Towards Description Set Profiles for RDF using SPARQL as Intermediate Language

Thomas Bosch
GESIS – Leibniz Institute for the Social Sciences, Mannheim, Germany
thomas.bosch@gesis.org

Kai Eckert
Research Group Data and Web Science
University of Mannheim, Germany
kai@informatik.uni-mannheim.de

Abstract

Description Set Profiles (DSP) are used to formulate constraints on valid data within a Dublin Core Application Profile. For RDF, SPARQL is generally seen as the method of choice to validate data according to certain constraints, although it is not ideal for their formulation. In contrast, DSPs are comparatively easy to understand, but lack an implementation to validate RDF data. In this paper, we use SPIN as basic validation framework and present a general approach how domain specific constraint languages like DSP can be executed on RDF data using SPARQL as an intermediate language.

1 Introduction

In 2013, the W3C invited experts from industry, government and academia to the RDF Validation Workshop[1] to discuss use cases and requirements for constraint representation and RDF data validation. The following needs are reported:

1. Declarative definition of the structure of a graph for validation and description.
2. Extensible to address specialized use cases.
3. A mechanism to associate descriptions with data.

An important finding is that there are non-functional requirements for data validation in a Linked Data setting, particularly the need to "communicate the constraints against which data is to be validated in a way which is both easy to understand by human beings and discoverable by programs."

Partly as follow-up to the W3C workshop and partly due to further expressed requirements at the Semantic Web in Libraries conference 2013[2], the Dublin Core Metadata Initiative in collaboration with the W3C currently establishes a Task Group for RDF Application Profiles (RDF-AP) that will investigate existing approaches and best-practices, identify possible gaps and propose practical solutions for the representation of application profiles, including the formulation of data constraints.[3] In a heterogenous environment like the Web, there is not necessarily a one-size-fits-all solution, especially as existing solutions should rather be integrated than replaced, not least to avoid long and fruitless discussions about the "best" approach.

[1] RDF Validation Workshop – Practical Assurances for Quality RDF Data. 10-11 September 2013, Cambridge, MA, USA. `http://www.w3.org/2012/12/rdf-val/report`

[2] SWIB13 – Semantic Web in Libraries, 25 - 27 November 2013, Hamburg, Germany. `http://swib.org/swib13/`

[3] `http://wiki.dublincore.org/index.php/RDF-Application-Profiles`

SPARQL and SPIN are powerful and widely used for constraint formulation and validation (Fürber & Hepp, 2010), but constraints formulated as SPARQL queries are not as understandable as one wishes them to be. Consider the following example of the simple constraint stating that only dogs are allowed as pets:

```
SELECT ?this ?subope ?object WHERE {
    ?C owl:allValuesFrom :Dog .
    ?C owl:onProperty :hasPet .
    ?C a owl:Restriction .
    ?this rdf:type ?subC . ?subC rdfs:subClassOf* ?C .
    ?this ?subOPE ?object . ?subOPE rdfs:subPropertyOf* :hasPet .
    FILTER NOT EXISTS { ?object rdf:type :Dog . } }
```

This query checks the constraint and returns violating triples, but the actual constraint could be formulated easier using Description Set Profiles[4]:

```
[ a dsp:NonLiteralStatementTemplate;
  dsp:property :hasPet;
  dsp:nonLiteralConstraint [
    dsp:valueClass :Dog;
  ]
]
```

Of course, it can be argued if DSPs are the best possible way to represent constraints. They are, however, familiar to the DCMI community and tailored to the Dublin Core Abstract Model and the Singapore Framework. As stated above, there will probably be more than one constraint language that can be used in an application profile, with DSPs being one of them. This leaves the question, how the validation of data based on different constraint languages can be implemented. Different implementations using different underlying technologies hamper the interoperability of application profiles and a full implementation of several constraint languages is hard to maintain for solution providers. We therefore propose to use SPARQL as intermediate language: constraints in arbitrary languages are transformed to executable SPARQL queries used to validate the data.

This approach obviously requires that all constraint languages can be expressed in SPARQL. We have no formal proof, as use-cases and requirements still are collected and there is neither a complete list of possible constraints nor one of supported constraint languages. However, even if there are constraints that can not be translated to SPARQL, the subset of supported constraints is certainly large enough to justify the limitation to SPARQL-expressable constraints at least for one class of RDF Application Profiles, comparable to the sublanguages of OWL.

