Investigating learnability, user performance, and preferences of the path query language SemwidgQL compared to SPARQL

—

Appendix*

Timo Stegemann and Jürgen Ziegler

University of Duisburg-Essen, Duisburg, Germany,
timo.stegemann@uni-due.de,
http://interactivesystems.info

Abstract. Appendix for "Investigating learnability, user performance, and preferences of the path query language SemwidgQL compared to SPARQL".

A Appendix

This appendix contains the three-page handout, the participants received at the beginning of the seminar. This handout contained a small RDF graph (Fig. 1) that was used for the seminar and the evaluation, as well as an overview of relevant SPARQL (Fig. 2) and SemwidgQL (Fig. 3) commands. Tables 1–12 are showing the tasks of the evaluation and their corresponding sample solutions. The prefix definitions of SPARQL and SemwidgQL queries were predefined and did not have to be entered by the participant. We list them here for the sake of completeness.

* This appendix is accompanying our ISWC'17 research track paper [1]

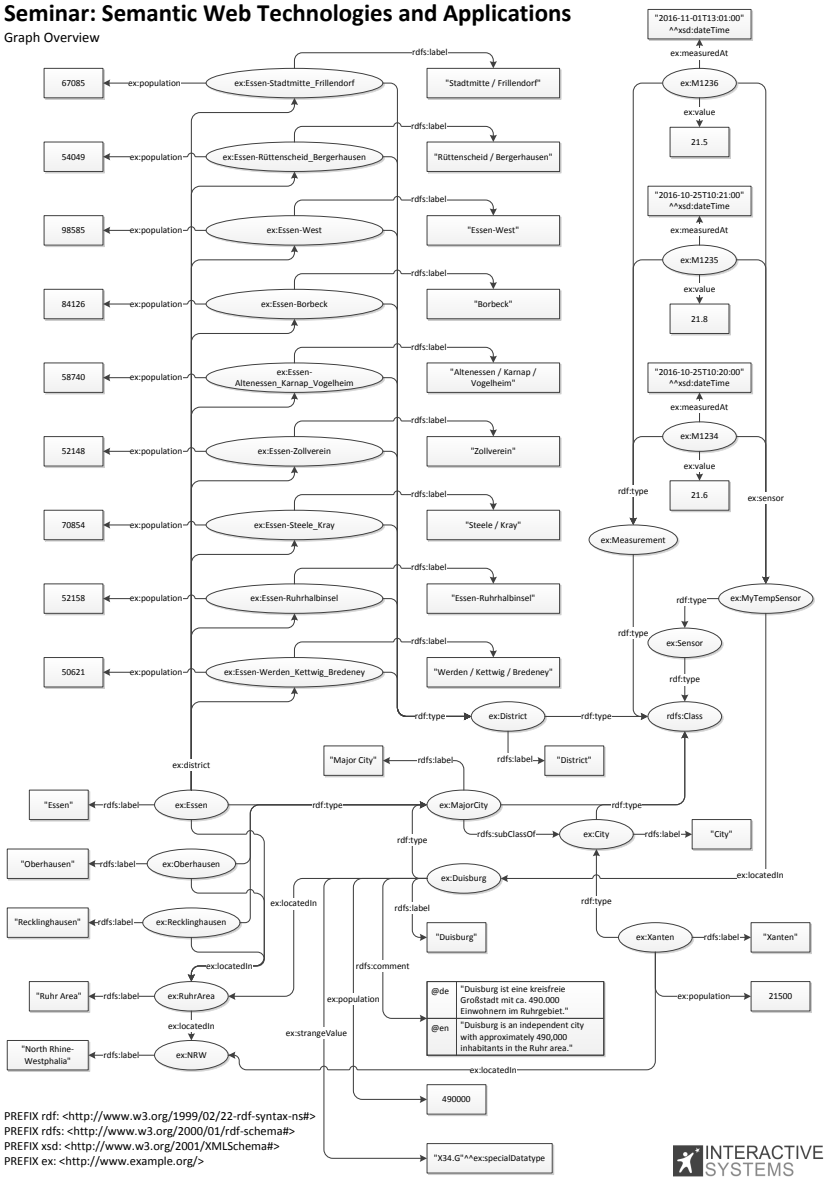


Fig. 1: Handout from the seminar "Semantic Web Technologies and Applications". The graph shows the data used for the examples and tasks from the lecture and the user study.

