universität wien

Faculty of Computer Science
Workflow Systems and Technology Group

Ontologies - RDF - SPARQL

University of Vienna, Austria
Workflow Systems and Technology Group (WST)

juergen.mangler@univie.ac.at

Literature

[Josh13] Joshua Tauberer: rdf:about. URL http://www.rdfabout.com/. - accessed 05.06.2013.

[Powe09] Powers, Shelley: Practical RDF : O'Reilly Media, 2009

[HaSP13] Harris, Steve; Seaborne, Andy; Prud'hommeaux, Eric: SPARQL query language for RDF. URL http://www.w3.org/TR/sparql11-query/. - accessed 05.06.2013. — W3C recommendation.

[Grub93] Gruber, Thomas R.: A translation approach to portable ontology specifications. In: Knowledge acquisition 5 (1993), Nr. 2, S. 199–220

According to Gruber (1993): "Ontologies are often equated with taxonomic hierarchies of classes, class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions — that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world.[9] To specify a conceptualization, one needs to state axioms that do constrain the possible interpretations for the defined terms." [Grub93]

Or Simpler:

- Ontology – A set of ideas/terms/words we want to become understandable, in relationship to each other. Ontologies describe a domain. Ontologies contain concepts.
- Domain – The topic or area that the words can occur in. Example:
  - Bank (Ufer) -> Domain Geography
  - Bank -> Domain Money
- Concept – A single idea, its properties and relationship to other concepts. Example:
  - A Dog is a concept.
  - A Dog is connected to the concept mammal.
  - A Dog has a name, a color and an identification number (optional).
  - Pluto is an instance of Dog.

The purpose of Ontologies is to describe the semantics of something (i.e. not how something is stored [syntax]) but how something can be understood.

How to describe Ontologies?

As Triplets:

Examples:

```
:juergen     a           :person
:eva-maria   a           :person
:martin      a           :person
:pluto       a           :dog
:martin      :hasJob     :SPL
:juergen     :hasSister  :eva-maria
:juergen     :hasFriend  :martin
:juergen     :hasFriend  :pluto
```

Formats to save and describe such triplets:

- rdfxml RDF/XML
- ntriples N-Triples
- turtle Turtle Terse RDF Triple Language
- grddl Gleaning Resource Descriptions from Dialects of Languages
  Normal HTML page (or other xml documents) contain a special link (information) to an XSL file, that transforms the document to RDF.
- rdfa RDF/A
  Normal HTML page is annotated with special attributes to identify concepts and properties
- rdfjson RDF/JSON (either Triples or Resource-Centric)
- nquads N-Quads

Example In Turtle

```
@prefix: <http://wst.univie.ac.at/> .
:juergen    a           :person .
:eva-maria  a           :person .
:martin     a           :person .
:pluto      a           :dog .
:martin     :hasJob     :SPL .
:juergen    :hasSister :eva-maria .
:juergen    :hasFriend :martin .
:juergen    :hasFriend :pluto .
```

The **“a”** is the short version of predefined rdf “type” attribute.

Each description has to contain one or more @prefix lines, which are equivalent to namespaces in xml a prefix with no name (as above) denotes the default namespace. The rdf namespace is implicit (see “a” above.

Turtle Syntax:

- URIs: <http://example.com/resource>:name or prefix:name
- Literals: "plain string" "13.4"^^xsd:float or "string with language"@en
- Triple: prefix:subject other_prefix:predicate "object" .

To transform between different ways of describing semantic information:

- Install the tools from http://librdf.org
- Ubuntu CLI: `apt-get install rasqal-utils redland-utils raptor2-utils`
- Conversion: `rapper –i [INPUT FORMAT] –o [OUTPUT FORMAT] FILE|URL`
- Example: `rapper –i guess –o rdfxml test.ttl`