This claim is supported by the fact that SPARQL is already widely used for constraint formulation, as mentioned above. Additionally, Sirin and Tao showed how constraints can be translated to nonrecursive Datalog programs for validation (Sirin & Tao, 2009), while Angles and Gutierrez explained that SPARQL has the same expressive power as nonrecursive Datalog programs (Angles & Gutierrez, 2008).

In this paper, we present our first results regarding the implementation of our approach using SPIN. We will show that besides SPIN, no further dependencies exist. We create a full validation environment based on SPIN that can be used to validate domain specific constraint languages (Section 2). The only limitations are that the constraints have to be expressed in RDF and that the constraint language is expressable in SPARQL. In Sec-

[4]In RDF-Turtle Syntax, ommitting the surrounding description template, for details refer to `http://dublincore.org/documents/dc-dsp`

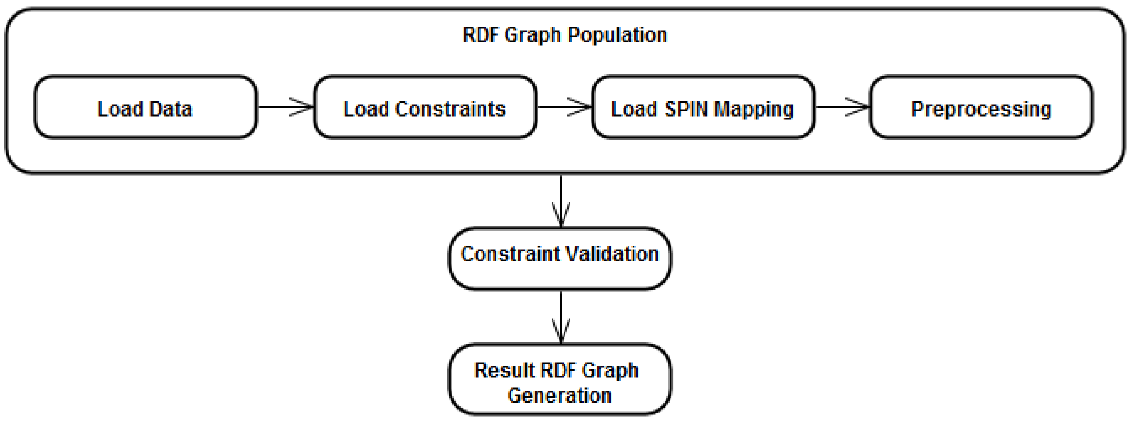

FIG 1: Constraint Validation Process

tion 3, we introduce Description Set Profiles as domain specific constraint language and subsequently describe its implementation in our environment (Section 4). We conclude in Section 5 with a discussion of open questions and an outlook to the next steps.

2 Validation Environment

We use the SPARQL Inferencing Notation (SPIN)[5] to create what we call a validation environment. The overall idea is that we see constraint languages as domain specific languages (hence domain specific constraint languages, DSCL) that are translated and executed on RDF data within our validation environment.

The translation is done once, for instance by the developer of the DSCL, and provided in form of a **SPIN mapping** plus optional **preprocessing instructions**. From a user's perspective, all that is needed is a representation of **constraints using the DSCL** and some **data to be validated** against these constraints. All these resources are purely declarative and provided in RDF or as SPARQL queries. The actual implementation is trivial using SPIN and illustrated in Figure 1.

First, an RDF graph has to be populated as follows:

1. the **data** is loaded that is to be validated,
2. the **constraints** in the DSCL are loaded,
3. the **SPIN mapping** is loaded that contains the SPARQL representation of the DSCL (see Section 4 for a detailed explanation), and
4. the **preprocessing** is performed, which can for example be provided in the form of CONSTRUCT queries.

When the graph is ready, the SPIN engine checks for each resource in the RDF data if the resource satisfies all defined constraints and generates a result RDF graph containing information about all constraint violations.

With this implementation, there is one obvious limitation of our approach: the DSCL needs an RDF serialization. For DSP, this is the case, but in the future, we would like to support non-RDF languages as well. We will further elaborate on this interesting topic in Section 5.

[5]http://spinrdf.org/

Connect SPIN to your data. SPIN uses templates for SPARQL queries that are executed on every instance of a given class – for instance `:toValidate`.

