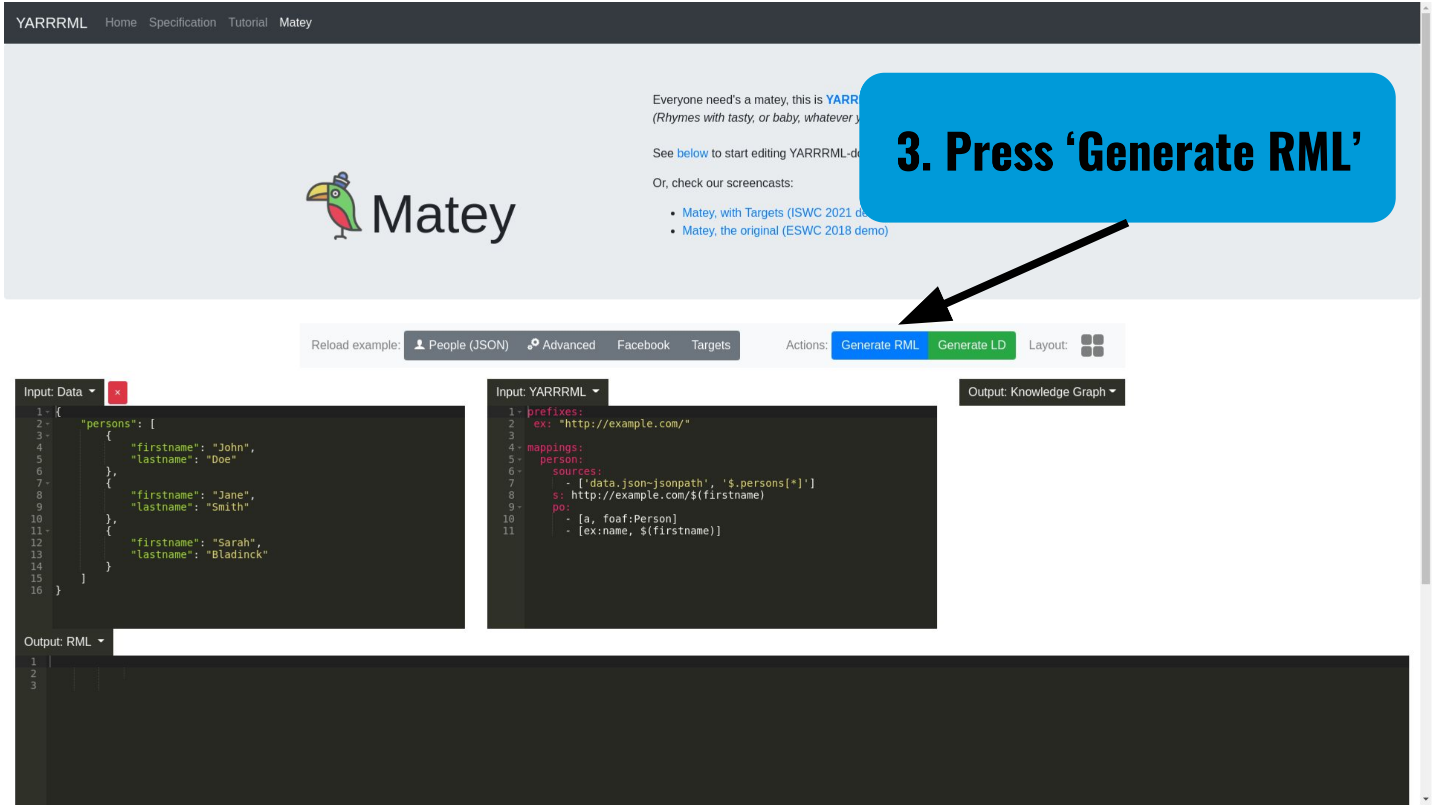


2. Write your YARRRML rules here

```
{
    "persons": [
        {
            "firstname": "John",
            "lastname": "Doe"
        },
        {
            "firstname": "Jane",
            "lastname": "Smith"
        },
        {
            "firstname": "Sarah",
            "lastname": "Bladinck"
        }
    ]
}
```

```
prefixes:
 ex: "http://example.com/"

mappings:
  person:
    sources:
      - ['data.json~jsonpath', '$.persons[*]']
    s: http://example.com/$(firstname)
    po:
      - [a, foaf:Person]
      - [ex:name, $(firstname)]
```

Hands-on: demo



Everyone need's a matey, this is YARR
(Rhymes with tasty, or baby, whatever y

See below to start editing YARRRML-d

Or, check our screencasts:

- Matey, with Targets (ISWC 2021 de
- Matey, the original (ESWC 2018 demo)

Input: Data

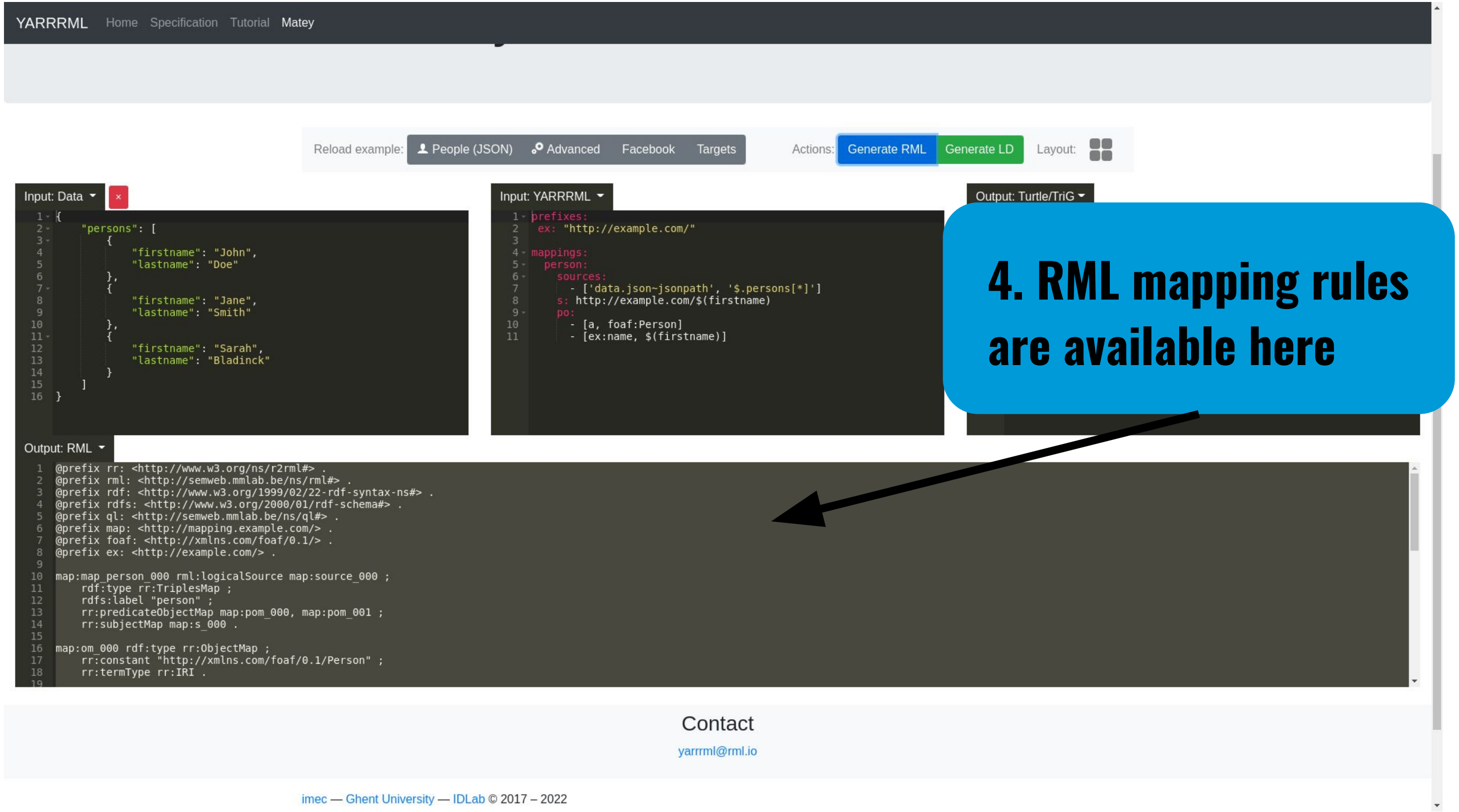
```
{
    "persons": [
        {
            "firstname": "John",
            "lastname": "Doe"
        },
        {
            "firstname": "Jane",
            "lastname": "Smith"
        },
        {
            "firstname": "Sarah",
            "lastname": "Bladinck"
        }
    ]
}
```

