Guidance, Please!

Towards a Framework for RDF-based Constraint Languages.

Thomas Bosch, GESIS – Lebniz Institute for the Social Sciences
Kai Eckert, Stuttgart Media University
Prerequisites

• For RDF data, we want to express constraints, as it is common for XML files using for example XML Schema: „Every Book must have at least one author.“

• Several languages exist or are in development that support this, for instance Description Set Profiles, OWL 2 (despite not intended to be a constraint language), Shape Expressions or Resource Shapes.

• For the further development of constraint languages, the DCMI RDF Application Profiles task group collected requirements in a public database.
R-1  
Validate for uniqueness of URIs

R-102  
Intuitive Constraint Language

R-106  
Extensible Constraint Language

R-11  
Context-Specific Exclusive OR of Properties

R-113  
Interaction of Validation with Reasoning

R-117  
Define context of constraints

R-121  
Check numerical values across multiple properties

R-10  
Define Disjoint Properties

R-103  
High-Level Constraint Language

R-107  
Transformations Between Constraint Language And UML

R-110  
Transformations Between Constraint Language And SPARQL

R-114  
Provide RDF REST Services for RDF Validation

R-118  
Namespace-Sensitive Constraints

R-122  
Trade-Off Between Dimensions Expressivity, Complexity, Predictability

R-100  
Subsumption

R-104  
Constraint Language Having Implementation Language

R-108  
Transformations Between Constraint Language And XML Schema

R-111  
Basic Use Cases Covered By Constraint Language

R-115  
Closed World Assumption (CWA)

R-119  
Validation on Named Graphs

R-123  
State

R-101  
Declarative Constraint Language

R-105  
Constraint Language Translatable To Implementation Language

R-109  
Transformations Between Constraint Language And OCL

R-112  
Extensible Constraints

R-116  
Unique Name Assumption (UNA)

R-120  
Handle RDF Collections

R-124  
Describe Data

R-128  

R-75

**label:** Minimum Qualified Cardinality on Properties

**alphanumeric ID:** R-75-MINIMUM-QUALIFIED-CARDINALITY-ON-PROPERTIES

**definition:**
A minimum cardinality expression ObjectMinCardinality(n OPE CE) consists of a nonnegative integer n, an object property expression OPE, and a class expression CE, and it contains all those individuals that are connected by OPE to at least n different individuals that are instances of CE. If CE is missing, it is taken to be owl:Thing.

A minimum cardinality expression DataMinCardinality(n DPE DR) consists of a nonnegative integer n, a data property expression DPE, and a unary data range DR, and it contains all those individuals that are connected by DPE to at least n different literals in DR. If DR is not present, it is taken to be rdfs:Literal.

**examples:**
Consider the ontology consisting of the following axioms.
ObjectPropertyAssertion(a:fatherOf a:Peter a:Stewie) Peter is Stewie's father.
ClassAssertion(a:Man a:Stewie) Stewie is a man.
ObjectPropertyAssertion(a:fatherOf a:Peter a:Chris) Peter is Chris's father.
ClassAssertion(a:Man a:Chris) Chris is a man.
DifferentIndividuals(a:Chris a:Stewie) Chris and Stewie are different from each other.

The following minimum cardinality expression contains those individuals that are connected by a:fatherOf to at least two different instances of a:Man:
ObjectMinCardinality(2 a:fatherOf a:Man)

Since a:Stewie and a:Chris are both instances of a:Man and are different from each other, a:Peter is classified as an instance of this class expression.
Disclaimer: This is not an accurate visualization!
A publication needs an author!

**OWL 2:** Publication a owl:Restriction ;
  owl:minQualifiedCardinality 1 ;
  owl:onProperty author ;
  owl:onClass Person .

**ShEx:** Publication { author @Person{1, } }

**ReSh:** Publication a rs:ResourceShape ; rs:property [  
  rs:propertyDefinition author ;  
  rs:valueShape Person ;  
  rs:occurs rs:One-or-many ; ] .

**DSP:** [ dsp:resourceClass Publication ; dsp:statementTemplate [  
  dsp:minOccur 1 ;  
  dsp:property author ;  
Let’s spin....

**SPIN:** CONSTRUCT { [ a spin:ConstraintViolation ... . ] } WHERE {

?this
    a ?C1 ;
?p ?o .
BIND( STRDT ( STR ( ?c ), xsd:nonNegativeInteger ) AS ?cardinality ) .
FILTER ( ?cardinality < 1 ) .
FILTER ( ?C1 = Publication ) .
FILTER ( ?C2 = Person ) .
FILTER ( ?p = author ) .
}

**SPIN function qualifiedCardinality:**

From high-level to low-level

High Level Languages: Constraint Formulation

DSP  OWL 2  ReSh  ShEx

Low Level Language: Constraint Validation

SPIN/SPARQL
Mapping from DSP to SPIN using SPARQL

CONSTRUCT {
  _:constraintViolation a spin:ConstraintViolation ;
  rdfs:label ?violationMessage ;
  spin:violationRoot ?this ;
  spin:violationPath ?property ;
  spin:violationSource ?violationSource . }
WHERE {
  ?this a ?resourceClass .
  ?descriptionTemplate dsp:resourceClass ?resourceClass ;
    dsp:statementTemplate ?statementTemplate .
  ?statementTemplate dsp:minOccur ?minimum ;
    dsp:property ?property ;
    dsp:nonLiteralConstraint ?nonLiteralConstraint .
  ?nonLiteralConstraint dsp:valueClass ?valueClass .
  BIND ( qualifiedCardinality( ?this, ?property, ?valueClass ) AS ?cardinality ) .
  FILTER ( ?cardinality < ?minimum ) . }
Step by step:
Constraint validation using DSP

