RML-view-to-CSV: A Proof-of-Concept Implementation for RML Logical Views

Els de Vleeschauwer[1], Pano Maria[2], Ben De Meester[1], and Pieter Colpaert[1]

[1]Ghent University – imec – IDLab, Belgium, [2]Skemu
els.devleeschauwer@ugent.be

Content

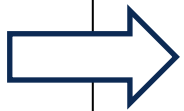
RML Mapping Language (RML)
RML Logical Views
RML-view-to-CSV
Evaluation
Conclusion

Content

RML is a language to express mapping rules from heterogeneous data to RDF

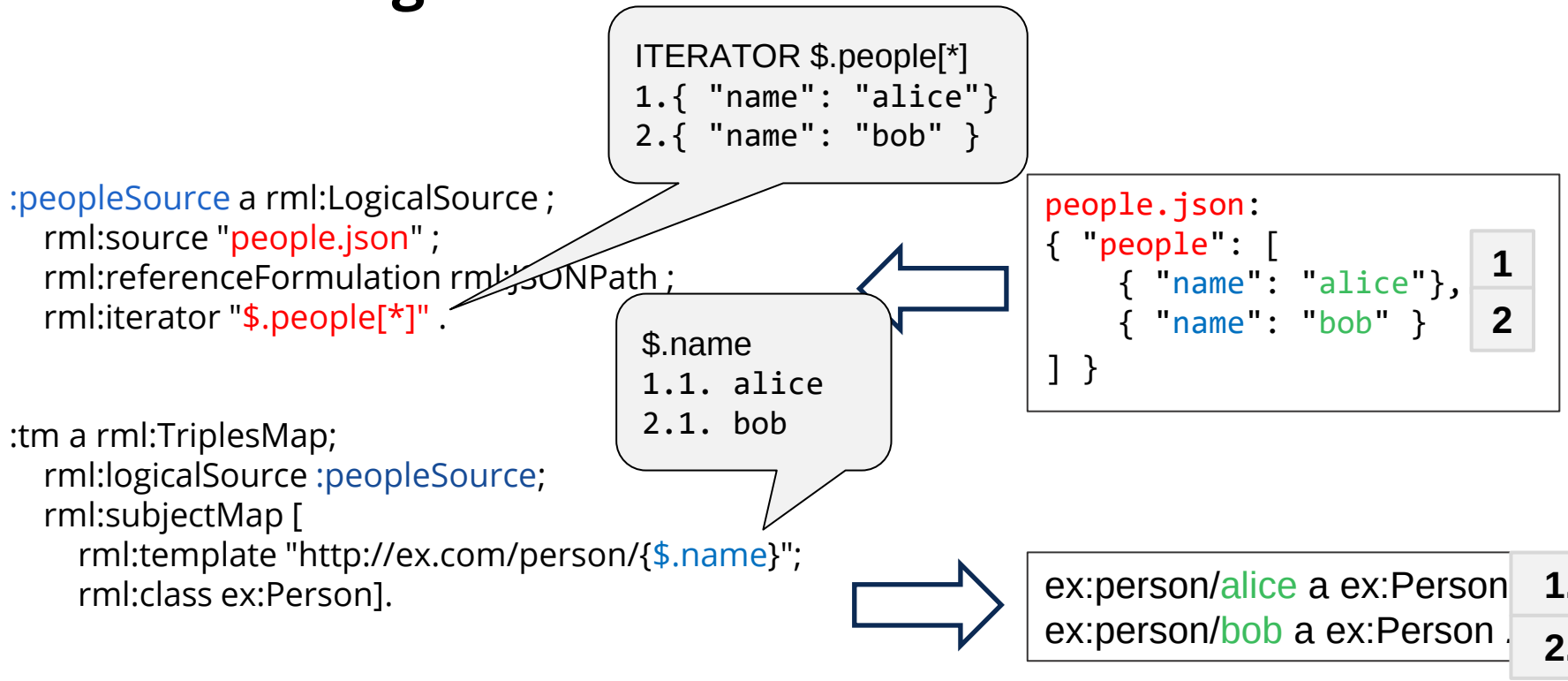
```
:peopleSource a rml:LogicalSource ;
   rml:source "people.json" ;
   rml:referenceFormulation rml:JSONPath ;
   rml:iterator "$.people[*]" .


:tm a rml:TriplesMap;
   rml:logicalSource :peopleSource;
   rml:subjectMap [
      rml:template "http://ex.com/person/{$.name}";
      rml:class ex:Person].
```

```
people.json:
{ "people": [
    { "name": "alice"},
    { "name": "bob" }
] }
```



```
ex:person/alice a ex:Person .
ex:person/bob a ex:Person .
```