Input: YARRRML

```
prefixes:
 ex: "http://example.com/"

mappings:
  person:
    sources:
      - ['data.json~jsonpath', '$.persons[*]']
    s: http://example.com/$(firstname)
    po:
      - [a, foaf:Person]
      - [ex:name, $(firstname)]
```

3. Press 'Generate RML'

Hands-on: demo



```
{
    "persons": [
        {
            "firstname": "John",
            "lastname": "Doe"
        },
        {
            "firstname": "Jane",
            "lastname": "Smith"
        },
        {
            "firstname": "Sarah",
            "lastname": "Bladinck"
        }
    ]
}
```

```
prefixes:
 ex: "http://example.com/"

mappings:
  person:
    sources:
      - ['data.json~jsonpath', '$.persons[*]']
    s: http://example.com/$(firstname)
    po:
      - [a, foaf:Person]
      - [ex:name, $(firstname)]
```

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix map: <http://mapping.example.com/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .

map:map_person_000 rml:logicalSource map:source_000 ;
    rdf:type rr:TriplesMap ;
    rdfs:label "person" ;
    rr:predicateObjectMap map:pom_000, map:pom_001 ;
    rr:subjectMap map:s_000 .

map:om_000 rdf:type rr:ObjectMap ;
    rr:constant "http://xmlns.com/foaf/0.1/Person" ;
    rr:termType rr:IRI .
```

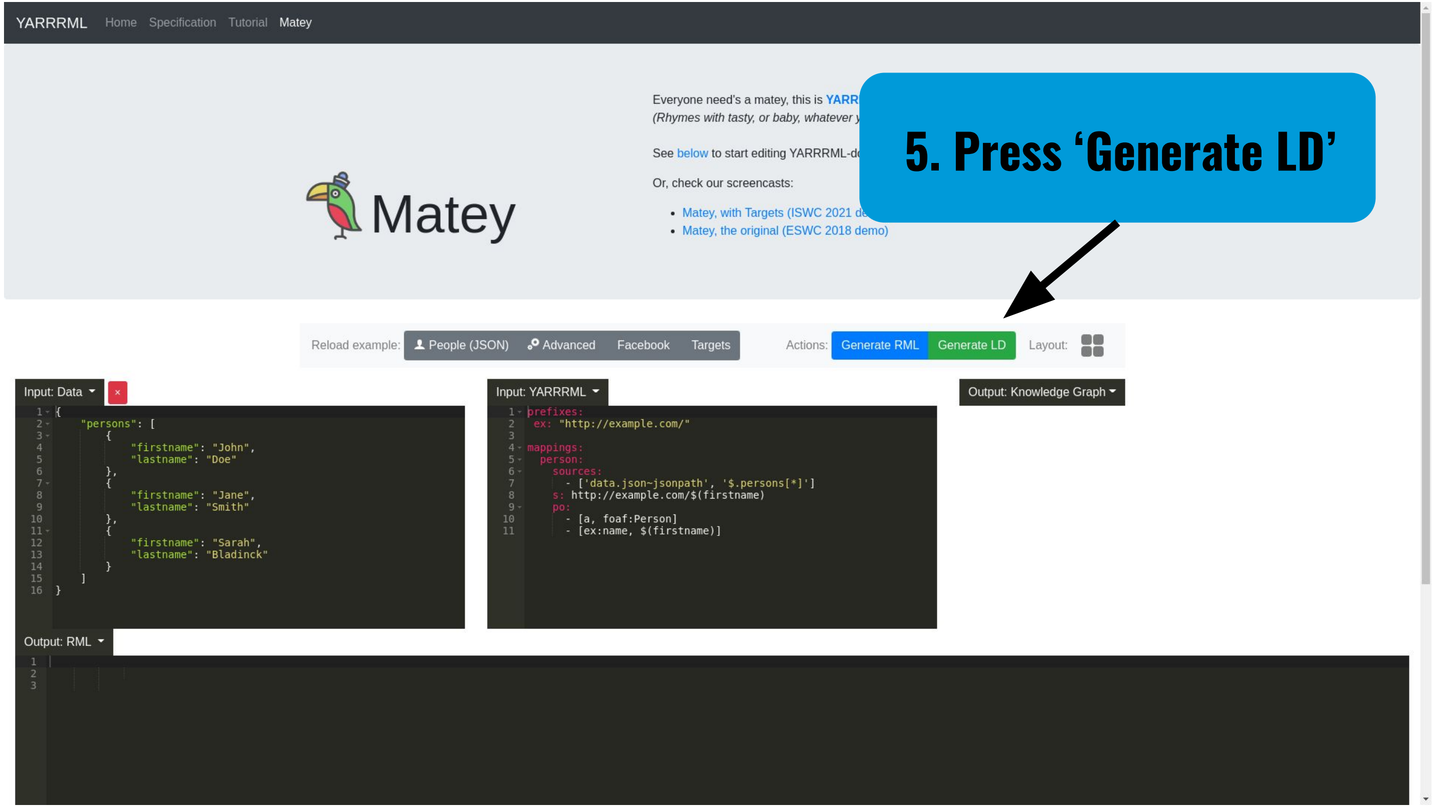
Contact

yarrrml@rml.io

imec — Ghent University — IDLab © 2017 – 2022

4. RML mapping rules are available here

Hands-on: demo



Everyone need's a matey, this is YARR
(Rhymes with tasty, or baby, whatever y

See below to start editing YARRRML-d

Or, check our screencasts:

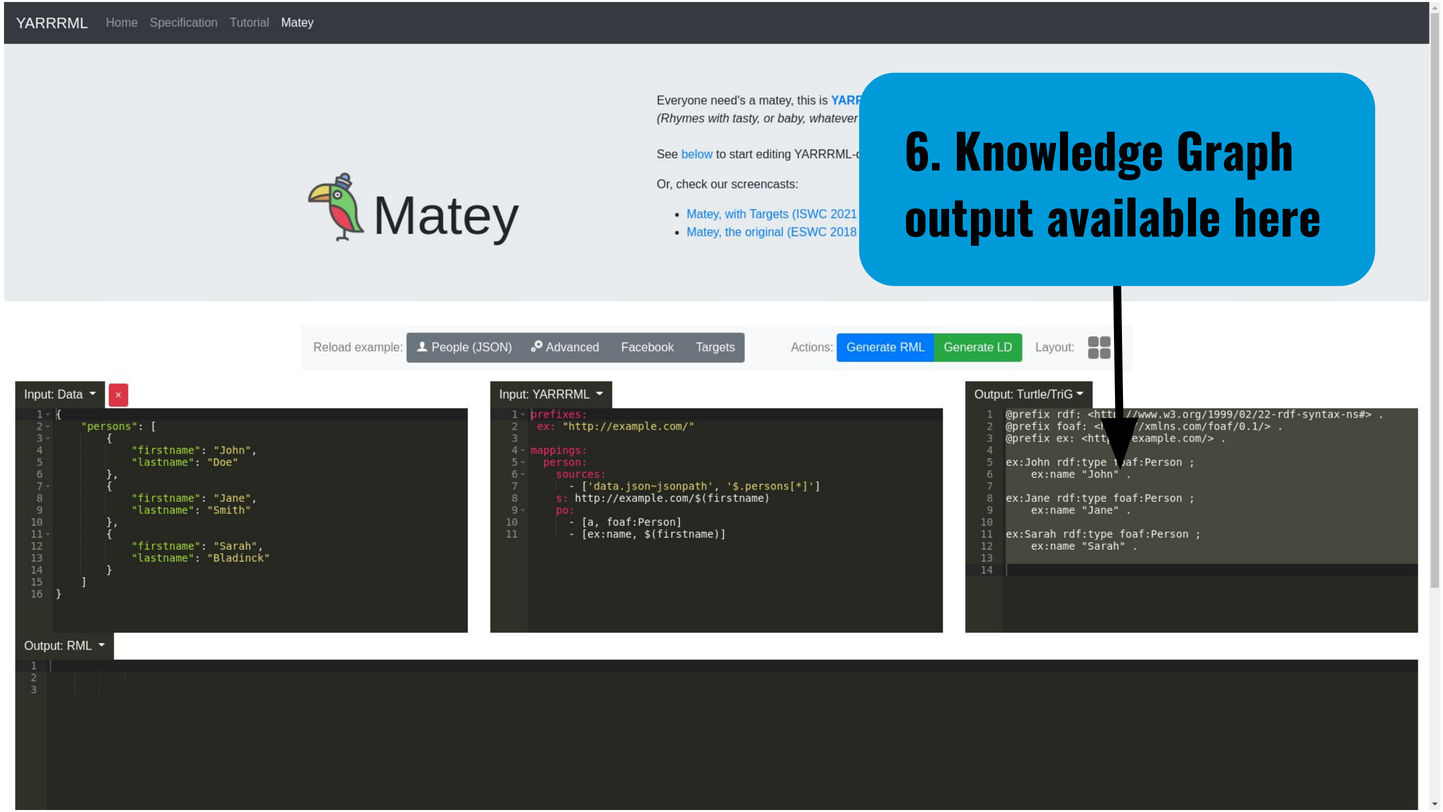
- Matey, with Targets (ISWC 2021 de
- Matey, the original (ESWC 2018 demo)

