Meta-RDF concepts white paper draft

Document draft version from: 2017-07-24
Author: Johannes Frey (AKSW/KILT, Leipzig University)
Feedback / Comments: https://github.com/AKSW/meta-rdf/issues/new (tag as enhancement or question)

Contents

Introduction

Meta-RDF is a java based tool and command line utility to convert datasets into various Metadata Representation Models. It features a novel JSON representation, which allows the association of metadata to RDF quad(s) for different levels of granularity. Moreover it supports meta-metadata. Once the source dataset is converted into the JSON representation, this intermediate format can be used to create NQuads files for the various MRMs. The JSON representation is optimized for a parallel conversion of huge datasets, which do not fit into main memory. Meta-RDF supports different serialization and optimization schemes (factorization/shareCompactness, combination of ngraphs with other MRMs for efficient meta-metadata representation, logical metadata groups etc.) for the MRMs. While the JSON format is intended for a batch conversion of a complete dataset, applications can also use the integrated Java data model abstraction (DAO) to convert RDF metadata on-the-fly. The model was introduced to explicitly represent different aspects of metadata storage which can be leveraged by different MRMs. It allows among others to express different granularity and share levels, an easy way of nesting metadata and the definition of logical metadata groups.

Metadata Representation Models (MRMs) & Metadata Basics

As **Metadata Representation Model (MRM)**, we define a strategy of splitting an RDF statement or triple *t* and its set of key-value based metadata facts *m* into several triples or quads, such that we can store and query metadata - for all statements individually - in an RDF Store. The MRMs supported by MaSQue are displayed and briefly discussed in Figure 1. For a detailed explanation we refer to [1] and [2] (cpprop and rdr). As **metadata** we understand detailed, descriptive information (confidence, provenance, validity scope, traceability information, license etc.) for an individual triple or a small subset of triples from the knowledge graph. **Meta-metadata** is characterized by one or more nested layers of metadata, which describe metadata itself.

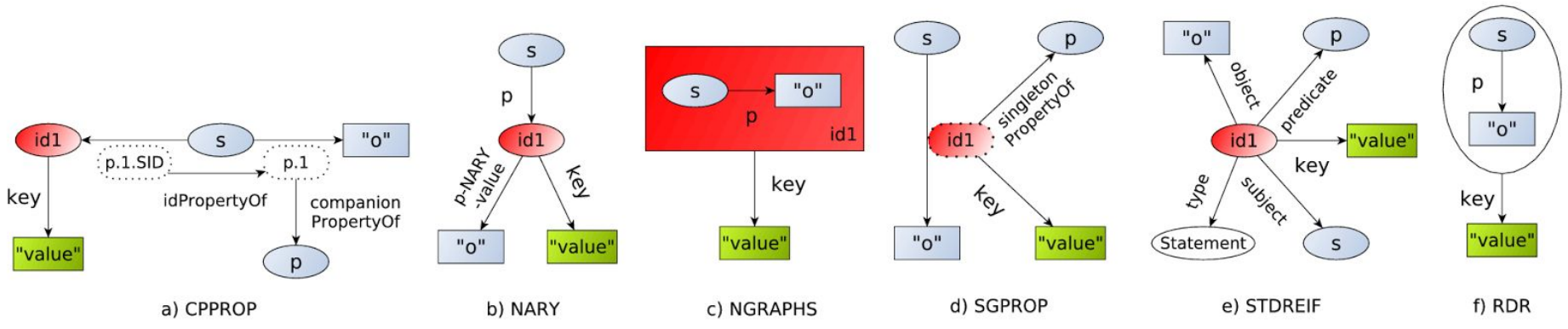


Figure 1: **Structure of different Metadata Representation Models:** Six different ways of describing (or reifying) an RDF triple `s, p, o` with a metadata `key` and `value` pair are supported by meta-rdf, Companion property (cpprop), nary relation (nary), named graphs (ngraphs), singleton properties (sgprop), standard reification (stdreif), and the Blazegraph-specific Reification Done Right (rdr). Besides rdr all approaches use an explicit statement identifier (red), which is used to attach metadata (green) to the data (grey). Cpprop and stdreif are based on additional triple handlers (white). Properties which also deal as subjects are drawn with dashed lines.