RML is a language to expresses mapping rules from heterogeneous data to RDF



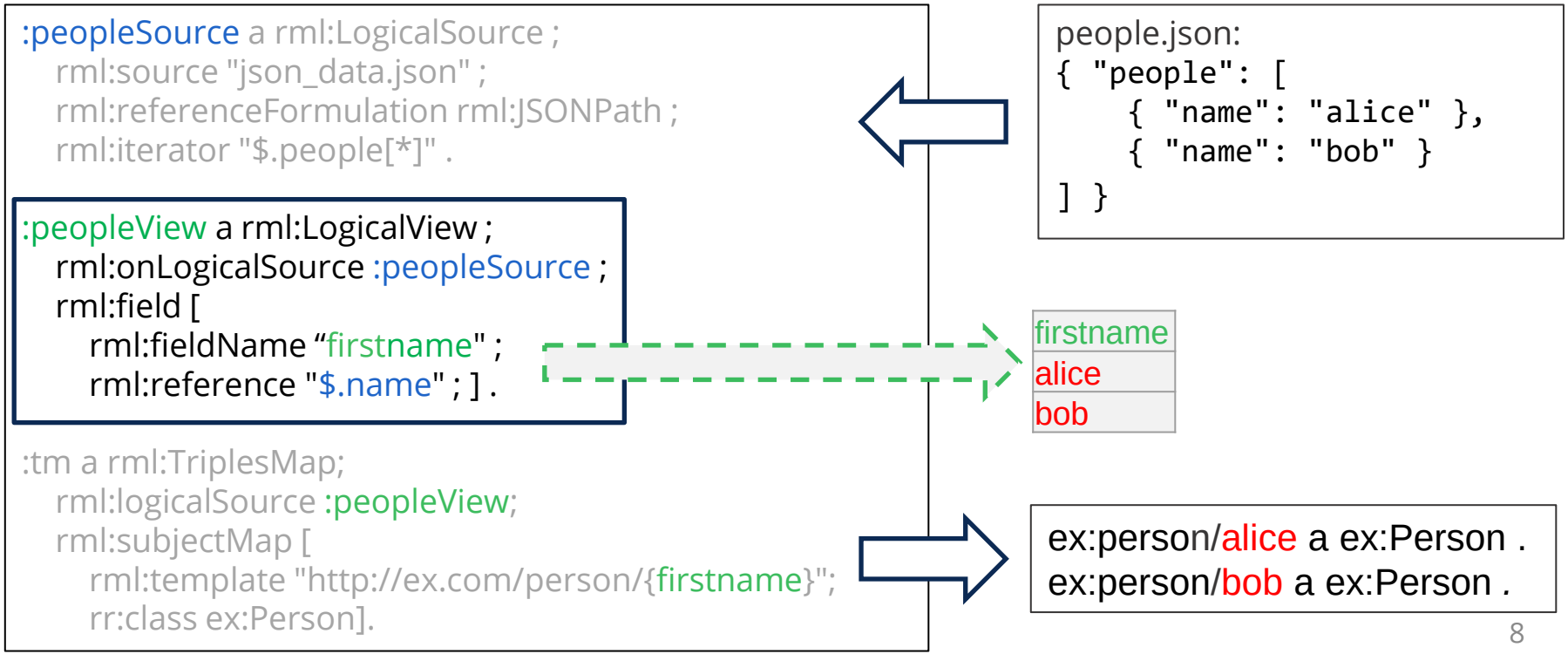
RML Ontology Modules

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.

Content

RML Logical Views are virtual views on top of logical sources



RML Logical Views offer solutions for open issues in RML

Inability to handle hierarchy in nested data
Inability to handle mixed data formats
Limited join functionality

⇨ RML-LV module under development
https://github.com/kg-construct/rml-lv

Issue 1: Inability to handle hierarchy in nested data

```
{ "people": [
      { "name": "alice",
        "items": [
              { "type": "sword",  "weight": 1500},
              { "type": "shield", "weight": 2500} ] } ,
      { "name": "bob",
        "items":[
              { "type": "flower",  "weight": 15 } ] } ] }
```

ex:person/alice/sword ex:hasWeight 1500.
ex:person/alice/shield ex:hasWeight 2500.
ex:person/bob/flower ex:hasItem 15.