Example In RDF/XML

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns="http://www.example.org/">
  <rdf:Description rdf:about="http://www.example.org/juergen">
    <rdf:type rdf:resource="http://www.example.org/person"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/eva-maria">
    <rdf:type rdf:resource="http://www.example.org/person"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/martin">
    <rdf:type rdf:resource="http://www.example.org/person"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/pluto">
    <rdf:type rdf:resource="http://www.example.org/dog"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/martin">
    <hasJob rdf:resource="http://www.example.org/SPL"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/juergen">
    <hasSister rdf:resource="http://www.example.org/eva-maria"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/juergen">
    <hasFriend rdf:resource="http://www.example.org/martin"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/juergen">
    <hasFriend rdf:resource="http://www.example.org/pluto"/>
  </rdf:Description>
</rdf:RDF>
```

Special Prefixes

Prefixes identify a certain set of predefined predicates or objects. The most widely know prefix is Dublin Core: http://dublincore.org/

Use in your Ontology:

```
<rdf:RDF
         xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
         xmlns:dc="http://purl.org/dc/elements/1.1/">
```

or

```
@prefix dc: <http://purl.org/dc/elements/1.1/>
```

Annotate Stuff:

```
<rdf:Description>
  <dc:title>Internet Ethics</dc:title>
  <dc:creator>Duncan Langford</dc:creator>
  <dc:format>Book</dc:format>
  <dc:identifier>ISBN 0333776267</dc:identifier>
</rdf:Description>
```

or

```
:bible a :book .
:bible dc:title “The Holy Bible” .
:bible dc:creator “Whoever had something to say” .
```

Special Prefixes (ctd.)

Prefixes identify a certain set of predefined predicates or objects. The most widely know prefix is Dublin Core: http://dublincore.org/

Dublin core defines the following relations:

- Title
- Creator
- Subject
- Description
- Publisher
- Contributor
- Date
- Type
- Format
- Identifier
- Source
- Language
- Relation
- Coverage
- Rights

What to do with Ontologies?

We use them to connect our semantics to other semantics:

- Company A has an ontology
- Company B has an ontology
- Company A creates a new ontology to describe how the concepts of company A are related to their own concepts

We use them to identify concepts and extract data!

- Identify all persons that have sisters and green eyes.
- Identify all persons that have dead uncles.
- …

A common language to query is SPARQL. Structure:

```
# prefix declarations
PREFIX foo: <http://example.com/resources/> ...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE { ... }
# query modifiers
ORDER BY ...
```

SPARQL

SPARQL queries are executed against *RDF datasets (one or more).*

The results of SPARQL queries can be returned and/or rendered in a variety of formats:

- *XML*. SPARQL specifies an XML vocabulary for returning tables of results.
- *JSON*. A JSON "port" of the XML vocabulary, particularly useful for Web applications.
- *CSV/TSV*. Simple textual representations ideal for importing into spreadsheets
- *RDF*. Certain SPARQL result clauses trigger RDF responses, which in turn can be serialized in a number of ways (RDF/XML, N-Triples, Turtle, etc.)
- *HTML*. When using an interactive form to work with SPARQL queries. Often implemented by applying an XSL transform to XML results.

SPARQL 1.0 became a standard in January, 2008, and included:

- **SPARQL 1.0 Query Language** - for matching patterns in RDF data
- **SPARQL 1.0 Protocol** - for sending queries over HTTP
- **SPARQL Results XML Format**
- **SPARQL Results JSON Format**

SPARQL 1.1 became a standard in March, 2013, and included:

- **Updated 1.1 versions of SPARQL Query and SPARQL Protocol**
- **SPARQL 1.1 Update** - for inserting, deleting, modifying RDF data
- **SPARQL 1.1 Graph Store HTTP Protocol** - RESTful access of RDF graphs
- **SPARQL 1.1 Service Descriptions** - describe capabilities of SPARQL endpoints
- **SPARQL 1.1 Entailments** - how to combine reasoning with SPARQL
- **SPARQL 1.1 Basic Federated Query** - querying multiple endpoints at once
- **SPARQL Results CSV/TSV Formats**

The librdf.org tools provide SPARQL 1.1 (with some minor exceptions).

SPARQL

SPARQL queries are executed against *RDF datasets (one or more).*

The results of SPARQL queries can be returned and/or rendered in a variety of formats:

- *XML*. SPARQL specifies an XML vocabulary for returning tables of results.
- *JSON*. A JSON "port" of the XML vocabulary, particularly useful for Web applications.
- *CSV/TSV*. Simple textual representations ideal for importing into spreadsheets
- *RDF*. Certain SPARQL result clauses trigger RDF responses, which in turn can be serialized in a number of ways (RDF/XML, N-Triples, Turtle, etc.)
- *HTML*. When using an interactive form to work with SPARQL queries. Often implemented by applying an XSL transform to XML results.

SPARQL 1.0 became a standard in January, 2008, and included:

- **SPARQL 1.0 Query Language** - for matching patterns in RDF data
- **SPARQL 1.0 Protocol** - for sending queries over HTTP
- **SPARQL Results XML Format**
- **SPARQL Results JSON Format**

SPARQL 1.1 became a standard in March, 2013, and included:

- **Updated 1.1 versions of SPARQL Query and SPARQL Protocol**
- **SPARQL 1.1 Update** - for inserting, deleting, modifying RDF data
- **SPARQL 1.1 Graph Store HTTP Protocol** - RESTful access of RDF graphs
- **SPARQL 1.1 Service Descriptions** - describe capabilities of SPARQL endpoints
- **SPARQL 1.1 Entailments** - how to combine reasoning with SPARQL
- **SPARQL 1.1 Basic Federated Query** - querying multiple endpoints at once
- **SPARQL Results CSV/TSV Formats**

The librdf.org tools provide SPARQL 1.1 (with some minor exceptions).

SPARQL by Example