```
{
    "persons": [
        {
            "firstname": "John",
            "lastname": "Doe"
        },
        {
            "firstname": "Jane",
            "lastname": "Smith"
        },
        {
            "firstname": "Sarah",
            "lastname": "Bladinck"
        }
    ]
}
```

```
prefixes:
 ex: "http://example.com/"

mappings:
  person:
    sources:
      - ['data.json~jsonpath', '$.persons[*]']
    s: http://example.com/$(firstname)
    po:
      - [a, foaf:Person]
      - [ex:name, $(firstname)]
```

5. Press 'Generate LD'

Hands-on: demo



Everyone need's a matey, this is YARR

(Rhymes with tasty, or baby, whatever

See below to start editing YARRRML-c

Or, check our screencasts:

- Matey, with Targets (ISWC 2021
- Matey, the original (ESWC 2018

```
{
    "persons": [
        {
            "firstname": "John",
            "lastname": "Doe"
        },
        {
            "firstname": "Jane",
            "lastname": "Smith"
        },
        {
            "firstname": "Sarah",
            "lastname": "Bladinck"
        }
    ]
}
```

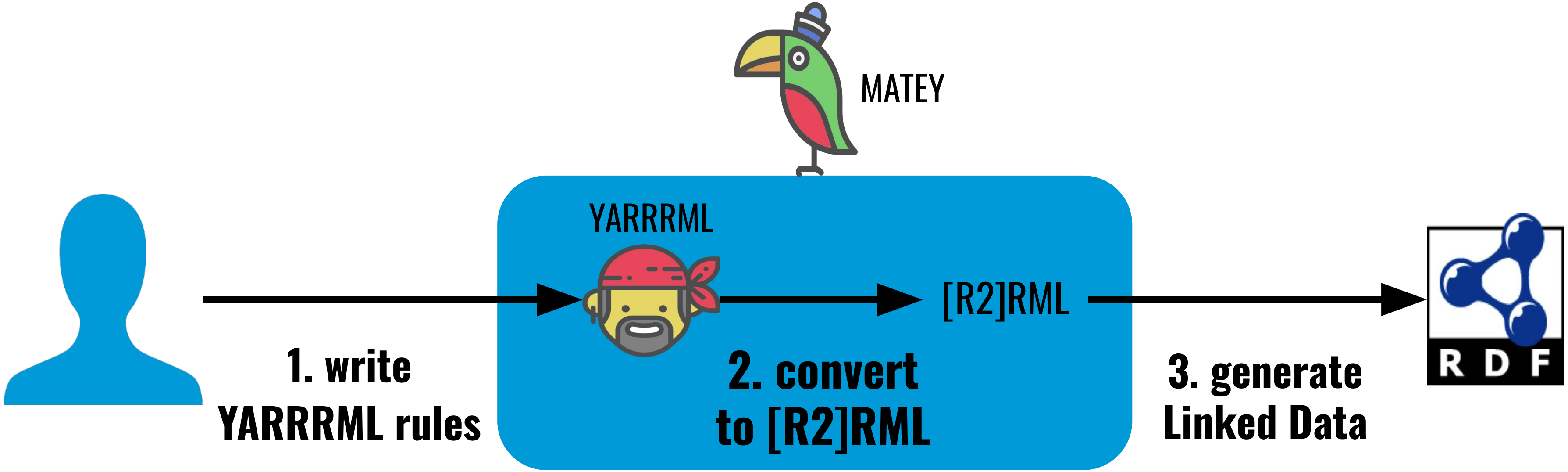
```
prefixes:
 ex: "http://example.com/"

mappings:
  person:
    sources:
      - ['data.json~jsonpath', '$.persons[*]']
    s: http://example.com/$(firstname)
    po:
      - [a, foaf:Person]
      - [ex:name, $(firstname)]
```