Issue 1: Inability to handle hierarchy in nested data

```
{ "people": [
      { "name": "alice",
        "items": [
              { "type": "sword",  "weight": 1500},
              { "type": "shield", "weight": 2500} ] } ,
      { "name": "bob",
        "items":[
              { "type": "flower",  "weight": 15 } ] } ] }
```

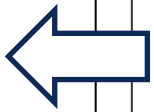
ITERATOR $.people[*]
1.{"name": "alice", "items": [...]}
2.{"name": "bob", "items":[...]}

$.name,$.items[*].type, $items[*].weight
1.1. alice, sword, 1500
2. alice, sword, 2500
3. alice, shield, 1500
4. alice, shield, 2500
2.1. bob, flower, 15

ex:person/alice/sword ex:hasWeight 1500.
ex:person/alice/shield ex:hasWeight 2500.
ex:person/bob/flower ex:hasItem 15.

Solution 1: Flattening of nested data structures

```
:peopleView a rml:LogicalView ;
   rml:onLogicalSource :peopleSource ;
   rml:field [
      rml:fieldName “name" ;
      rml:reference "$.name" ; ] ;
   rml:field [
      rml:fieldName "item" ;
      rml:reference "$.items[*]" ;
      rml:field [
         rml:fieldName "type" ;
         rml:reference "$.type" ; ] ;
      rml:field [
         rml:fieldName "weight" ;
         rml:reference "$.weight" ; ] ; ] .
```



people.json

```
{ "people": [
    { "name": "alice",
      "items": [
          { "type": "sword",
            "weight": 1500 },
          { "type": "shield",
            "weight": 2500} ] },
    { "name":"bob",
      "items":[
          { "type":"flower",
            "weight":15 } ] } ] }
```

name	item	item.type	item.weight
alice	{...}	sword	1500
alice	{...}	shield	2500
bob	{...}	flower	15

Solution 1: Flattening of nested data structures

```
:peopleView a rml:LogicalView ;
   rml:onLogicalSource :peopleSource ;
   rml:field [
      rml:fieldName “name" ;
      rml:reference "$.name" ; ] ;
   rml:field [
      rml:fieldName "item" ;
      rml:reference "$.items[*]" ;
      rml:field [
         rml:fieldName "type" ;
         rml:reference "$.type" ; ] ;
      rml:field [
         rml:fieldName "weight" ;
         rml:reference "$.weight" ; ] ; ] .
```

Nested fields

people.json

```
{ "people": [
    { "name": "alice",
      "items": [
          { "type": "sword",
            "weight": 1500 },
          { "type": "shield",
            "weight": 2500} ] },
    { "name":"bob",
      "items":[
          { "type
            "weig
```

Dot notation

name	item	item.type	item.weight
alice	{...}	sword	1500
alice	{...}	shield	2500
bob	{...}	flower	15

Issue 2: Inability to handle mixed data formats

```
name,  item
alice, "{""type"":""sword"",""weight"": 2500}"
alice, "{""type"":""shield"",""weight"": 1500}"
bob,   "{""type"":""flower"",""weight"": 15 }"
```

ex:person/alice ex:hasItem "sword", "shield".
ex:person/bob ex:hasItem "flower".

Solution 2: Handling of mixed data formats

```
:peopleView2 a rml:LogicalView ;
   rml:onLogicalSource [
      rml:source "./people.csv" ;
      rml:referenceFormulation rml:CSV ] ;
   rml:field [
      rml:fieldName "item" ;
      rml:reference "$.items.[*]" ;
      rml:referenceFormulation rml:JSONPath ;
      rml:field [
         rml:fieldName "type" ;
         rml:reference "$.type" ; ] ;
      rml:field [
         rml:fieldName "weight" ;
         rml:reference "$.weight" ; ] ; ] .
```

people.csv

```
name,  item
alice, "{""type"":""sword"", …}"
alice, "{""type"":""shield"", …}"
bob,   "{""type"":""flower"", …}"
```

name	item	item.type	item.weight
alice	{...}	sword	1500
alice	{…}	shield	2500
bob	{…}	flower	15

Solution 2: Handling of mixed data formats

```
:peopleView2 a rml:LogicalView ;
   rml:onLogicalSource [
      rml:source "./people.csv" ;
      rml:referenceFormulation rml:CSV ] ;
   rml:field [
      rml:fieldName "item" ;
      rml:reference "$.items.[*]" ;
      rml:referenceFormulation rml:JSONPath ;
      rml:iterator "$.*"
      rml:field [
         rml:fieldName "type" ;
         rml:reference "$.type" ; ] ;
      rml:field [
         rml:fieldName "weight" ;
         rml:reference "$.weight" ; ] ; ] .
```

people.csv

```
name,  item
alice, "{""type"":""sword"", …}"
alice, "{""type"":""shield"", …}"
bob,   "{""type"":""flower"", …}"
```

Optional reference formulation per field

name	item	item.type	item.weight
alice	{...}	sword	1500
alice	{…}	shield	2500
bob	{…}	flower	15