Most of the SPIN mapping that has to be created by the DSCL developer consists of such templates that are linked to a class for which the constraints should be evaluated:

```
:ToValidate
    spin:constraint
        [ a dsp2spin:StatementTemplates_MinimumOccurrenceConstraint ] .
```

As the mapping is designed to be independent of any actual data, the class `:toValidate` is purely generic. Instead of using such a generic class, it is also possible to link the constraints to `owl:Thing` or `rdfs:Resource`, i.e., to all instances.

Neither of these classes have to be assigned explicitly to instances within the data to be validated. They are either inferred using reasoning or explicitly assigned during the **preprocessing**: a reasonable approach would be to assign `:toValidate` to all classes for which constraints are actually defined – in the case of DSP classes that are linked via `dsp:resourceClass` to a description template; this can be accomplished by a suitable CONSTRUCT query that is executed before the actual validation.

After preprocessing, the data might look like this – with the added generic class in italics:

```
:ArtficialIntelligence
    a swrc:Book, :ToValidate ;
    dcterms:subject :ComputerScience .
```

`:ArtificialIntelligence` denotes a book with the assigned subject "Computer Science."

Mapping from a DSCL to SPARQL. The actual mapping is performed by creating appropriate SPARQL templates for every constraint that is supported in the DSCL, for example a minimum occurrence that is required:

```
dsp2spin:StatementTemplates_MinimumOccurrenceConstraint
    a spin:Template;
    spin:body [
        a sp:Construct ;
        sp:templates (...) ;
        sp:where (...) ] .
```

This is the general structure of a SPIN template representing a SPARQL CONSTRUCT query. We use CONSTRUCT queries to generate descriptions of each constraint violation, for instance:

```
CONSTRUCT {
    _:violation
        a spin:ConstraintViolation ;
        rdfs:label ?violationMessage ;
        spin:violationRoot ?violationRoot ;
        spin:violationPath ?violationPath ;
        spin:violationSource ?violationSource ;
        spin:fix ?violationFix ;
        :severityLevel ?severityLevel }
```

In SPIN, such a CONSTRUCT query is represented in RDF as follows:

```
a sp:Construct ;
sp:templates (
[ sp:subject _:violation ; sp:predicate rdf:type ; sp:object spin:ConstraintViolation ]
[ sp:subject _:violation ; sp:predicate rdfs:label ; sp:object [ sp:varName "violationMessage" ] ]
... ) ;
```

Constraint violation triples (1) provide useful messages explaining the reasons why RDF instances did not satisfy the constraints (`rdfs:label`), (2) contain references to RDF triples causing the constraint violations (`spin:violationRoot`), and (3) include references to the constraints causing constraint violations (`spin:violationSource`). Constraint violation triples give some guidance how to become valid data (`spin:fix`) in order to be able to fix constraint violations. Constraint violations can be classified according to different levels of severity (`:severityLevel`).

These constraint violation triples are generated for each RDF instance which matches against the WHERE clause graph pattern in the SPIN template. The SPARQL variable `this` represents the current RDF resource for which the constraint is checked.

As the mapping of a DSCL is independent of a concrete constraint specification, all constraints are generally linked to all instances (of the generic class, if applicable). Therefore, the WHERE clause of the template always have to restrict on a class for which the constraint was actually defined, for example in the case of DSP via the resource class:

```
WHERE { ?this rdf:type ?resourceClass . }
```

As for the CONSTRUCT part of the query, SPIN represents the WHERE clause in RDF as well:

```
[ sp:subject [ sp:varName "this" ] ;
    sp:predicate rdf:type ; sp:object [ sp:varName "resourceClass" ] ]
```

With this framework, we have all we need to implement our own DSCL, Description Set Profiles, which we will briefly introduce in the next section. Full examples for SPIN mappings are provided afterwards in Section 4.

