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

Els de Vleeschauwer¹, Pano Maria², Ben De Meester¹, and Pieter Colpaert¹

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

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

: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"
    }
  ]
}

$.name
1.1. alice
2.1. bob

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

ex:person/alice a ex:Person .
ex:person/bob a ex:Person .
# RML Ontology Modules

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

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

RML Mapping Language (RML)
RML Logical Views
RML-view-to-CSV
Evaluation
Conclusion
RML Logical Views are virtual views on top of logical sources.

people.json:

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

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

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

```
<table>
<thead>
<tr>
<th>Ex: Person/ Alice/ Sword Ex: Has Weight 1500.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex: Person/ Alice/ Shield Ex: Has Weight 2500.</td>
</tr>
<tr>
<td>Ex: Person/ Bob/ Flower Ex: Has Item 15.</td>
</tr>
</tbody>
</table>
```
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 }
      ] }
  ] }

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 } ] } ]
}
```

<table>
<thead>
<tr>
<th>name</th>
<th>item</th>
<th>item.type</th>
<th>item.weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>{...}</td>
<td>sword</td>
<td>1500</td>
</tr>
<tr>
<td>alice</td>
<td>{...}</td>
<td>shield</td>
<td>2500</td>
</tr>
<tr>
<td>bob</td>
<td>{...}</td>
<td>flower</td>
<td>15</td>
</tr>
</tbody>
</table>
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`

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

<table>
<thead>
<tr>
<th>name</th>
<th>item</th>
<th>item.type</th>
<th>item.weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>{...}</td>
<td>sword</td>
<td>1500</td>
</tr>
<tr>
<td>alice</td>
<td>{...}</td>
<td>shield</td>
<td>2500</td>
</tr>
<tr>
<td>bob</td>
<td>{...}</td>
<td>flower</td>
<td>15</td>
</tr>
</tbody>
</table>

**Nested fields**

**Dot notation**
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

```rml
t :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**

<table>
<thead>
<tr>
<th>name</th>
<th>item</th>
<th>item.type</th>
<th>item.weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td><code>{&quot;type&quot;:&quot;sword&quot;, ...}</code></td>
<td>sword</td>
<td>1500</td>
</tr>
<tr>
<td>alice</td>
<td><code>{&quot;type&quot;:&quot;shield&quot;, ...}</code></td>
<td>shield</td>
<td>2500</td>
</tr>
<tr>
<td>bob</td>
<td><code>{&quot;type&quot;:&quot;flower&quot;, ...}</code></td>
<td>flower</td>
<td>15</td>
</tr>
</tbody>
</table>
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 "weight" ;

Optional reference formulation per field

<table>
<thead>
<tr>
<th>name</th>
<th>item</th>
<th>item.type</th>
<th>item.weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>{...}</td>
<td>sword</td>
<td>1500</td>
</tr>
<tr>
<td>alice</td>
<td>{...}</td>
<td>shield</td>
<td>2500</td>
</tr>
<tr>
<td>bob</td>
<td>{...}</td>
<td>flower</td>
<td>15</td>
</tr>
</tbody>
</table>
Issue 3: Limited join functionality

<table>
<thead>
<tr>
<th>name, id</th>
<th>name, item_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice, 1</td>
<td>sword</td>
</tr>
<tr>
<td>bob, 2</td>
<td>shield</td>
</tr>
<tr>
<td>tobias, 3</td>
<td>flower</td>
</tr>
</tbody>
</table>

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" ; ] ; ] .
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>id</td>
<td>item</td>
<td>item.weight</td>
</tr>
<tr>
<td>alice</td>
<td>1</td>
<td>sword</td>
<td>1500</td>
</tr>
<tr>
<td>alice</td>
<td>1</td>
<td>shield</td>
<td>2500</td>
</tr>
<tr>
<td>bob</td>
<td>2</td>
<td>flower</td>
<td>15</td>
</tr>
</tbody>
</table>
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>id</td>
<td>item</td>
<td>item.type</td>
</tr>
<tr>
<td>alice</td>
<td>1</td>
<td>sword</td>
<td></td>
</tr>
<tr>
<td>alice</td>
<td>1</td>
<td>shield</td>
<td></td>
</tr>
<tr>
<td>bob</td>
<td>2</td>
<td>flower</td>
<td></td>
</tr>
<tr>
<td>tobias</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
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

<table>
<thead>
<tr>
<th>name</th>
<th>item</th>
<th>item_type</th>
<th>item_weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>{...}</td>
<td>sword</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td></td>
<td>shield</td>
<td>2500</td>
</tr>
<tr>
<td>bob</td>
<td></td>
<td>flower</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>item_type</th>
<th>item_weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>1</td>
<td>sword</td>
<td>1500</td>
</tr>
<tr>
<td>alice</td>
<td>1</td>
<td>shield</td>
<td>2500</td>
</tr>
<tr>
<td>bob</td>
<td>2</td>
<td>flower</td>
<td>15</td>
</tr>
<tr>
<td>tobias</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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

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

RML-view-to-Csv

Rewrite referencing object maps as logical views

Execute logical views
Needed optimizations encountered when running the benchmarks

RML-view-to-CSV

- Rewrite unnecessary self-joins to normal object maps
- Rewrite referencing object maps as logical views
- Execute logical views
- Eliminate unnecessary fields and duplicate lines
Content

RML Mapping Language (RML)
RML Logical Views
RML-view-to-CSV
Evaluation
Conclusion
## Faster knowledge graph construction

<table>
<thead>
<tr>
<th>Name</th>
<th>RML-view-to-CSV &amp; RML engine</th>
<th>RML Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARML</td>
<td>1531</td>
<td>2532</td>
</tr>
<tr>
<td>Morph-KGC</td>
<td>639</td>
<td>958</td>
</tr>
<tr>
<td>RMLMapper</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMLStreamer</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>RPT/Sansa</td>
<td>1162</td>
<td>1396</td>
</tr>
<tr>
<td>SDM-Rdfizer</td>
<td>2255</td>
<td>2697</td>
</tr>
</tbody>
</table>
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


Els de Vleeschauwer¹, Pano Maria², Ben De Meester¹, and Pieter Colpaert¹

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