Issue 3: Limited join functionality

```
name,  id
alice, 1
bob,   2
tobias,3
```

```
name,  item_type
alice, sword
alice, shield
bob,   flower
```

```
ex:person/1 ex:hasItem "sword", "shield".
ex:person/2 ex:hasItem "flower".
```

Solution 3: Extended joining of data sources

```
:idView a rml:LogicalView ;
   rml:onLogicalSource :idSource ;
   rml:field [
      rml:fieldName "name" ;
      rml:reference "name" ; ] ;
   rml:field [
      rml:fieldName "id" ;
      rml:reference "id" ; ] ;
   rml:leftJoin [
      rml:parentLogicalView :peopleView ;
      rml:joinCondition [
         rml:parent "name" ;
         rml:child "name" ; ] ;
      rml:field [
         rml:fieldName "item_type" ;
         rml:reference "item.type" ; ] ;
      rml:field [
         rml:fieldName "item_weight" ;
         rml:reference "item.weight" ; ] ; ] .
```

id.csv

```
name,  id
alice, 1
bob,   2
tobias,3
```

name	item	item.type	item.weight
alice	{...}	sword	1500
alice	{…}	shield	2500
bob	{…}	flower	15

name	id	item_type	item_weight
alice	1	sword	1500
alice	1	shield	2500
bob	2	flower	15
tobias	3		

Solution 3: Extended joining of data sources

```
:idView a rml:LogicalView ;
   rml:onLogicalSource :idSource ;
   rml:field [
      rml:fieldName "name" ;
      rml:reference "name" ; ] ;
   rml:field [
      rml:fieldName "id" ;
      rml:reference "id" ; ] ;
   rml:leftJoin [
      rml:parentLogicalView :peopleView ;
      rml:joinCondition [
         rml:parent "name" ;
         rml:child "name" ; ] ;
      rml:field [
         rml:fieldName "item_type" ;
         rml:reference "item.type" ; ]
      rml:field [
         rml:fieldName "item_weight" ;
         rml:reference "item.weight" ; ] ; ] .
```

Join with another logical view

Compare fields in join condition

Add and rename fields from the join

id.csv

```
name,  id
alice, 1
bob,   2
tobias,3
```

name	item	item.type	item.weight
alice	{...}	sword	1500
		shield	2500
		flower	15

name	id	item_type	item_weight
ice	1	sword	1500
ice	1	shield	2500
ob	2	flower	15
tobias	3		

Content

RML Mapping Language (RML)

RML Logical Views

RML-view-to-CSV

Evaluation

Conclusion

RML-view-to-CSV implements RML Logical Views

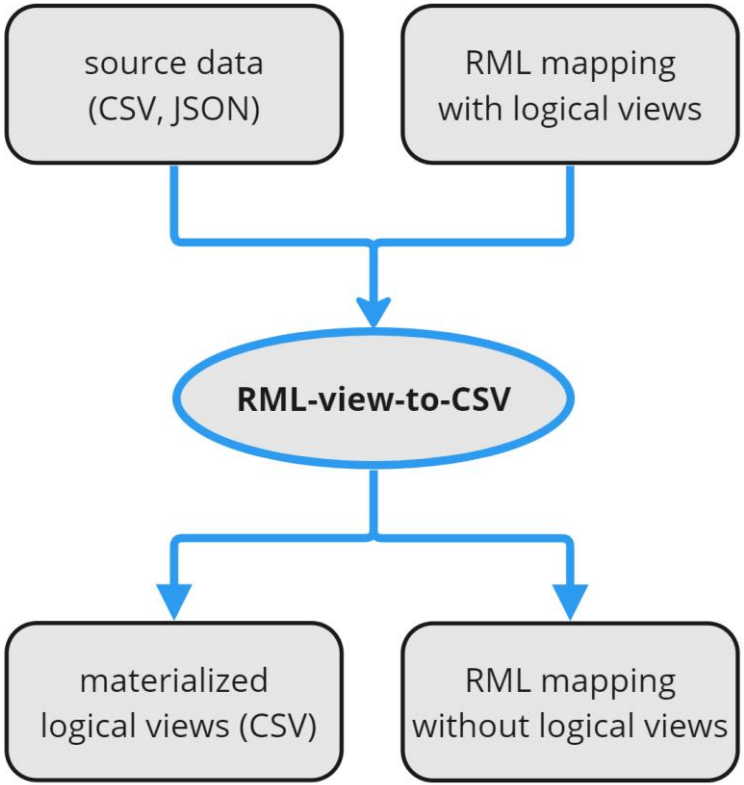
Proof-of-Concept:

- one tabular source format (CSV)
- one nested source format (JSON)

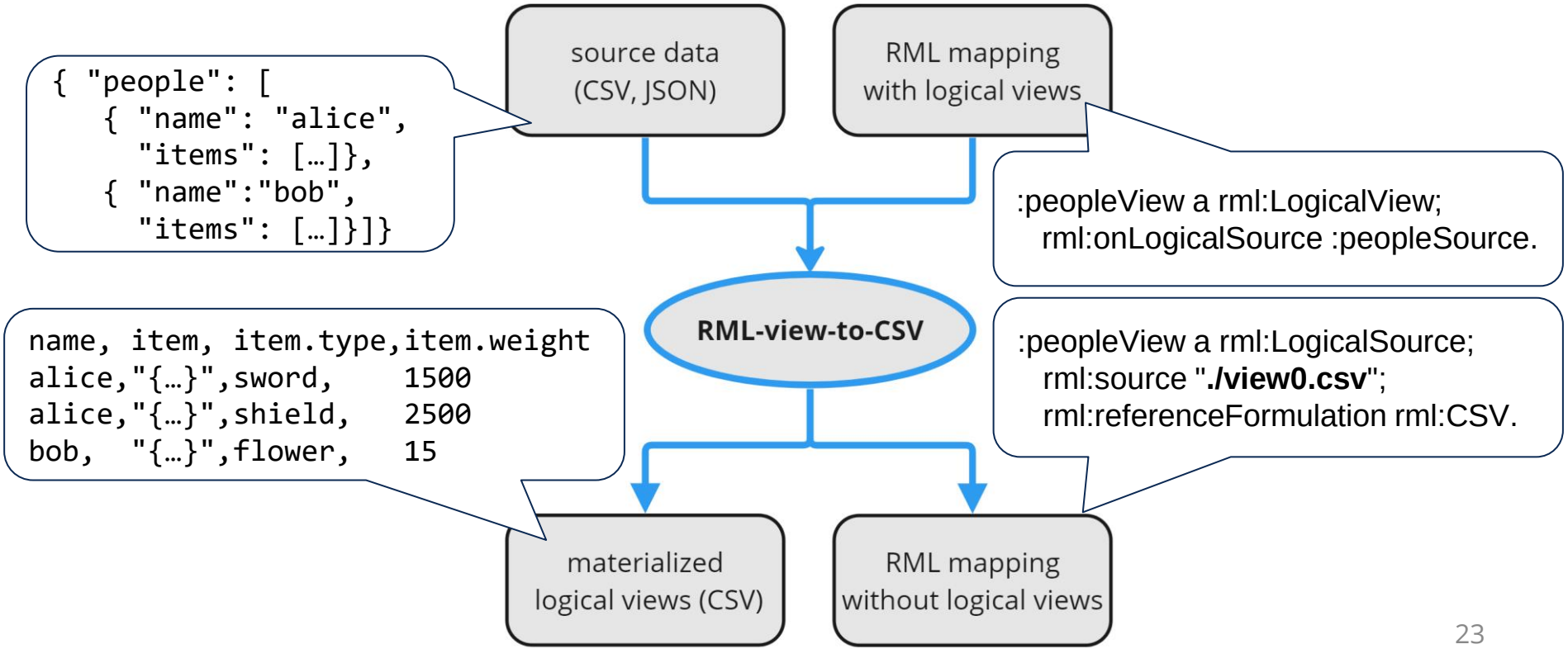
Built on top of Python pandas

https://github.com/RMLio/rml-view-to-csv

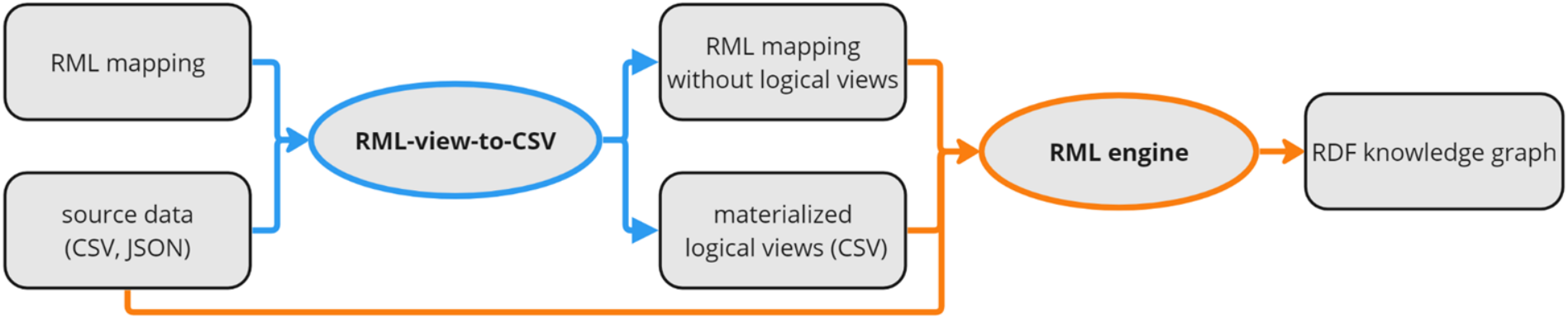
RML-view-to-CSV materializes RML Logical Views and rewrites the mapping