Metadata can be recorded for individual triples or sets of triples. In the context of meta-rdf we distinguish between three granularity levels. Metadata on **dataset/graph-level** provides information for all entities and statements within the same dataset / named graph. The **entity/resource-level** is the level where all statements of one entity share the same meta information. The most fine-grained metadata is on **triple-level**, where metadata is kept for each statement or triple. As **factorization** we denote the feature of cpprop and ngraphs to store shared metadata (on various granularity levels) only once. This is realized by using the same statement identifier for all statements sharing the same metadata. The remainder MRMs are not capable of this technique since they rely on the identifier to reconstruct the actual data triple or, in the case of rdr, do not use an id. Within meta-rdf we use a workaround. Instead of connecting the metadata to every statement, the metadata will be linked to a new shared resource, and only the link from the statements to that resource will be stored redundantly. Another requirement towards metadata storage is the creation of metadata fact groups or logical units. To give an example: If a fact was retrieved from two sources with two different confidence scores, the source and score form a logical unit. The confidence score does only make sense in the scope of the source.

Meta-RDF: concepts, features and options

Meta-RDF file format

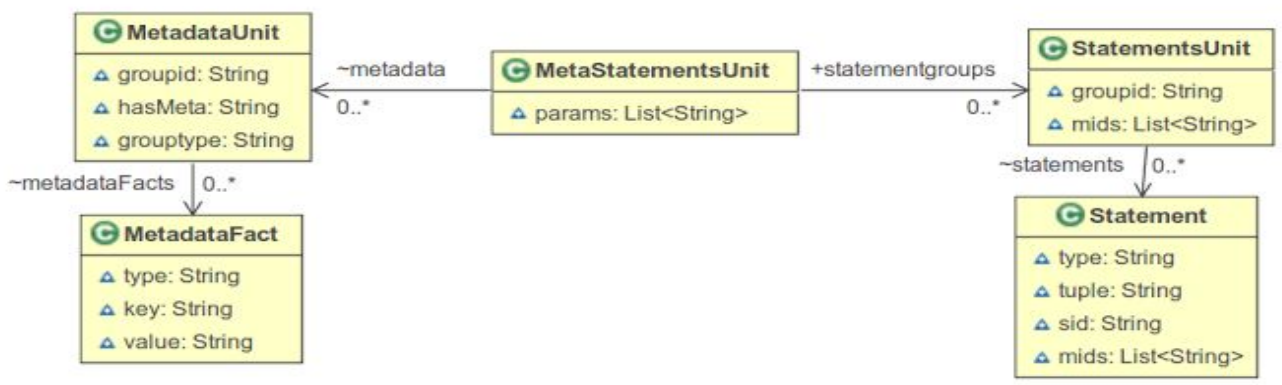
Why did we develop the dedicated JSON based meta-rdf file format (.mrdf)?

- need for an interoperable (store independ), large scale, file based abstraction layer to explicitly attach metadata to triples/quads in multiple use cases
- reuse of metadata / express of share level and different granularity
- avoids need for a sparql store or context window to run conversions (if you want explicitly attach metadata to a statement you need to know the metadata belonging to the statement (or even all other predicates and objects of a given resource if you want to convert to companion property) therefore you either would need to rely on a correct grouping /order of the statements in the nquads file or would need to query for the information
- allows per line parallelization (many core support)
- easy way of nesting metadata
- explicit support of logical groups of metadata (confidence and provenance which belong together)

⇒ in short a format which is able to represent information which can be leveraged by the different MRMs.

Meta-RDF format Data Model

The smallest independent processing unit is a MetaStatementsUnit. Every such unit is stored as a compact JSON (no newline feeds) serialization in one line of the mrdf file. Multiple units can be processed in parallel. A MetaStatementsUnit typically contains all data and metadata of one resource splitted into subunits (unordered lists) for data and metadata. A StatementsUnit contains all data triples/quads which share metadata. Metadata is associated with the mids field - a list of metadataUnits (i.e. its groupids). The list of MetadataUnits contains all MetadataUnits referenced in the mids field of StatementsUnits. A MetadataUnit groups MetadataFacts which belong together (“strong” grouptype) or which have the same meta-metadata. It can also be used to associate multiple (independent) metadata facts in one go (“flat” grouptype). The sid field of a Statement is optional and can be used to specify an explicit statement identifier for triple/quad. The remaining fields are discussed in the comments of the example below.