```
@prefix rdf: <htt //www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <h   //xmlns.com/foaf/0.1/> .
@prefix ex: <http  example.com/> .

ex:John rdf:type f af:Person ;
    ex:name "John" .

ex:Jane rdf:type foaf:Person ;
    ex:name "Jane" .

ex:Sarah rdf:type foaf:Person ;
    ex:name "Sarah" .

```

6. Knowledge Graph output available here

YARRRML & Matey hands-on!

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



KG4DI

Morph-KGC

David Chaves-Fraga

✉ david.chaves@kuleuven.be

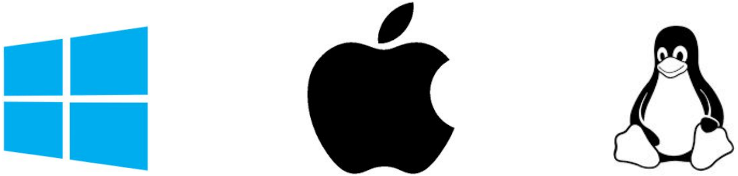
@dchavesf

Main Characteristics

Mapping languages: **R2RML, RML and RML-star**

RDF serializations: **N-TRIPLES, N-QUADS**

Operating Systems:



License: **Apache 2.0**

Supported Data Formats

RELATIONAL DBs

MySQL

PostgreSQL

ORACLE®

Microsoft® SQL Server™

SQLite

TABULAR FILES

CSV

Parquet

Apache orc

HIERARCHICAL FILES

{JSON}

</> XML

Architecture

Implemented in:

Built on top of:

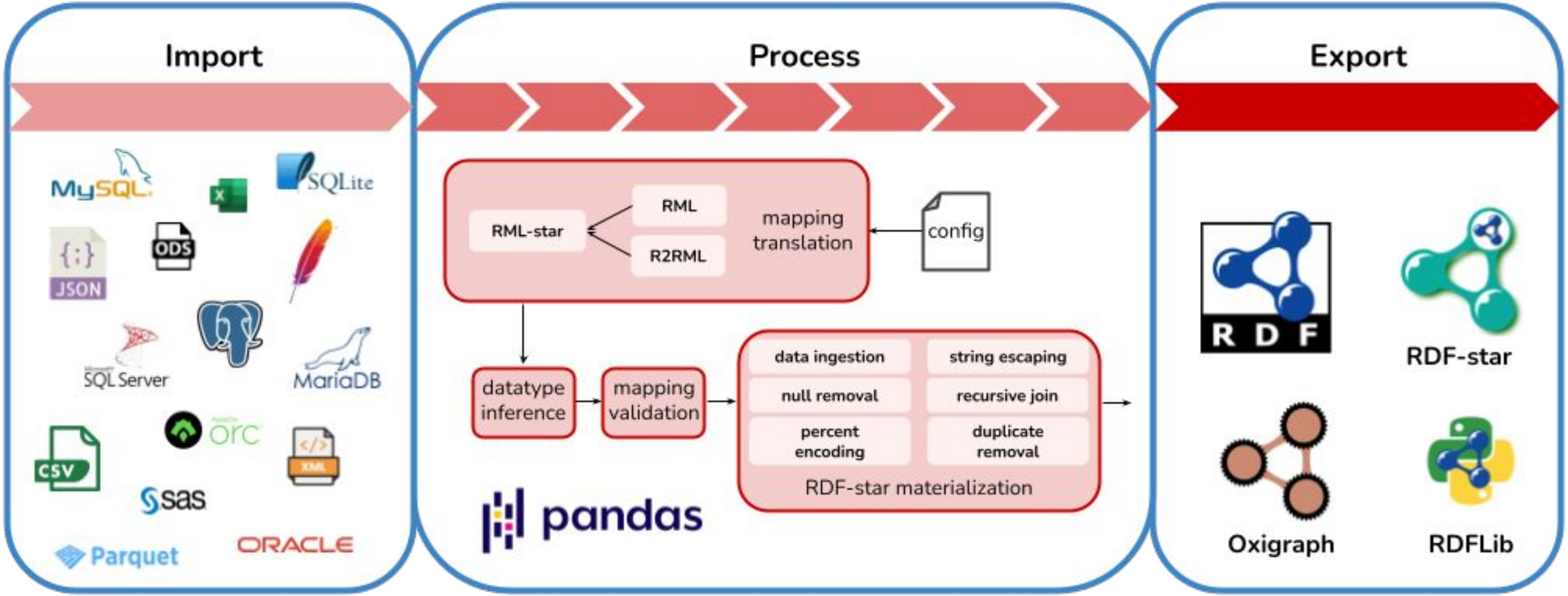
Relational databases access with:

Why Morph-KGC

morph

- Simple and easy: PyPi
- Well-documented (we do not want users to get lost)
- Reliable and robust: continuous integration!
- Fast materialization (very fast!)