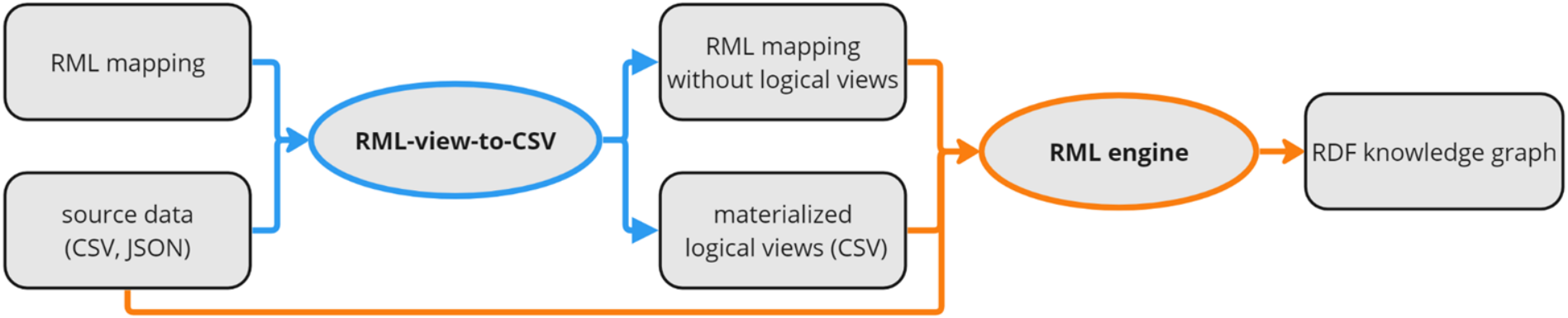
RML-view-to-CSV materializes RML Logical Views and rewrites the mapping