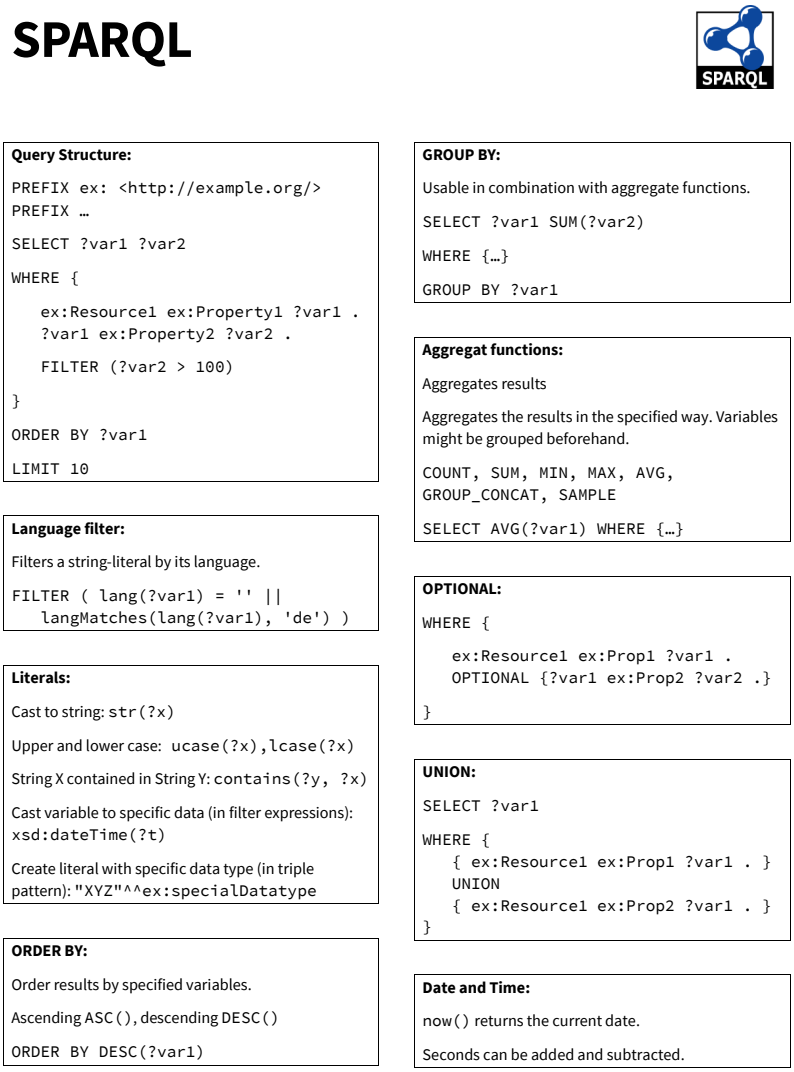
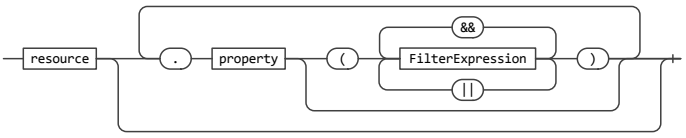


Fig. 2: Handout from the seminar "Semantic Web Technologies and Applications". The handout contains all S PARQL commands that are relevant for solving the tasks of the user study.

SemwidgQL



Query Structure:

`ex:Resource1.ex:Prop1.ex:Prop2`

Multiple Property Selection:

`ex:Resource1.[ex:Prop1, ex:Prop2]`

Wildcards:

For resources and properties.

`*.ex:Prop1, ex:Resource1.*`

Inverse Properties:

Inversion oft the subject-predicate-object relation. Returns the subject of this relation instead of the object.

`ex:Resource1.^ex:Prop1`

Filter:

Relate to the property if the object that is filtered.

`*(ex:Prop1 > 100)`

Additional "Contains" operator:

`*(ex:Prop1 ~ 'xyz')`

Filter Expressions:

`@lang` – Language filter.
`(@lang = 'de')`

`@self` – Access to the value of the filter property.
`(@self > 10)`

`@type` – Abbreviation for `rdf:type`.
`(@type = ex:MyClass)`

`@timestart / @timeend` – Filtering of time-related data. Relative times are possible.
`(@timestart = "now - 2 hours")`

`Pseudo-Filter:`

`@aggregate` – Aggregates the results in the specified way. Possible values are `COUNT`, `SUM`, `MIN`, `MAX`, `AVG`, `GROUP_CONCAT`, and `SAMPLE`. If necessary, other values must be removed with `@hide`.
`(@aggregate = 'SUM')`

`@hide` – Removes specified parts from the result.
`(@hide = true)`

`@optional` – marks specified Properties as optional.
`(@optional = true)`

`@predicate` – Adds the corresponding predicate of the S-P-O relation to the result.
`(@predicate = true)`

`@timeinterval` – Aggregates time-related data within the specified interval.
`(@timeinterval = '10 mins')`

Fig. 3: Handout from the seminar "Semantic Web Technologies and Applications". The handout contains all SemwidgQL commands that are relevant for solving the tasks of the user study.

Table 1: Task 1 of the user study

Task 1:	Interpret the semantics of the following SPARQL / SemwidgQL query. What information is queried?
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?x`
	`WHERE {`
	`  ex:Duisburg rdfs:comment ?x .`
	`}`

SemwidgQL:	`ex:Duisburg.rdfs:comment`
Solution:	All comments of Duisburg are queried.