- Actively maintained

morph



Import

Process

Export

RML-star

RML

R2RML

mapping translation

config

datatype inference

mapping validation

data ingestion

string escaping

null removal

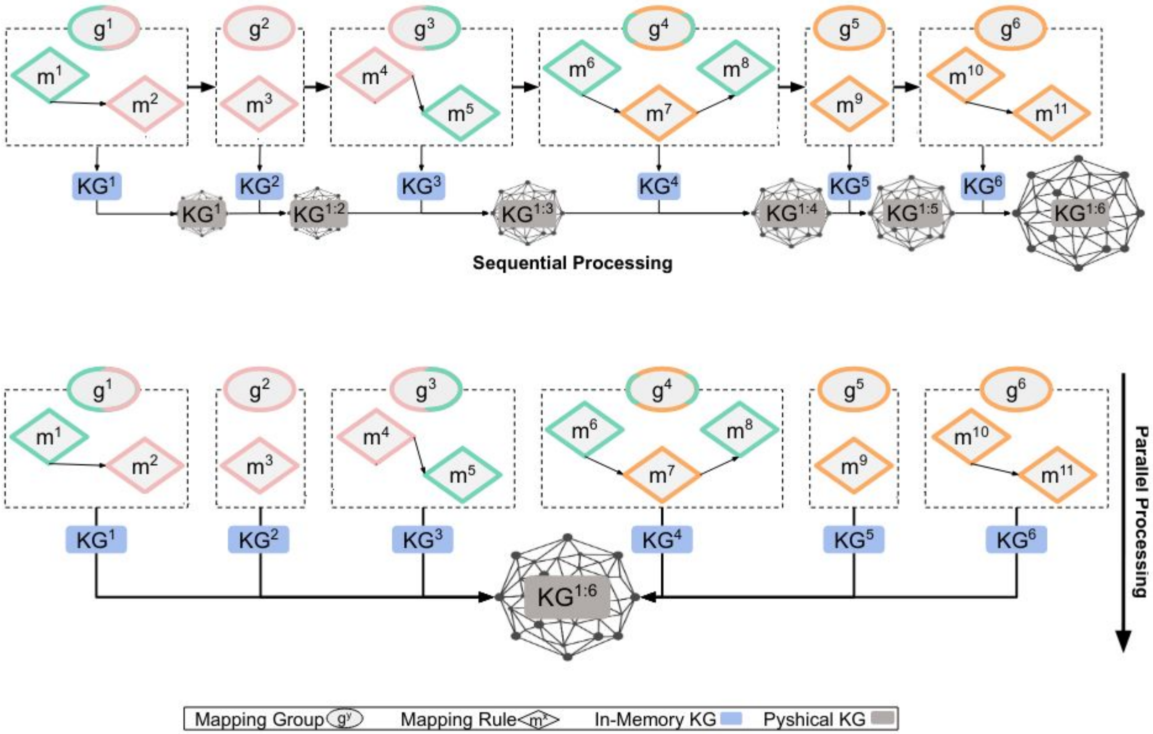
recursive join

percent encoding

duplicate removal

RDF-star materialization

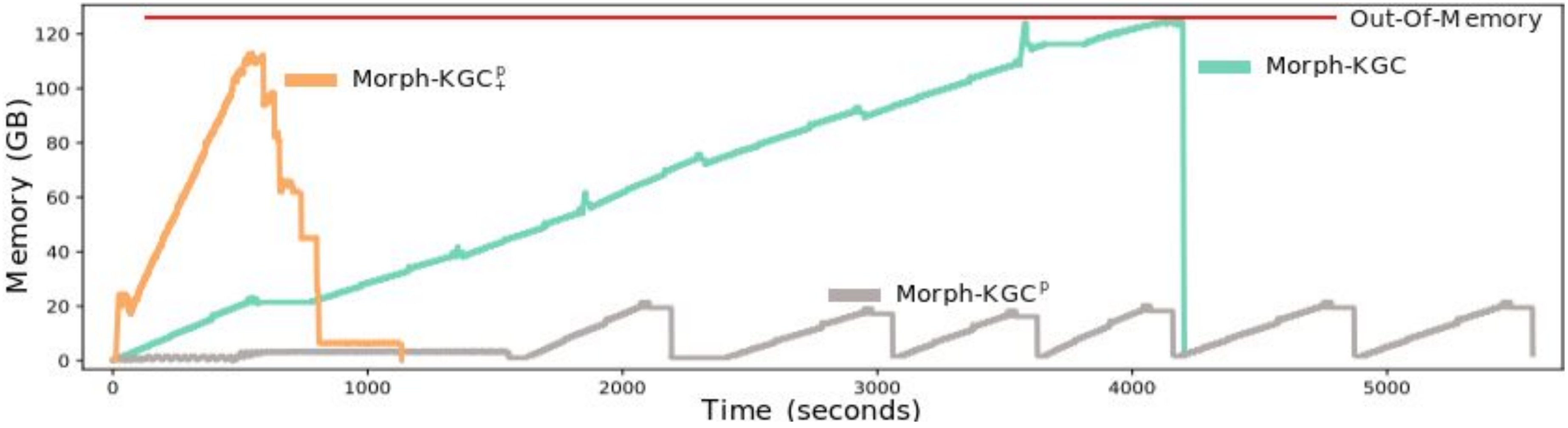
Efficient? How?



Mapping partitioning:

- Sequential processing
- Parallel processing

Memory vs Time (GTFS1000-csv)

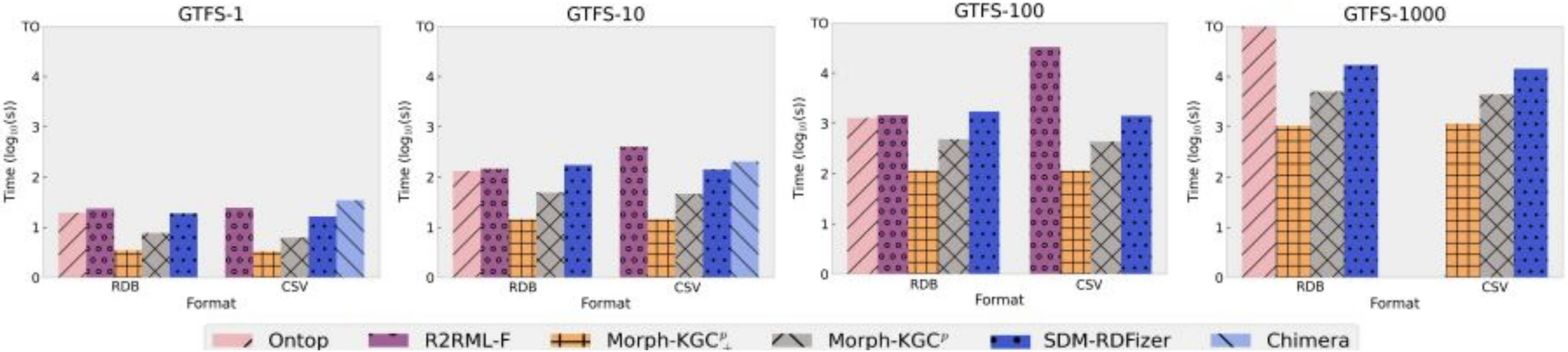


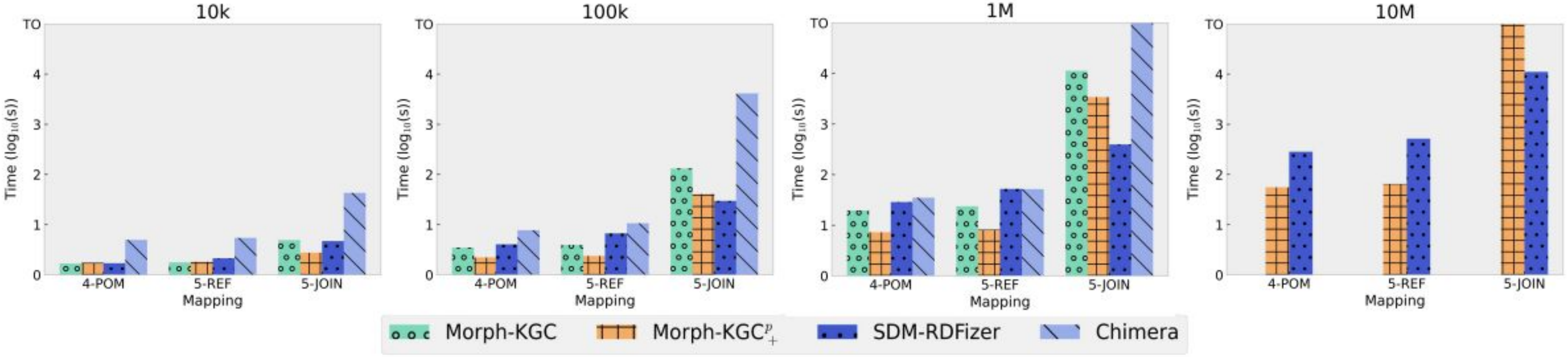
Naive

Sequential

Parallel

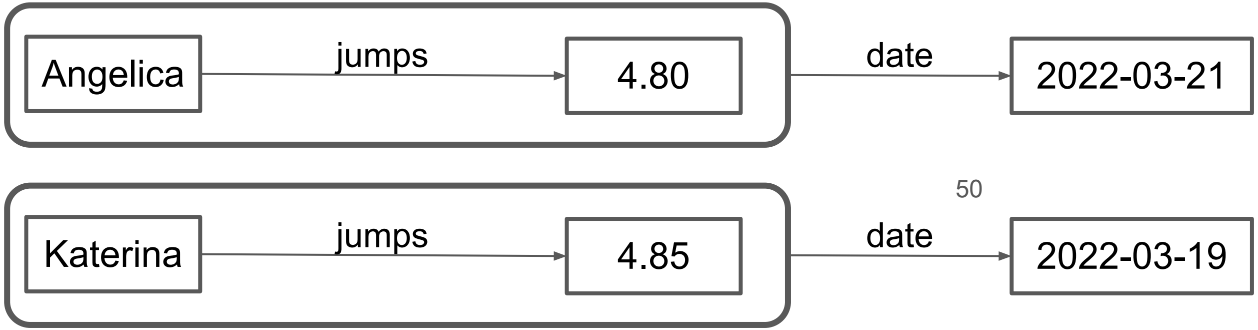
Execution Time (GTFS and COSMIC)





Statements about Statements

ID	DATE	MARK	PERSON
1	2022-03-21	4.80	Angelica
2	2022-03-19	4.85	Katerina

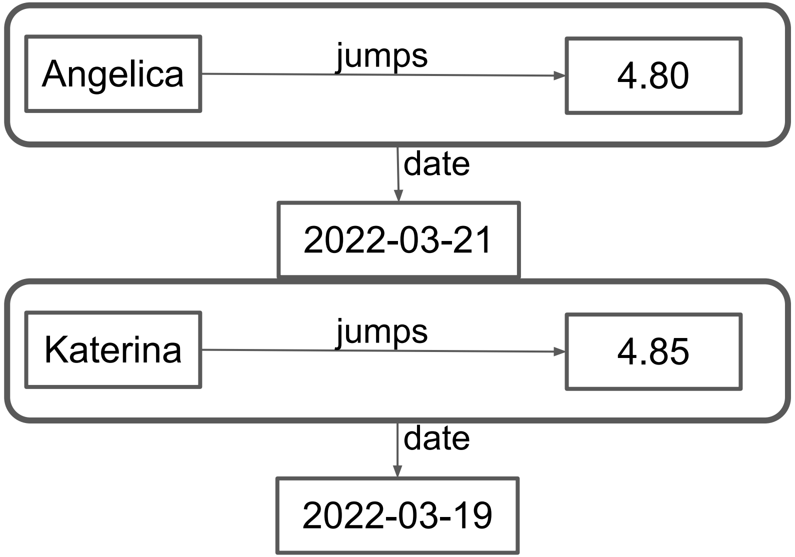


The RDF-star solution

Triples that include a **triple as a subject or an object** are known as RDF-star triples

An RDF-star graph is a **set of RDF-star triples**.

SPARQL-star extends SPARQL to query RDF-star graphs



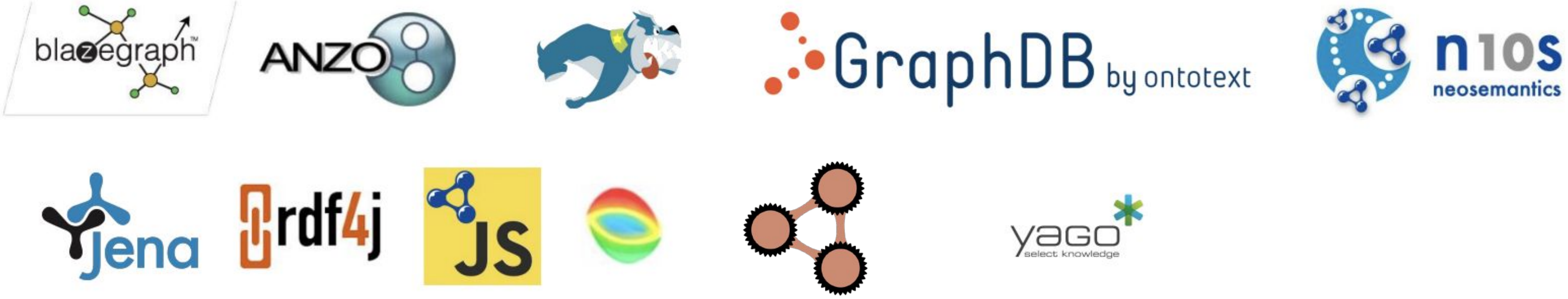
```
<< :Angelica :jumps "4.80" >> :date "2022-03-21"
.
<< :Katerina :jumps "4.85" >> :date "2022-03-19"
.