```
PREFIX wst: <http://wst.univie.ac.at/>
SELECT distinct ?person
FROM <test.ttl>
WHERE {
  ?person a wst:person.
}
```

From file test.ttl, find all distinct persons. Distinct can be used to filter out duplicate entries. Duplicate entries can e.g. also occur when it is possible to find out in multiple ways that :juergen is a person (e.g. when you look for all teachers and all persons, and teachers are also persons).

- SPARQL *variables* start with a ? and can match any node (resource or literal) in the RDF dataset.
- *Triple patterns* are just like triples, except that any of the parts of a triple can be replaced with a variable.
- The *SELECT* result clause returns a table of variables and values that satisfy the query.
- Dataset: test.ttl (could also be e.g. http://wst.univie.ac.at/test.rdf)

SPARQL by Example

```
PREFIX wst: <http://wst.univie.ac.at/>
SELECT distinct ?person
FROM <test.ttl>
WHERE {
  ?person a wst:person.
}
```

From file test.ttl, find all distinct persons. Distinct can be used to filter out duplicate entries. Duplicate entries can e.g. also occur when it is possible to find out in multiple ways that :juergen is a person (e.g. when you look for all teachers and all persons, and teachers are also persons).

- SPARQL *variables* start with a ? and can match any node (resource or literal) in the RDF dataset.
- *Triple patterns* are just like triples, except that any of the parts of a triple can be replaced with a variable.
- The *SELECT* result clause returns a table of variables and values that satisfy the query.
- Dataset: test.ttl (could also be e.g. http://wst.univie.ac.at/test.rdf)

Use librdf.org **roqet** tool the check the results:

- Save query in `test.rq`
- Syntax: `roqet -i [INPUT FORMAT] -q [FILE]`
  -q stands for quiet to suppress debug information
- Example: `roqet -i sparql -q test.rq`
- Result:

```
result: [person=uri<http://www.example.org/juergen>]
result: [person=uri<http://www.example.org/eva-maria>]
result: [person=uri<http://www.example.org/martin>]
```

- Errors: When something goes wrong instead of the result an error is presented.
- Also see: http://goo.gl/L0ACK

SPARQL by Example

```
PREFIX wst: <http://wst.univie.ac.at/>
SELECT *
FROM <test.ttl>
WHERE {
  ?person a wst:person
  ?person wst:hasJob ?SPL
}
```

Result:

```
[
    person=uri<http://www.example.org/martin>,
    SPL=uri<http://www.example.org/SPL>
]
```

Alternative:

```
PREFIX wst: <http://www.example.org/>
SELECT *
FROM <test.ttl>
WHERE {
  ?person a wst:person
  ?person wst:hasJob <http://wst.univie.ac.at/SPL>
}
```

We can use multiple triple patterns to retrieve multiple properties about a particular resource.
Shortcut: *SELECT* * selects all variables mentioned in the query.

SPARQL by Example