RML mapping engines can use RML-view-to-CSV as preprocessor

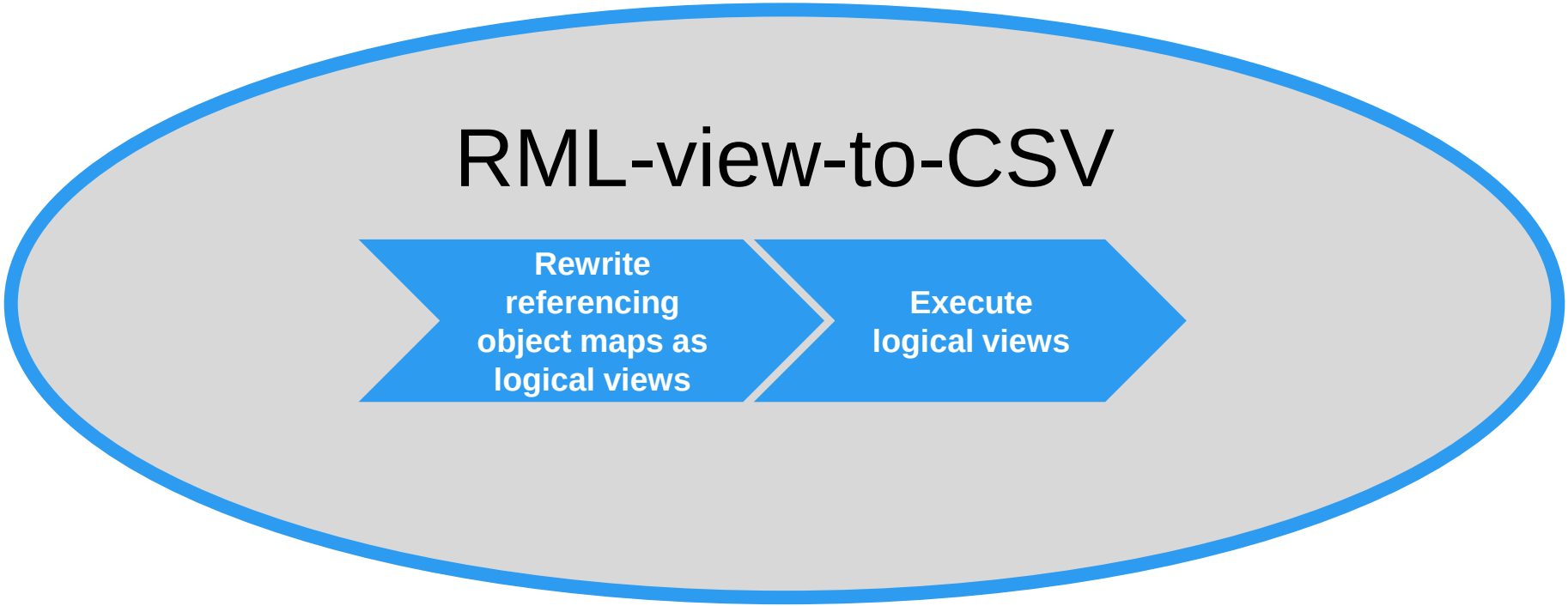


RML mapping engines can use RML-view-to-CSV as preprocessor

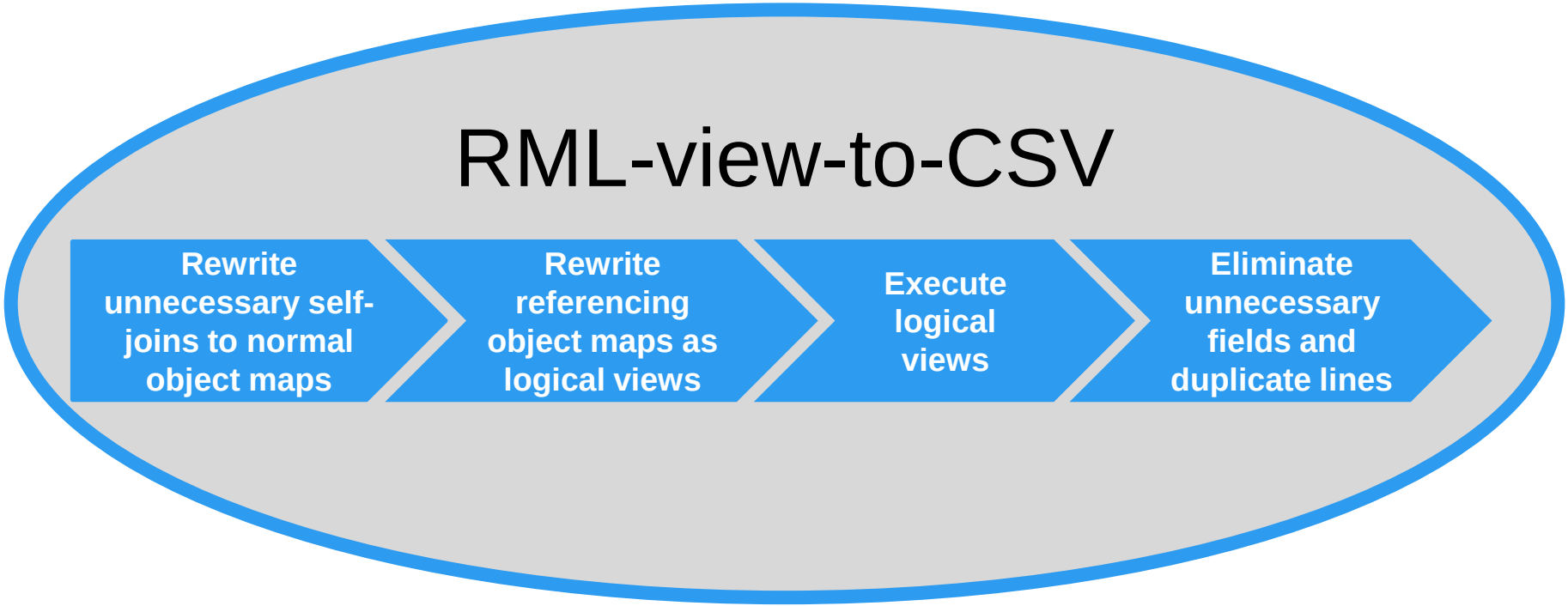


With this pipeline we **executed** and **corrected the test cases** in the RML Logical Views module

