Automatic Bootstrapping of GraphQL Endpoints for RDF Triple Stores

Lars Gleim **Tim Holzheim** István Koren Stefan Decker

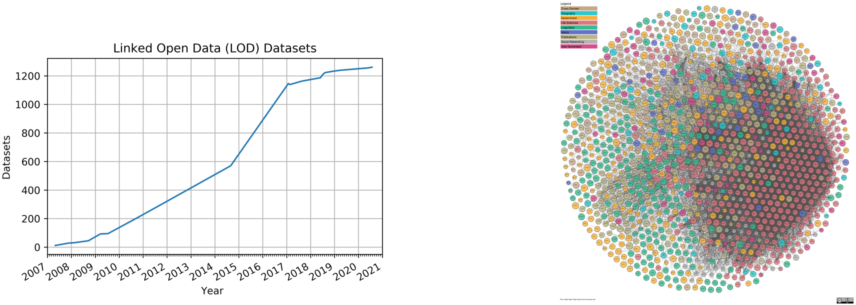
{gleim@dbis, tim.holzheim@, koren@dbis, decker@dbis} .rwth-aachen.de

Chair of Computer Science 5 Information Systems & Databases
RWTH Aachen University

QuWeDa 2020: 4th Workshop on Storing, Querying, and Benchmarking the Web of Data ISWC 2020

Linked Data

- Rapid growth of Linked Data
- Increasing importance with the rise of IoT

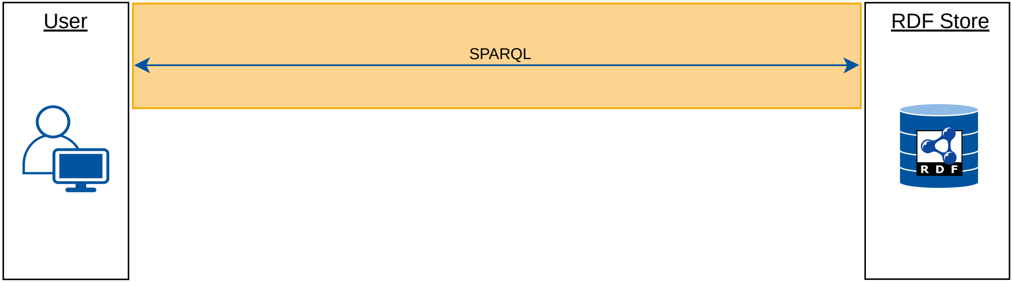


(a)

(b)

Growth (a) and current state (b) of LOD [1]

Linked Data - SPARQL



User

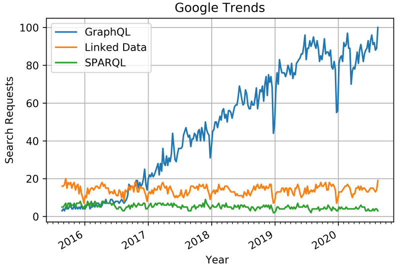
SPARQL

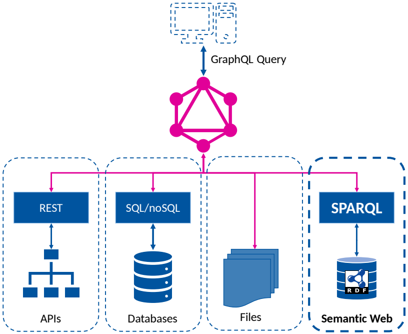
RDF Store

- Query writing requires
  - In-depths knowledge about data structure
  - Experience in queries on graphs
- SPARQL results criticized for
  - Unnecessary metadata
  - Duplicate data