3 DSP as Domain Specific Constraint Language

The Singapore Framework[6] is a framework for designing metadata and for defining Dublin Core Application Profiles (DCAP). The framework comprises descriptive components that are necessary or useful for documenting DCAPs. A DCAP is a means to assemble and to customize components from different independently created metadata standards within the context of a specific community, application, and domain.[7]

The DCMI Abstract Model (DCAM)[8] with its Description Set Model (DSM) forms the basis of Dublin Core metadata. While the DSM is highly related to RDF, it differs in some aspects worth mentioning. Table 1 shows the mappings from DSM elements to RDF triples, according to DC-RDF, the recommendation how Dublin Core metadata is represented in RDF.[9]

[6]http://dublincore.org/documents/singapore-framework/
[7]cf. `http://dublincore.org/documents/profile-guidelines/`
[8]http://dublincore.org/documents/2007/06/04/abstract-model/
[9]`http://dublincore.org/documents/dc-rdf/`

TABLE 1: DSM in RDF

DSM	RDF
Description Set	RDF graph (containing description RDF graphs)
Description	RDF graph
Resource	RDF subject: DSM resource URI (or blank node) (root of description RDF graph)
Statement	RDF subject: DSM resource RDF predicate: RDF property RDF object: DSM value (surrogate)
Non-Literal Value Surrogate	DSM value URI (or blank node)
Vocabulary Encoding Scheme	RDF subject: DSM value RDF predicate: dcam:memberOf RDF object: DSM vocabulary encoding scheme
Value String	RDF subject: DSM value RDF predicate: rdf:value RDF object: RDF Literal (DSM value string) (RDF plain literal or RDF typed literal)
Literal Value Surrogate	DSM value is RDF literal (RDF plain literal or RDF typed literal)
Value String Language	Language tag of RDF literal
Syntax Encoding Scheme	RDF datatype of RDF typed literal

A Description Set Profile (DSP)[10] contains constraints on the data within a DCAP, i.e., a DSP restricts valid descriptions of resources in a description set. Consider the following example of a DSP:

```
:bookDescriptionTemplate
    a dsp:DescriptionTemplate ;
    dsp:standalone "true"^^xsd:boolean ;
    dsp:minOccur "1"^^xsd:nonNegativeInteger ; dsp:maxOccur "infinity"^^xsd:nonNegativeInteger ;
    dsp:resourceClass swrc:Book ;
    dsp:statementTemplate [
        a dsp:NonLiteralStatementTemplate ;
        dsp:minOccur "1"^^xsd:nonNegativeInteger ; dsp:maxOccur "5"^^xsd:nonNegativeInteger ;
        dsp:property dcterms:subject ;
        dsp:nonLiteralConstraint [
            a dsp:NonLiteralConstraint ;
            dsp:descriptionTemplate :subjectDescriptionTemplate ;
            dsp:valueClass skos:Concept ;
            dsp:valueURIOccurrence "mandatory"^^dsp:occurrence ;
            dsp:valueURI :ComputerScience, :SocialScience, :Librarianship ;
            dsp:vocabularyEncodingSchemeOccurrence "mandatory"^^dsp:occurrence ;
            dsp:vocabularyEncodingScheme :BookSubjects ;
            dsp:valueStringConstraint [
                a dsp:ValueStringConstraint ;
                dsp:minOccur "1"^^xsd:nonNegativeInteger ; dsp:maxOccur "1"^^xsd:nonNegativeInteger ;
                dsp:literal "Computer Science"@en , "Computer Science"^^xsd:string ;
                dsp:literal "Social Science"@en , "Social Science"^^xsd:string ;
                dsp:literal "Librarianship"en , "Librarianship"^^xsd:string ;
                dsp:languageOccurrence "optional"^^dsp:occurrence ;
                dsp:language "en"^^xsd:language ;
                dsp:syntaxEncodingSchemeOccurrence "optional"^^dsp:occurrence ;
                dsp:syntaxEncodingScheme xsd:string ] ] ] .
```

[10]http://dublincore.org/documents/2008/03/31/dc-dsp/

A DSP consists of `dsp:DescriptionTemplate`s that put constraints on instances of a certain class, denoted by `dsp:resourceClass`. The constraints can either be constraints on the description itself, e.g., a minimum occurrence of instances of this class. Additionally, constraints on single properties can be defined within a `dsp:StatementTemplate`. The example above contains all but one of the 23 constraints defined in DSP (except the subproperty constraint; the 5 literal value constraints can be used for value strings as well).