```
PREFIX wst: <http://wst.univie.ac.at/>
SELECT distinct *
FROM <test.ttl>
WHERE {
  ?person a wst:person
  ?person wst:hasJob ?SPL
}
LIMIT 50;
```

LIMIT is a *solution modifier* that limits the number of rows returned from a query. SPARQL has two other solution modifiers:

- ORDER BY for sorting query solutions on the value of one or more variables
- OFFSET, used in conjunction with LIMIT and ORDER BY to take a slice of a sorted solution set (e.g. for paging)

The SPARQL keyword a is a shortcut for the common predicate rdf:type, giving the class of a resource.
A DISTINCT modifier eliminates duplicate rows from the query results.

```
PREFIX wst: <http://wst.univie.ac.at/>
SELECT …
FROM …
WHERE {
  …
  FILTER (?variable < … && ?variable > …)
}
```

FILTER constraints use boolean conditions to filter out unwanted query results.

SPARQL by Example - FILTER

Allowed in FILTER (the red functions are currently not available in librdf.org tools – 2013-06-05):

- Logical: !, &&, ||
- Math: +, -, *, /
- Comparison: =, !=, >, <, IN, NOT IN...
- SPARQL tests: isURI, isBlank, isLiteral, isNumeric, bound
- SPARQL accessors: str, lang, datatype
- Other: sameTerm, langMatches, regex, REPLACE
- Conditionals (SPARQL 1.1): IF, COALESCE, **EXISTS, NOT EXISTS**
- Constructors (SPARQL 1.1): URI, BNODE, STRDT, STRLANG, UUID, STRUUID
- Strings (SPARQL 1.1): STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, STRENDS, CONTAINS, STRBEFORE, STRAFTER, CONCAT, ENCODE_FOR_URI
- More math (SPARQL 1.1): abs, round, ceil, floor, RAND
- Date/time (SPARQL 1.1): now, year, month, day, hours, minutes, seconds, timezone, tz
- Hashing (SPARQL 1.1): MD5, SHA1, SHA256, SHA384, SHA512

SPARQL by Example

```
PREFIX wst: <http://wst.univie.ac.at/>
SELECT distinct *
FROM <test.ttl>
WHERE {
  ?person a wst:person
  OPTIONAL {?person wst:hasPicture ?pic }
}
```

Not every person has a picture associated.

OPTIONAL tries to match a graph pattern, but doesn't fail the whole query if the optional match fails.

If an OPTIONAL pattern fails to match for a particular solution, any variables in that pattern remain *unbound* (no value) for that solution.

Please check the filter function **bound** on the last slide. With bound or !bound it is possible to check if a person has a picture. Thus it is possible to filter for all persons without picture.

```
PREFIX wst: <http://wst.univie.ac.at/>
SELECT distinct *
FROM <test.ttl>
WHERE {
  { ?person a wst:person }
  UNION
  { ?person wst:socialSecurityNumber ?num }
}
```

Only persons have a social scurity number, thus the second part is a refinement of the first. Without UNION the results would only comprise the second part.

SPARQL MORE Features

The BIND keyword lets you assign a value to a variable within the body of a query:

```
BIND(?birthYear – now AS ?age)
```

[] can be used if no variable is to be assigned.

```
[] a <http://wst.univie.ac.at/Person>
```

SPARQL has an **UPDATE language**. If not a FILE, but an URL is given it is possible to create triplets and push (HTTP POST) them to URLs.

Multiple locations. SPARQL allows to have multiple FROM locations.

Projected expressions. It is possible to query results to contain values derived from constants, function calls, or other expressions in the SELECT list.

Aggregates. It is possible to group results and calculate aggregate values (e.g. count, min, max, avg, sum, ...), and filter them (i.e. having).

Subqueries. one query can be embedded within another.

Property paths. It is possible to query arbitrary length paths through a graph via a regular-expression-like syntax known as property paths (e.g. rdf:type/wst:hasJob*).

Basic federated query. It is possible to split a single query among multiple SPARQL endpoints and combine together the results from each.

universität wien

Fin.