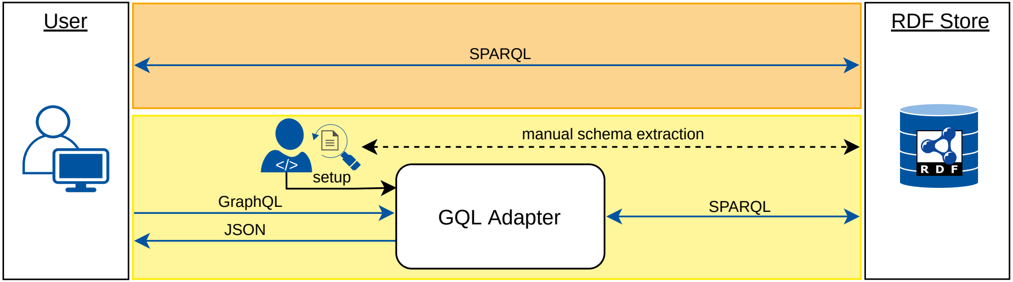
GraphQL - Rising Popularity

- Simplistic API Query language
  - Tree structure well-known by developers
  - No duplicate data
  - Schema introspection
- Wrapper language for underlying data structure





GraphQL - Overview of Current Approaches

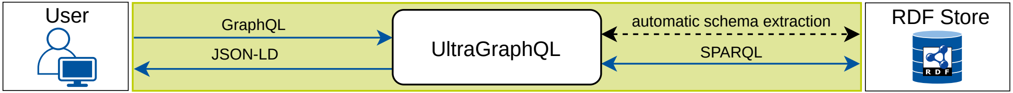


GraphQL - Overview of Current Approaches

Features	**GQLM [2]**	**MGQL [3]**	**GQLD [4]**	**Stardog [5]**	**TopBraid [6]**	**HGQL [7]**
Automatic Schema Extraction	(✓)	(✓)	-	(✓)	(✓)	-
Schema Introspection	✓	✓	-	(✓)	✓	✓
RDF-interpretable Results	-	-	✓	-	-	✓
Filtering and Ordering	-	-	✓	✓	✓	(✓)
Federated Query Support	-	-	✓	-	-	✓
Mutation Support	-	-	-	-	✓	-
License	None	Apache	MIT	Commercial	Commercial	Apache

Overview of GraphQL to RDF tools – '(✓)' denotes partial support

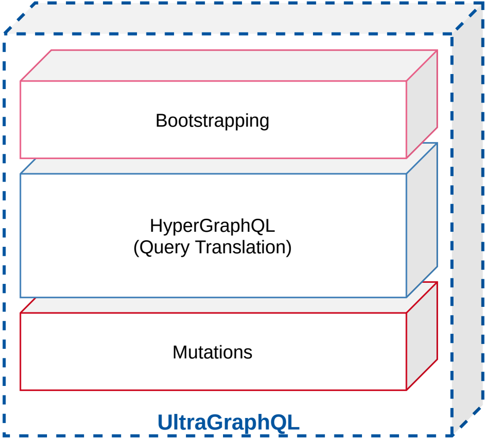
UGQL - Brief Overview I



Features	**UGQL**
Automatic Schema Summarization	✓
Schema Introspection	✓
RDF-interpretable Results	✓
Filtering and Ordering	✓
Federated Query Support	✓
Mutation Support	✓
License	Apache

UGQL - Brief Overview II

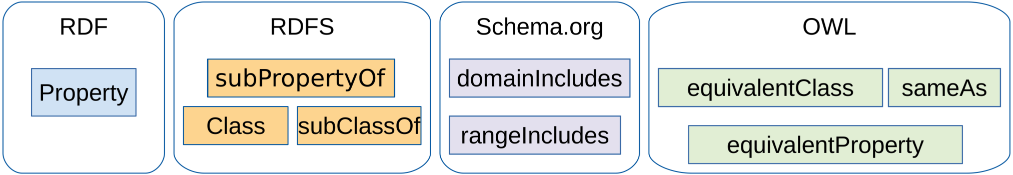
- Bootstrapping
  - Schema summarization
  - Schema mapping
- HGQL feature extensions
- Mutations
  - Providing CRUD operations
  - Translation to SPARQL Update



*HGQL heavily modified

Schema Extraction and Summarization

- Schema summarization with SPARQL Query
  - Fixed vocabulary (configurable)
  - Default vocabuary is class/property based
  - Query and vocabulary configurable
- GraphQL schema less expressive than RDF
  - Vocabulary limitation necessary