Example snippet

```
{  "statementgroups":[   ***contains all data triples/quads (of one resource) separated into groups
    {  ***one statement group contains all triples sharing the same metadata (entity granularity level)
      "groupid":"<http://ex.org/id>",  ***id used as (graph) identifier for this group
      "statements":[
         {
           "tuple":"<http://ex.org/person> <http://ex.org/name> \"Person\".", ***raw ntriple/nquads
           "sid":"" *** optional, can be used to specify an explicit statement identifier for triple
         }
       ],
       "mids":[ ***list of metadataUnits (link to its groupid field) which hold for that statement group
         "<http://ex.org/meta-1>", "<http://ex.org/meta-2>" ***meta-2 not listed for brevity
       ]
    }
  ],
  "metadata": [ ***contains all metadataUnits referenced in the statementgroups
      { ***a metadataUnit groups metadata facts which belong together or which have the same meta-metadata
         "groupid": "<http://ex.org/meta-1>", ***the id, if empty the id refers to the number
         "metadataFacts": [
            {
               "type": "kv-meta", ***use simple key value metadata (later version supports triples as 'values')
               "key":  "metadatakey", "value":"example value"
            }
         ],
         "grouptype": "flat" , ***shows that the metadata within the group is logically independent: 'strong' for logical unit
         "hasMetadata": ""     ***optionally specify another metadataUnit id describing this metadata unit
      }
    ],
}
```

Structure Constraints

During the conversion of a dataset into an .mrdf file, a few constraints about the file format structure need to considered. These constraints are caused by different MRMs and allow an efficient conversion into all supported MRMs.

(caused by companion property)
    all statements belonging to one subject which should be reified need to be in one MetaStatementsUnit
(caused at least by rdr and because of meta-metastatements idea)
    all metaGroups which are referenced (in fields: mids or hasMeta) in a MetaStatementsUnit msu need to exist in the StatementsGroupsUnit of msu,
(caused by rdr strongGroup companions and others? + general Idea of this format (normalization))
    a triple should be unique (so it is not allowed that a statement occurs twice) ⇒ maybe could fix this when remembering the groupcount per subject
(caused by various representation using the groupId as resource id)
    when metaGroups occur in different MetaStatementsUnits but share the same groupID then they need to look exactly the same; within a metaGroupsUnit metaGroup-Ids must be unique

Type explanations

grouptype for statementgroup

empty ""	flat group (every statement should have its own mids or none if it doesn't have metadata, if you specify a sid for a statement this will be preferred rather than creating a new one
<id-string>	all of the statements in this group share the metadata specified in the mid field of the *group* if you use shareCompactness id-string will be used as identifier for the whole group. there is just one exception if a sid for a statement is specified this will not use the id-string instead its sid will be used and will be attached to the mids of the group + the mids for this statement if existing

grouptype for metadata:

flat	this group is just virtual to link one statement(group) to multiple metadata facts without the need to link to each of them individually
graph	use the groupid as graph identifier (overrides defaultgraph)
strong	this group forms a logical unit. the facts in the group need their context (the other facts in this group) to be interpreted correctly (confidence score does only make sense if you specify the algorithm where the score came from)

metafact type:

(not checked/implemented yet; for later use; by now there is a simple check whether it's rdf or not and in case it's not,it is going to apply default keyConvert and valueConvert function)

kv-meta	keys and values are not rdf and will be transformed/converted by default keyConvert and valueConvert
kv-rdf-meta	

triple	attach triple to metadata (e.g. for rdr) ?s ?p ?o :newVersionOf ?s ?p2 ?o
kv-customMapping:<name>	name of custom keyConvert and valueConvert functions (you need to implement the functions and pass it as cp)

Command line parameters

Meta-RDF features a command line operation mode which allows to translate an rdf
command line options

short name	**full parameter name and options**	**description**
-i	--input <arg>	input file path of mrdf (JSON file) or compressed mrdf file
-o	--output <arg>	output file path pattern (file name pattern without file suffix)
-t	--threads <number>	the number of threads which should be used for parallel reading (java standard for default)
-f	--formats <list>	(comma separated) list of representation formats which should be used or all (default); available are: ngraphs,sgprop,stdreif,rdr,cpprop,nary,data
	-D<property=value>	use value for given Java property (see table below) to customize the serialization output

Serialization and optimization configuration

The way of how data will be serialized (e.g. whether the same metadata is shared or not) depends on the output MRM but can be influenced by declaring the mentioned types in the mrdf definition of the data but additionally by passing “behavior flags” as Java-Properties (either via command line or as Properties object when using meta-rdf from within Java).