SELECT ?jumper ?mark ?date WHERE {
        << ?jumper :jumps ?mark >> :date ?date
}
```

RDF-star features

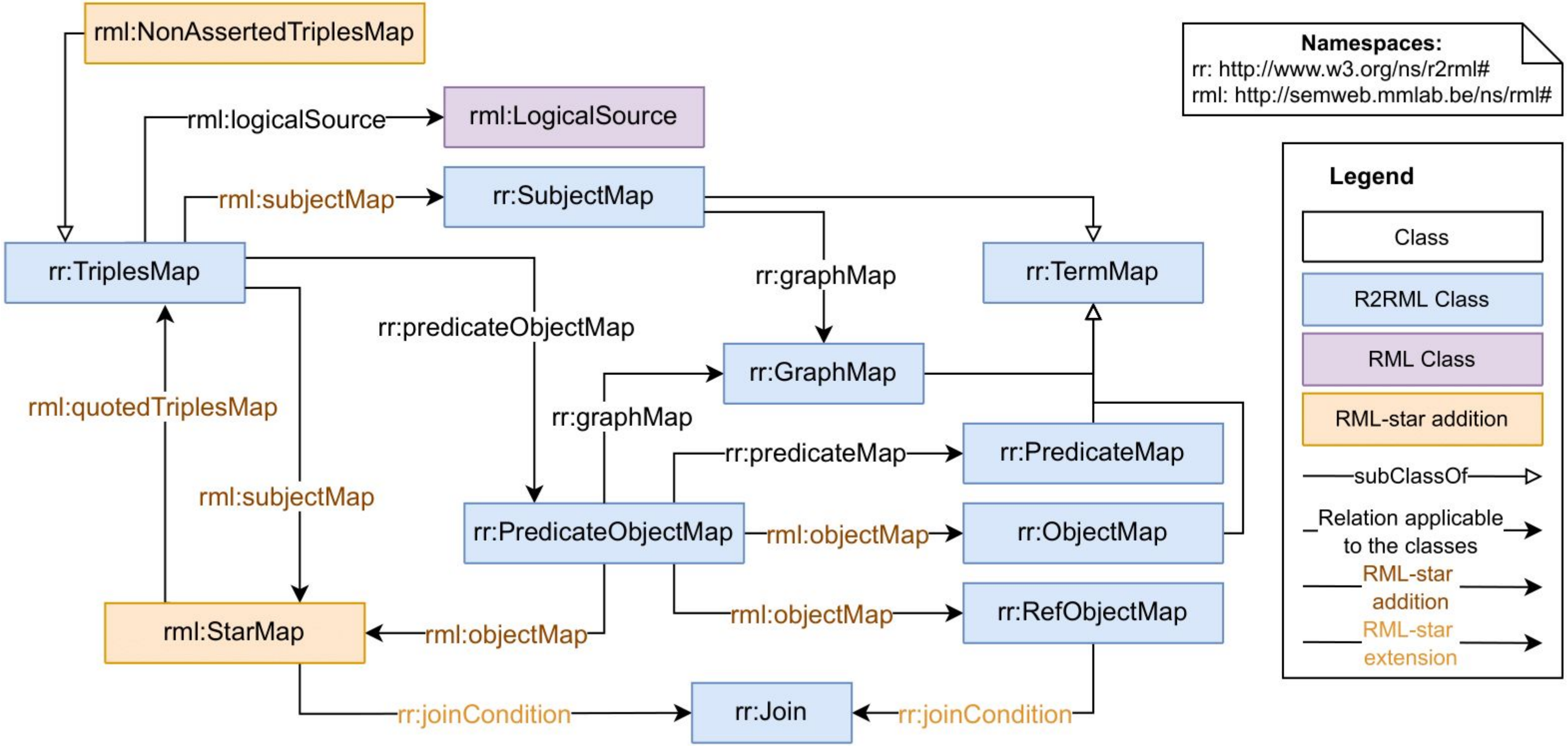
Wide adoption of the approach from industry and vendors

Standardization process through the World Wide Web Consortium (W3C)

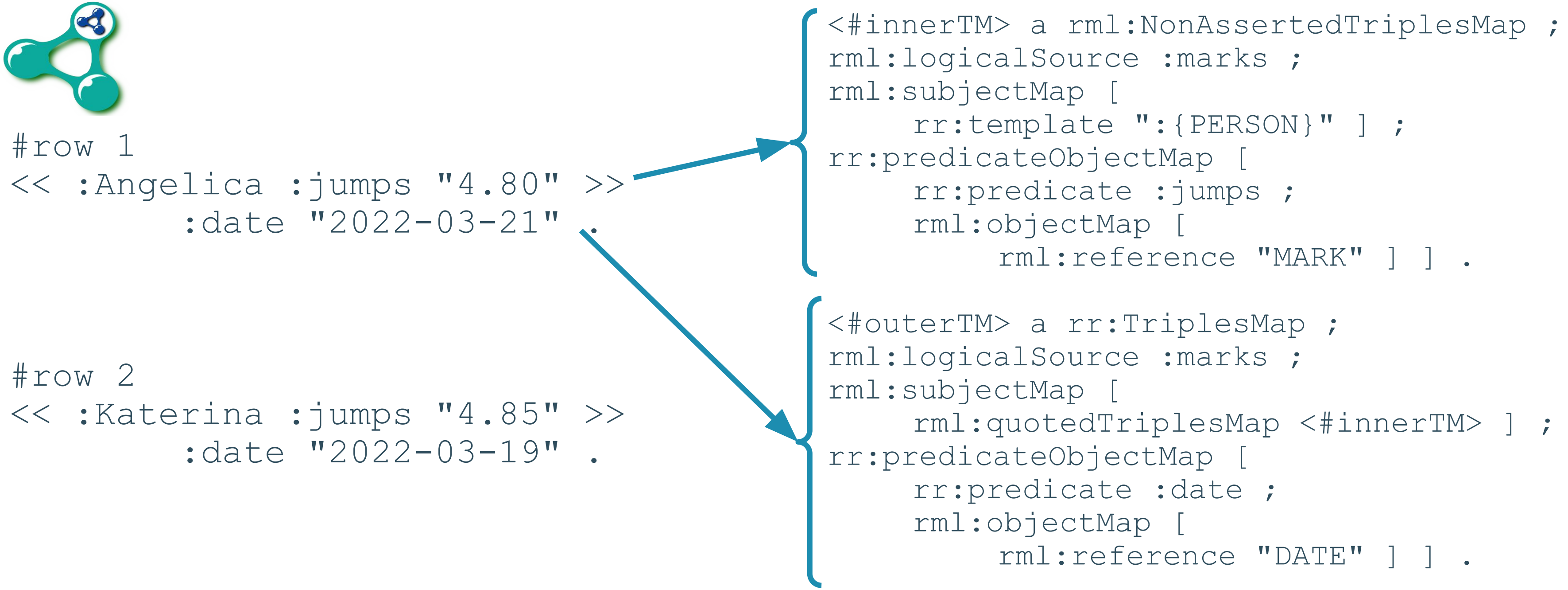
No sustainable procedure to generate RDF-star graphs



RML-star: A declarative generator for RDF-star



RML-star