RDF

Property

RDFS

subPropertyOf

Class

subClassOf

Schema.org

domainIncludes

rangeIncludes

OWL

equivalentClass

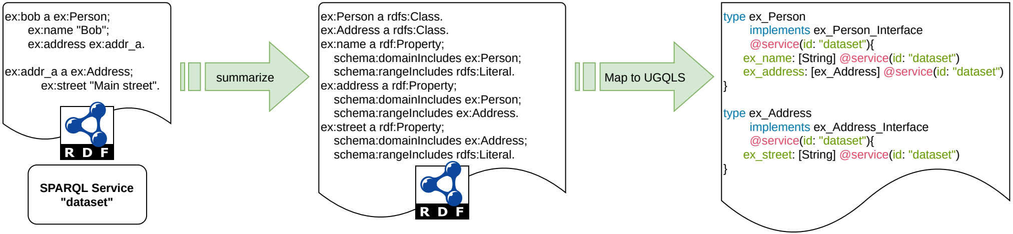
sameAs

equivalentProperty

Schema Mapping

- Mapping from class/property to object/field structure
- Features that are natively not supported by GraphQL schema are either
  - Mapped to new structure inside the schema, or
  - Mapped to directives (resolved at runtime)
- Schemas of multiple Services are merged together
  - Maintaining relations across services
- Details covered in the paper and code documentation

Bootstrapping Overview



```
ex:bob a ex:Person;
      ex:name "Bob";
      ex:address ex:addr_a.

ex:addr_a a ex:Address;
         ex:street "Main street".
```

```
ex:Person a rdfs:Class.
ex:Address a rdfs:Class.
ex:name a rdf:Property;
   schema:domainIncludes ex:Person;
   schema:rangeIncludes rdfs:Literal.
ex:address a rdf:Property;
   schema:domainIncludes ex:Person;
   schema:rangeIncludes ex:Address.
ex:street a rdf:Property;
   schema:domainIncludes ex:Address;
   schema:rangeIncludes rdfs:Literal.
```

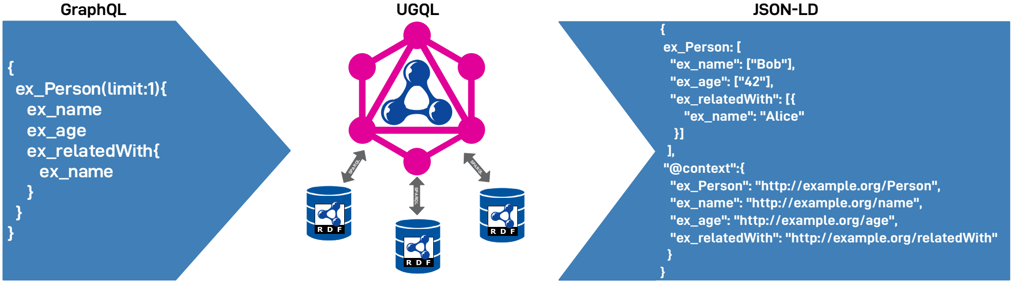
```
type ex_Person
      implements ex_Person_Interface
      @service(id: "dataset"){
    ex_name: [String] @service(id: "dataset")
    ex_address: [ex_Address] @service(id: "dataset")
}

type ex_Address
      implements ex_Address_Interface
      @service(id: "dataset"){
    ex_street: [String] @service(id: "dataset")
}
```

- Automatic generation of UGQL schema (UGQLS)
  - Allows booting UGQL
- Feature set of HGQL extended to support all features of the generated schema
  - HGQL does not support the full GraphQL schema specification

Queries

- Generation of CRUD operations based on UGQLS
  - Objects of the schema are the root of each query