Table 2: Task 2 of the user study

Task 2:	Interpret the semantics of the following SPARQL / SemwidgQL query. What information is queried?
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?x ?y`
	`WHERE {`
	`  ex:Essen ex:stadtbezirk ?x .`
	`  ?x rdfs:label ?y .`
	`}`

SemwidgQL:	`ex:Essen.ex:stadtbezirk.rdfs:label`
Solution:	All the labels of all districts of Essen are queried.

Table 3: Task 3 of the user study

Task 3:	Interpret the semantics of the following SPARQL / SemwidgQL query. What information is queried?
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT DISTINCT ?y ?z`
	`WHERE {`
	`  ?x ex:sensor ex:MyTempSensor .`
	`  ?x ex:value ?y .`
	`  FILTER (`
	`  ?y > 21.7`
	`  ) .`
	`  ?x ex:measuredAt ?z .`
	`}`

SemwidgQL:	`ex:MyTempSensor.^ex:sensor.` `  [ex:value(@self > 21.7), ex:measuredAt]`
Solution:	All sensor values of "ex:MyTempSensor" are queried that are higher than 21.7 degree. Additionally, the corresponding time stamp is queried.

Table 4: Task 4 of the user study

Task 4:	Query the population of Duisburg.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?pop`
	`WHERE {`
	`  ex:Duisburg ex:population ?pop .`
	`}`

SemwidgQL:	`ex:Duisburg.ex:population`

Table 5: Task 5 of the user study

Task 5:	Query all districts of Essen and their corresponding population.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?district ?pop`
	`WHERE {`
	`  ex:Essen ex:district ?district .`
	`  ?district ex:population ?pop .`
	`}`

SemwidgQL:	`ex:Essen.ex:district:ex.population`

Table 6: Task 6 of the user study

Task 6:	Query the label and comment of Duisburg.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?label ?comment`
	`WHERE {`
	`  ex:Duisburg rdfs:label ?label .`
	`  ex:Duisburg rdfs:comment ?comment .`
	`}`

SemwidgQL:	`ex:Duisburg.[rdfs:label, rdfs:comment]`

Table 7: Task 7 of the user study

Task 7:	Query the German-language comment of Duisburg.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?comment`
	`WHERE {`
	`  ex:Duisburg rdfs:comment ?comment .`
	`  FILTER (`
	`  langMatches(lang(?comment), 'de')`
	`  ) .`
	`}`

SemwidgQL:	`ex:Duisburg.rdfs:comment(@lang = 'de')`

Table 8: Task 8 of the user study

Task 8:	Query the labels of all cities in the Ruhr area.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?label`
	`WHERE {`
	`  ?cities ex:locatedIn ex:RuhrArea .`
	`  ?cities rdfs:label ?label .`
	`}`

SemwidgQL:	`ex:RuhrArea.^ex:locatedIn.rdfs:label`

Table 9: Task 9 of the user study

Task 9:	Query the labels of all districts of Essen with more than 60,000 inhabitants.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?label`
	`WHERE {`
	`  ex:Essen ex:district ?district .`
	`  ?district ex:population ?pop .`
	`  FILTER (`
	`  ?pop > 60000`
	`  ) .`
	`  ?district rdfs:label ?label .`
	`}`

SemwidgQL:	`ex:Essen.ex:district(ex:population > 60000).rdfs:label`

Table 10: Task 10 of the user study

Task 10:	Query the label of all major cities.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?label`
	`WHERE {`
	`  ?cities rdf:type ex:MajorCity .`
	`  ?cities rdfs:label ?label .`
	`}`

SemwidgQL:	`*(@type = ex:MajorCity).rdfs:label`

Table 11: Task 11 of the user study

Task 11:	Query the label of all major cities that have the string "hausen" in their name.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT ?label`
	`WHERE {`
	`  ?cities rdf:type ex:MajorCity .`
	`  ?cities rdfs:label ?label .`
	`  FILTER (`
	`  CONTAINS( LCASE(STR(?label)), LCASE(STR('hausen')) )`
	`  ) .`
	`}`

SemwidgQL:	`*(@type = ex:MajorCity).rdfs:label(@self ~ 'hausen')`

Table 12: Task 12 of the user study

Task 12:	Query all measurements that where recorded within the last week.
Prefixes:	`rdfs: <http://www.w3.org/2000/01/rdf-schema#>` `ex: <http://www.example.org/>`

SPARQL:	`SELECT DISTINCT ?measurement`
	`WHERE {`
	`  ?measurement rdf:type ex:Measurement .`
	`  ?measurement ex:measuredAt ?measuredAt .`
	`  FILTER (`
	`  xsd:dateTime(?measuredAt) >= now() - 60*60*24*7`
	`  ) .`
	`}`

SemwidgQL:	`*(@type = ex:Measurement).ex:measuredAt(@timestart =` `  'now - 1 week' && @hide = true)`

References

1. Stegemann, T., Ziegler, J.: Investigating learnability, user performance, and preferences of the path query language semwidgql compared to sparql. In: d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. (eds.) The Semantic Web – ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I. pp. 611–627. Springer International Publishing, Cham (2017), https://doi.org/10.1007/978-3-319-68288-4_36