The DSM description template `:bookDescriptionTemplate` describes DSM resources of the type `swrc:Book` (referenced by `dsp:recourceClass`). `swrc:Book` resources are allowed to occur standalone (`dsp:standalone`), i.e. without being the value of a property. Books must occur at least once (`dsp:minOccur`) and may appear multiple times (`dsp:maxOccur`) in the DSM description set (the RDF graph). The `dsp:NonLiteralStatementTemplate` restricts books to have at least 1 (`dsp:minOccur`) and at most 5 (`dsp:maxOccur`) `dcterms:subject` (`dsp:property`) relationships to DSM non-literal value surrogates which are further described by the `dsp:NonLiteralConstraint`.

The DSM values have to be of the class `skos:Concept` (`dsp:ValueClass`) and are further described in a dedicated DSM description template (referenced by `dsp:descriptionTemplate`). A value URI must be given (`dsp:valueURIOccurrence`) for DSM values and allowed value URIs (`dsp:valueURI`) are `:ComputerScience`, `:SocialScience`, and `:Librarianship`. Controlled vocabularies (like `:BookSubjects`) are represented as `skos:ConceptSchemes` in RDF and as `dsp:VocabularyEncodingSchemes` in DSM. If DSM vocabulary encoding schemes must be stated (`dsp:vocabularyEncodingSchemeOccurrence`), they have to contain the DSM values. In this case, DSM values are classified as `skos:Concepts` and are related to `skos:ConceptSchemes` via the object properties `skos:inScheme` and `dcam:memberOf` (see RDF data above).

The DSM values must be represented as exactly one (`dsp:minOccur` and `dsp:maxOccur` - line 20) of the given three DSM value strings (`dsp:literal`). The language tag `en` (`dsp:language`) as well as the RDF datatype `xsd:string` (`dsp:syntaxEncodingScheme`) may be stated (`dsp:languageOccurrence` and `dsp:syntaxEncodingSchemeOccurrence`) for DSM value strings.

An example for RDF data satisfying all these constraints for resources of the type `swrc:Book` would be:

```
:ArtficialIntelligence
        a swrc:Book , :ToValidate ;
        dcterms:subject :ComputerScience .
:ComputerScience
        skos:Concept , :ToValidate ;
        dcam:memberOf :BookSubjects ;
        skos:inScheme :BookSubjects ;
        rdf:value "Computer Science"@en .
:BookSubjects
        a skos:ConceptScheme , :ToValidate .
```

4 Mapping of DSP Constraints to SPIN

After the introduction of the general approach in Section 2, we now present a concrete example of a SPIN mapping for a DSP constraint: the DSP statement template constraint 'Minimum Occurrence Constraint' (6.1) restricts the minimum number of times the given statement must appear in the enclosing description.

This constraint is implemented by the following SPARQL query which is then represented in SPIN RDF and linked to our generic class `:ToValidate`:

```
CONSTRUCT {
    _:violation
        a spin:ConstraintViolation ;
        rdfs:label ?violationMessage ;
        spin:violationRoot ?this ;
        spin:violationSource dsp:minOccur }
WHERE {
    ?this rdf:type ?resourceClass .
    ?descriptionTemplate rdf:type dsp:DescriptionTemplate .
    ?descriptionTemplate dsp:resourceClass ?resourceClass .
    ?descriptionTemplate dsp:statementTemplate ?statementTemplate .
    ?statementTemplate dsp:minOccur ?minOccurStatement .
    ?statementTemplate dsp:property ?property .
    BIND ( ( spl:objectCount ( ?this, ?property ) ) AS ?cardinalityStatement ) .
    FILTER ( cardinalityStatement < ?minOccurStatement ) .
    BIND ( (
        fn:concat('cardinality of ', ?property, ' ( ', ?cardinalityStatement,
        ' ) < mininum cardinality of ', ?property, ' ( ', ?minOccurStatement, ' )' ) )
        AS ?violationMessage ) . }
```

It can be seen that the WHERE clause is used to "detect" constraint violations. First, a graph is matched that contains the instance data (using `?this` as instance variable) and the applicable constraint formulation from the DSP (linked to the instance via `dsp:resourceClass`). The cardinality of the property in question is added. The actual validation is implemented by the FILTER that identifies only instances that violate the constraint.

In this example, we create a violation message (`?violationMessage`) that can be displayed to the user, together with the URI of the instance (`?this` as `spin:violationRoot`) and the violated constraint (`dsp:minOccur` as `spin:violationSource`).