```
#row 1
<< :Angelica :jumps "4.80" >>
        :date "2022-03-21" .


#row 2
<< :Katerina :jumps "4.85" >>
        :date "2022-03-19" .
```

```
<#innerTM> a rml:NonAssertedTriplesMap ;
rml:logicalSource :marks ;
rml:subjectMap [
    rr:template ":{PERSON}" ] ;
rr:predicateObjectMap [
    rr:predicate :jumps ;
    rml:objectMap [
        rml:reference "MARK" ] ] .
```

```
<#outerTM> a rr:TriplesMap ;
rml:logicalSource :marks ;
rml:subjectMap [
    rml:quotedTriplesMap <#innerTM> ] ;
rr:predicateObjectMap [
    rr:predicate :date ;
    rml:objectMap [
        rml:reference "DATE" ] ] .
```

Delva, T., Arenas-Guerrero, J., Iglesias-Molina, A., Corcho, O., **Chaves-Fraga, D., & Dimou, A.** (2021). RML-star: A declarative mapping language for RDF-star generation. In ISWC2021, the International Semantic Web Conference

Hands-on time!

morph

https://github.com/oeg-upm/morph-kgc/

python Package Index

https://pypi.org/project/morph-kgc/

Tutorial on Colab→ https://short.upm.es/8grm1

KG4DI

Reasoning over KGs

Christophe Debruyne

✉ c.debruyne@uliege.be

@chrdebru

What is a knowledge graph?

A knowledge graph is data stored in a graph that satisfies three conditions:

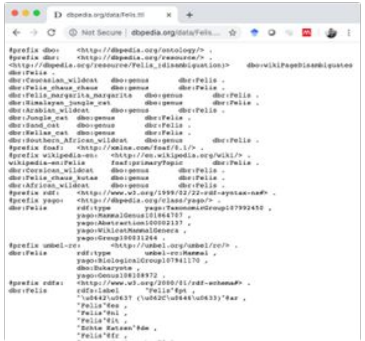
1. Types and relationships are formally described and documented in an ontology (definitions, properties).
2. Integration of information from different domains, organizations, departments, and even from different sources.
3. Support for the inference of implicit relationships, insights, knowledge,... via symbolic AI, statistical AI, or applications.

There are many definitions for the term "knowledge graph," and this presentation does not claim to provide an authoritative one. If one looks at several definitions, one will see that ontologies, integration, and inference are essential recurring themes.

What is an ontology?

An ontology is “a [formal,] explicit specification of a [shared] conceptualization" [Gru95] and extended by [Stu98]

- Explicit → externalized in a document to be shared and used by agents
- Formal → a mathematical or logic foundation to allow reasoning



dbpedia.org/data/Felis.ttl
Not Secure | dbpedia.org/data/Felis...