Java properties

Property and options	**Description**
shareCompactness	see MRM Optimization concepts section
metaGroupsAsGraph	metadata and meta-metadata is serailized using ngraphs MRM so only the data level is reified using the specified MRM.
defaultGraph=<IRI>	specifies a default graph identifier for metadata based MRMs, where no explicit named graph is specified. See MRM Optimization concepts section.
gzOutput	compress the output files with gz
graphAsAdditionalMeta	TODO (only affects ngraph,rdr): as the original graph information of one quad has to move for the id, this option attaches it as additional metadata
forceSID	see MRM Optimization concepts section
explicitReasoning	also emit the original data triple, which could be derived via reasoning, (affects sgprop, stdreif, comp) ⇒ (disabled in current version, use data format instead and append it to your representation)

MRM Optimization concepts

Depending on the target MRM different optimization or seriailization variants can be applied.

shareCompactness

groupId as sharedId for all s p o in the group instead of individual sid **(IMPLEMENTED - ngraphs & cpprop ONLY)**
?s ?p ?o → <sharedId> (does NOT work for rdr,sgprop,stdreif,nary ...WORKAROUND using forceSID)
<sharedId> ?k ?v ∈ only once per statementGroup
new shared Resource for the group using the statementgroups groupId **(IMPLEMENTED - NOT ngraphs & rdr & cpprop)**
?s ?p ?o → ?sid
?sid :hasSharedMeta <sharedId>

<sharedId> ?k ?v ∈ only once per statementGroup
create a shared object using the groupId of the metadataUnit
?s ?p ?o → ?sid
?sid :hasMetaGroup <MetaGroupId>
<MetaGroupId> ?k ?v ∈ only once per Dataset (can not be guaranteed, just implicit deduplication when loading into sparql endpoint)

forceSID

For rdr, sgprop, stdreif, nary it would also work to attach a dedicated SID (similiar to cpprop) as the only metadata to the internal MRM id (so e.g. the singleton property predicate). This would allow to use shared ids for multiple statements as well (while still being able to distinguish the different triples using the internal MRM id). From a structural point that is basically the same like linking to the shared metadata uri, but would make the retrieval of shared metadata more consistent between the different MRMs. Note the feature is experimental and not completely implemented yet.

strongGroup

Strong groups represent logical data units, which can not be mixed with other groups because they only make sense in the context of the members in their group.

- create a grouping Resource (holding all grouped key-values) using the groupId of the metadataUnit
?s ?p ?o → ?sid
?sid :hasStrongMetaGroup <MetaGroupId>
<MetaGroupId> ?k ?v ∈ only once per Dataset (can not be guaranteed yet, just implicit deduplication when loading into sparql endpoint)

- reify the metafacts and assign them the groupId where they belong to
assign the sharedId directly (sharedId=groupId of the strong metagroup) **(IMPLEMENTED - ngraphs ONLY)**
?s ?p ?o → ?sid
?sid ?k ?v → <sharedId> ∈ (group members can be identified implicitly via its common sharedId)
(does NOT work for rdr,sgprop,stdreif,nary)

use individual sids and use groupId as nested metadata (not rdr) **(IMPLEMENTED - N/A for ngraphs & rdr)**
?s ?p ?o → ?sid
?sid ?k ?v → ?metafact-sid
?metafact-sid :belongsToMetaGroup <MetaGroupId>

- use a companion property to represent members of a group (all members of the same group have the same suffix) **(IMPL. rdr)**
?s ?p ?o → ?sid
?sid ?k+”.i” ?v (where i is a integer starting from zero, which is increased for every strong group which is attached to ?sid; ⇒ all ?k with the same i suffix can be identified as members of the same group with respect to ?sid)

nestedMetadata

Nested metadata of a metadata group is attached in any case with sharedCompactness as doing not so would blow up the number of emitted triples.

Graph behavior

What happens if a quad needs to be converted to a MRM which does not support quads (rdr) or to ngraphs which needs the graph field for the statement id. What happens with metadata triples, where a graph is not explicitly specified.

representation	**behavior**	**IDEAS/NOTES/TODO**
rdr	no graphs in general	add graph as additional metadata for statements if using graphAsAdditionalMeta property
ngraphs	graphs of statements are discarded for metadata which is neither reified nor in a strong group the defaultGraph is used elsewhere the graph-field is used as identifier field	add graph as additional metadata for statements if using graphAsAdditionalMeta property
sgprop,cpprop	the graph of the statements containing the companion/singleton property is kept for everything else the defaultGraph is used	when explicitReasoning maybe for the statements containing the companion/singleton property use also defaultGraph???

sdtreif,nary	the graph of the statements containing the companion/singleton property is kept for everything else the defaultGraph is used	

References

1. Frey, J., Müller, K., Hellmann, S., Rahm, E., and Vidal, M.-E. Evaluation of metadata representations in RDF stores. Under review in *SWJ special issue on Linked Data Benchmarking*, http://www.semantic-web-journal.net/content/evaluation-metadata-representations-rdf-stores
2. Hernández, D., Hogan, A., Riveros, C., Rojas, C., and Zerega, E. Querying wikidata: Comparing sparql, relational and graph databases. In *ISWC 2016 Proceedings, Part II* (2016), pp. 88–103.