1. Write down your data **constraints**:

   ```
   :authorNeeded dsp:resourceClass Publication ;
   dsp:statementTemplate [ dsp:minOccur 1 ;
   dsp:property author ;
   dsp:nonLiteralConstraint [ dsp:valueClass Person ] ].
   ```

2. Create a **mapping** from DSP to SPARQL:
   - As shown before.
   - One mapping per DSP element.
   - This only has to be done once for DSP.

3. Load **constraints** and **mappings** and execute the validation using SPIN.

   ➔ This was our approach in our DC-2014 paper and is implemented in our demo:
   [http://purl.org/net/rdfval-demo](http://purl.org/net/rdfval-demo)
Challenge 1

A SPIN mapping constitutes two different aspects:

1. The definition of each language element (e.g. dsp:minOccur).
2. The definition of the violation that is created when validation fails.

The latter might be application specific. A consistent violation representation across mappings is desirable.

→ How can we ensure that two semantically equivalent constraints are actually validated consistently?
Challenge 2

Many constraint languages exist. Semantically equivalent constraints can be mapped from one language to another. But:

- Supporting 5 languages requires 20 mappings,
- 10 languages require already 90 mappings,
- generally $n \cdot (n-1)$ mappings (both directions) are required.

Every single language element need to be mapped. This is not feasible practically.

→ How can we support the transformation of semantically equivalent constraints from one constraint language to another?
Adding an additional layer

High Level Languages: Constraint Formulation
- DSP
- OWL 2
- ReSh
- ShEx

Intermediate Representation: Constraint Type (Requirement)
- SPARQL

RDF Constraints Vocabulary (RDF-CV)
- SPARQL

Low Level Language: Constraint Validation
- SPARQL

Bosch, Eckert: Guidance, please! Towards a Framework for RDF-based Constraint Languages.
September, 2nd 2015
RDF-CV building blocks

Constraints are either...
• Simple Constraints or
• Complex Constraints.

Complex Constraints contain Simple Constraints and/or other Complex Constraints.
Representing simple constraints

„Syntactical“ description of the constraint based on a constraining element and up to 5 parameters:

1. **Context class**: the class for which the constraint applies
2. **Left property list**: Depending on the constraining element the left side.
3. **Right property list**: the right side.
4. **Classes**: A class restriction for the constrained value.
5. **Constraining value**: a value to be used in the constraint.

Examples:

<table>
<thead>
<tr>
<th>context class</th>
<th>left p. list</th>
<th>right p. list</th>
<th>classes</th>
<th>c. element</th>
<th>c. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publication</td>
<td>author</td>
<td>-</td>
<td>Person</td>
<td>≥</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>context class</th>
<th>left p. list</th>
<th>right p. list</th>
<th>classes</th>
<th>c. element</th>
<th>c. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>authorOf, genre</td>
<td>authorOfGenre</td>
<td>T</td>
<td>property path</td>
<td>-</td>
</tr>
</tbody>
</table>
RDF-CV specification

The list of all 103 constraining elements and the representation of all 81 constraint types using RDF-CV is available in a technical report:

Mapping implementation

Example constraint:

```
:Publication a owl:Restriction ;
  owl:minQualifiedCardinality 1 ;
  owl:onProperty :author ;
  owl:onClass :Person .
```

RDF-CV:

```
[ a rdfcv:SimpleConstraint ;
  rdfcv:contextClass :Publication ;
  rdfcv:leftProperties ( :author ) ;
  rdfcv:classes ( :Person ) ;
  rdfcv:constrainingElement "minimum qualified cardinality restriction" ;
  rdfcv:constrainingValue 1 ] .
```
Mapping 1: To and from RDF-CV

- Using the usual SPARQL CONSTRUCT queries, we create the RDF-CV representation (Mapping m).
- To support transformations from one constraint language to another, another mapping $m'$ from RDF-CV back to the constraint language is added.

$$gc = m\alpha(sc\alpha)$$
$$sc\beta = m'\beta(gc)$$

→ This enables constraint transformations in a scalable way (Challenge 2).
Mapping 2: From RDF-CV to SPIN

• From RDF-CV, the SPIN representation can be created as usual.
• This mapping only has to be created only once.
• It contains the definition how exactly the constraint violations are created. Depending on the application, different mappings can of course be used.

→ This ensures that irrespective of the constraint language, semantically equivalent constraints are validated consistently (Challenge 1).
Conclusion

We think that this approach is suitable

• to implement the validation of constraints consistently across constraint languages,

• to support the extension of constraint languages when additional constraint types should be supported by means of a simple mapping, and

• to enhance or rather establish the interoperability of different constraint languages.
Thank you!

Acknowledgements: Thanks to all participants in the DCMI RDF Application Profiles task group and particularly the chairs, Karen Coyle and Antoine Isaac.
Licence

These slides are licensed according to the Creative Commons Attribution-ShareAlike Licence

http://creativecommons.org/licenses/by-sa/3.0