```
@prefix dbo:    <http://dbpedia.org/ontology/> .
@prefix dbr:    <http://dbpedia.org/resource/> .
<http://dbpedia.org/resource/Felis_(disambiguation)>    dbo:wikiPageDisambiguates
dbr:Felis .
dbr:Caucasian_wildcat   dbo:genus       dbr:Felis .
dbr:Felis_chaus_chaus   dbo:genus       dbr:Felis .
dbr:Felis_margarita_margarita   dbo:genus       dbr:Felis .
dbr:Himalayan_jungle_cat        dbo:genus       dbr:Felis .
dbr:Arabian_wildcat     dbo:genus       dbr:Felis .
dbr:Jungle_cat  dbo:genus       dbr:Felis .
dbr:Sand_cat    dbo:genus       dbr:Felis .
dbr:Kellas_cat  dbo:genus       dbr:Felis .
dbr:Southern_African_wildcat    dbo:genus       dbr:Felis .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix wikipedia-en:   <http://en.wikipedia.org/wiki/> .
wikipedia-en:Felis      foaf:primaryTopic       dbr:Felis .
dbr:Corsican_wildcat    dbo:genus       dbr:Felis .
dbr:Felis_chaus_kutas   dbo:genus       dbr:Felis .
dbr:African_wildcat     dbo:genus       dbr:Felis .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix yago:   <http://dbpedia.org/class/yago/> .
dbr:Felis       rdf:type        yago:TaxonomicGroup107992450 ,
                yago:MammalGenus101864707 ,
                yago:Abstraction100002137 ,
                yago:WikicatMammalGenera ,
                yago:Group100031264 .
@prefix umbel-rc:       <http://umbel.org/umbel/rc/> .
dbr:Felis       rdf:type        umbel-rc:Mammal ,
                yago:BiologicalGroup107941170 ,
                dbo:Eukaryote ,
                yago:Genus108108972 .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
dbr:Felis       rdfs:label      "Felis"@pt ,
                "\u0642\u0637 (\u062C\u0646\u0633)"@ar ,
                "Felis"@es ,
                "Felis"@nl ,
                "Felis"@it ,
                "Echte Katzen"@de ,
                "Felis"@fr ,
```

$$\ni \forall x(Cat(x) \rightarrow Animal(x))$$

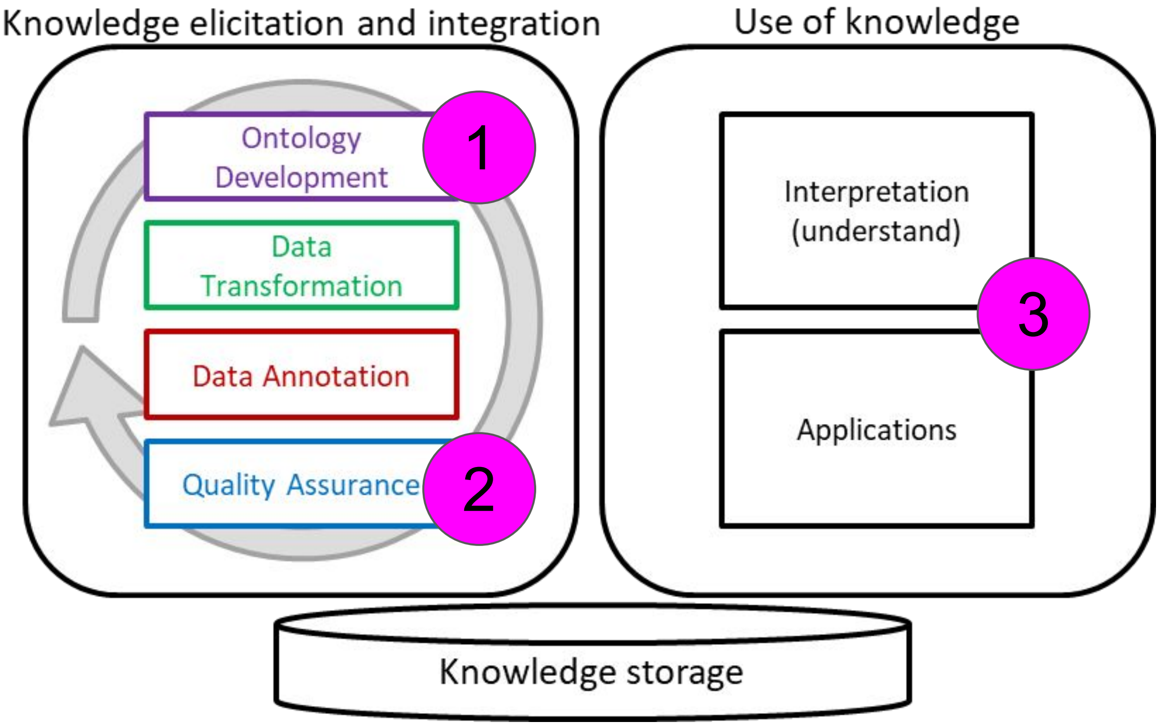
- Shared → for it to be meaningful (e.g., to manage jargon)

Why do we need reasoning?

- Ensure that a knowledge graph is
  - Meaningful - all named classes can have instances
  - Correct - captured intuitions of domain experts
  - Minimally redundant - no unintended synonyms

- Answer queries over ontology classes and instances, e.g.:
  - Find more general and specific classes
  - Retrieve individuals or tuples matching a given query
  - …

Building and maintaining KGs



Reasoning is useful to:

1. Ensure the ontology contains no errors.
2. Ensure the data contains no errors *w.r.t. to the ontology*.
3. Ensure that applications can exploit the semantics.

Figure based on [Den17].

Tutorial

Ontology engineering and in-depth exploration of reasoning are not within the scope of today’s tutorial, but starting from the data you’ve generated with morph-kgc, we will

1. Download the gtfs ontology used by the mapping.
2. Use this ontology with the data to
   a. Demonstrate inference for information retrieval (querying)
   b. Demonstrate satisfiability checking (reasoning)

This tutorial uses RDFLib’s OWL-RL reasoner.

Link: https://colab.research.google.com/drive/1h6n9R3tMXYMN7_kWUoUvSPm6GB4W61BG?usp=sharing

Sources

- [Ant09] Grigoris Antoniou, Frank van Harmelen: Web Ontology Language: OWL. Handbook on Ontologies 2009: 91-110
- [Den17] R. Denaux, Yuan Ren, B. Villazón-Terrazas, P. Alexopoulos, A. Faraotti, H. Wu: Knowledge Architecture for Organisations. Exploiting Linked Data and Knowledge Graphs in Large Organisations 2017: 57-84
- [Gru95] T. Gruber. Toward principles for the design of ontologies used for knowledge sharing? Int. J. Hum.-Comput. Stud., 43(5-6):907–928, 1995.
- [Stu98] R. Studer, R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. Data & Knowledge Engineering, 25(1–2):161–198, 1998.

KG4DI

Tutorial on Knowledge Graph Construction

David Chaves-Fraga (KULeuven), Dylan Van Assche (Ghent U.),
Christophe Debruyne (ULiège)

✉ public-kg-construct@w3.org

@kgc_workshop