According to our DSP, if a resource in the RDF data

1. is assigned to the class `swrc:Book` (line 5), and
2. has no `dcterms:subject` relationships (line 8 and 9),

then the following constraint violation triple is generated:

```
_:violation
    a spin:ConstraintViolation ;
    rdfs:label
        'cardinality of dcterms:subject ( 0 ) < mininum cardinality of dcterms:subject ( 1 )' ;
    spin:violationRoot :IntroductionToAlgorithms ;
    spin:violationSource dsp:minOccur .
```

This example demonstrates how a DSP constraint is implemented in SPARQL. In the same manner, most other constraints can be implemented as well, although often the mapping gets substantially longer and more complex. There are, however, constraints that can not be implemented at all, in the case of DSP for example the literal value constraint *Syntax Encoding Scheme Constraint (6.5.4)*. It determines whether DSP syntax encoding schemes (RDF datatypes) are allowed for RDF literals, which can be 'mandatory', 'optional', or 'disallowed'.

This type of constraint cannot be validated as RDF literals always have associated datatype IRIs. If there is no datatype IRI and no language tag explicitly stated, the datatype of an RDF literal is implicitly `xsd:string`. If there is a language tag, the datatype is implicitly `rdf:langString`. Fortunately this constraint can be replaced by an equivalent constraint using *Syntax Encoding Scheme List Constraint (6.5.5)* which restricts the allowed DSP syntax encoding schemes and which is fully implemented in the SPIN mapping for DSP.

5 Conclusion and Future Work

With our approach, we were able to fully implement Description Set Profiles, apart from the exception noted above. The implementation can be tested at `http://purl.org/net/rdfval-demo`. In this paper, we describe our general approach and demonstrated its applicability to Description Set Profiles. In particular, we use SPIN as basis to define a validation environment in which domain specific constraint languages – like DSP – can be implemented by representing them in SPARQL. The approach is particularly appealing as it has only one dependency being SPIN. The implementation of the DSCL is fully declarative, consisting of a SPIN mapping in RDF and preprocessing instructions in form of SPARQL CONSTRUCT queries – which can also be represented in RDF using SPIN. It is therefore possible to link the applicable constraints in a given DSCL to an application profile, as well as the SPIN mapping and the preprocessing instructions. All that is needed to validate data according to this application profile without the need for a DSCL-specific validator. Our approach therefore fulfills an important requirement for RDF Application Profiles.

A limitation of our approach are constraints that can not be expressed in SPARQL, as for example the Syntax Encoding Scheme Constraint of DSP. In this case this is an artefact resulting from the way how RDF is implemented. There are most certainly other cases, but we argue that our approach is nevertheless useful for the majority of constraints in the majority of DSCLs. We propose, however, to document such missing constraints clearly as part of the DSCL so that users can deal with it.

Our approach is currently limited to DSCLs that are expressible in RDF. This is not necessarily a problem – the data and the data models are in RDF, so at least it is consistent – but it might be sub-optimal regarding readability and understandability of the constraints and for now excludes many existing DSCLs. We therefore plan to investigate this issue further as part of our future work. Another interesting topic is the testing of the SPIN mappings, for which test data together with expected outcomes could be provided in a certain form. Our next steps include the application to further constraint languages, first and foremost OWL2, which is already used by many to formulate constraints. The DSP mapping is developed and maintained at `https://github.com/dcmi/DSP-SPIN-Mapping`.

Acknowledgments

Kai Eckert is funded by the European Commission within the DM2E project (http://dm2e.eu).

References

Angles, R., & Gutierrez, C. (2008). The expressive power of SPARQL. In *Proceedings of the 7th International Semantic Web Conference (ISWC2008)* (pp. 114–129).

Fürber, C., & Hepp, M. (2010). Using SPARQL and SPIN for Data Quality Management on the Semantic Web. In W. Abramowicz & R. Tolksdorf (Eds.), *Business information systems* (Vol. 47, pp. 35–46). Springer Berlin Heidelberg. Retrieved from `http://dx.doi.org/10.1007/978-3-642-12814-1_4` doi: 10.1007/978-3-642-12814-1_4

Sirin, E., & Tao, J. (2009). Towards integrity constraints. In *Proceedings of the Workshop on OWL: Experiences and Directions, OWLED 2009.*