Extension to all joins



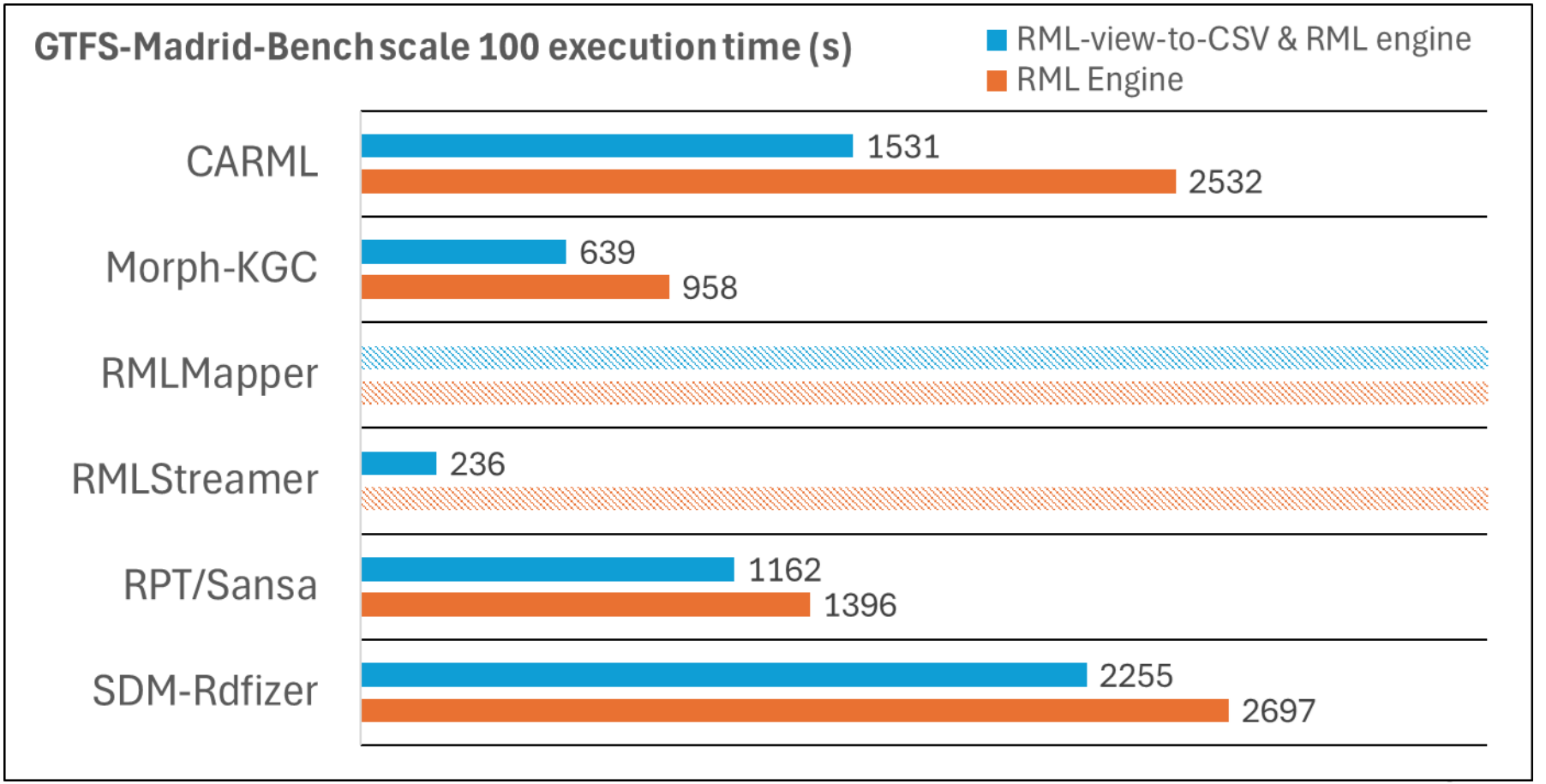
Needed optimizations encountered when running the benchmarks



Content

RML Mapping Language (RML)
RML Logical Views
RML-view-to-CSV
Evaluation
Conclusion

Faster knowledge graph construction



Content

RML Mapping Language (RML)
RML Logical Views
RML-view-to-CSV
Evaluation
Conclusion

Conclusion

The **RML Logical Views spec** is still **under development**.

RML-view-to-CSV validates and improves the RML Logical Views spec.

The evaluation shows **performance gains** and the **potential of a modular approach.**

Future

Continue to support the RML Logical Views development implementing **new features** (e.g. indexes, aggregations), and validating **formal definitions.**

RML-view-to-CSV: A Proof-of-Concept Implementation for RML Logical Views

https://raw.githubusercontent.com/elsdvlee/papers/main/2024/RML-view-to-CSV_A_Proof-of-Concept_Implementation_for_RML_Logical_Views.pdf

Els de Vleeschauwer[1], Pano Maria[2], Ben De Meester[1], and Pieter Colpaert[1]

[1]Ghent University – imec – IDLab, Belgium, [2]Skemu

els.devleeschauwer@ugent.be