GraphQL

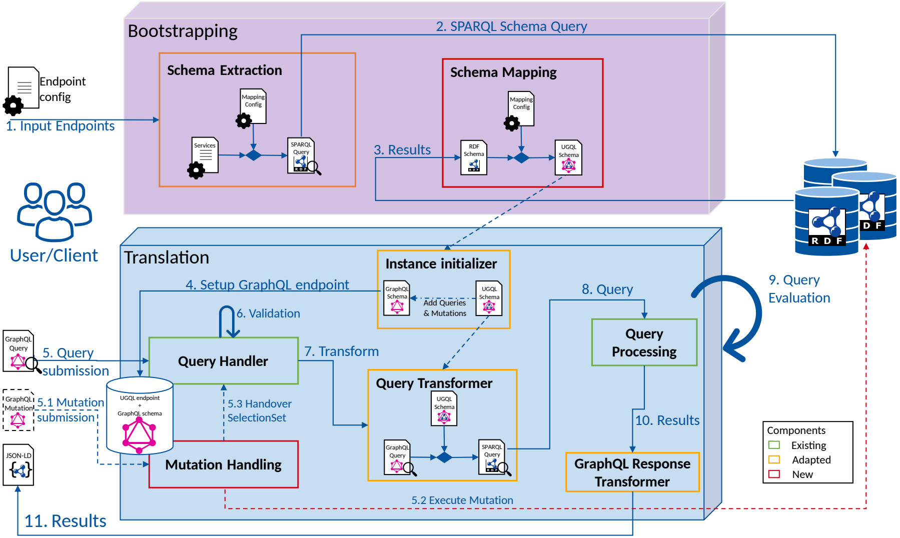
```
{
 ex_Person(limit:1){
   ex_name
   ex_age
   ex_relatedWith{
     ex_name
   }
 }
}
```

UGQL

JSON-LD

```
{
 ex_Person: [
  "ex_name": ["Bob"],
  "ex_age": ["42"],
  "ex_relatedWith": [{
     "ex_name": "Alice"
   }]
  ],
 "@context":{
  "ex_Person": "http://example.org/Person",
  "ex_name": "http://example.org/name",
  "ex_age": "http://example.org/age",
  "ex_relatedWith": "http://example.org/relatedWith"
  }
}
```

UltraGraphQL Realization

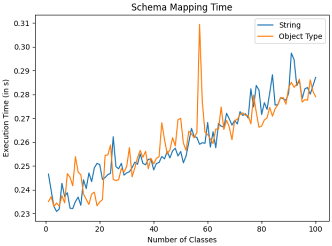


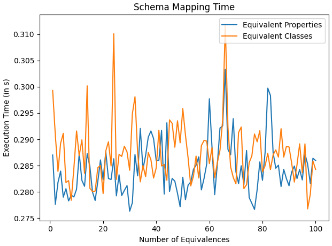
Evaluation

- Query and result transformation generate
  - Time overhead
  - Size overhead

Bootstrapping Evaluation

- Bootstrapping phase only at service start-up
  - Schema summarization takes orders of magnitudes longer
- Mapping time increases with the schema size
  - Relations between entities have minor impact





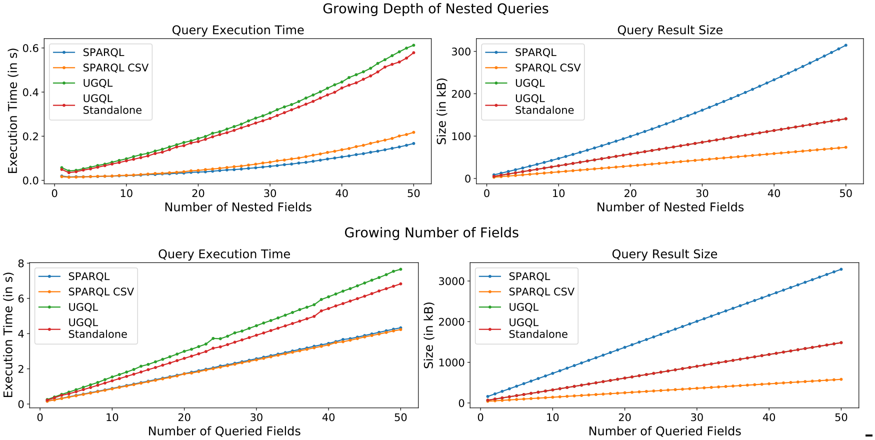
Query Translation Evaluation - Qualitative

Table: Qualitative comparison of UGQL, HGQL and SPARQL

Metric	**Query**	**UGQL**		**HGQL**	**SPARQL**	
		Standalone	Fuseki	Fuseki	CSV	JSON
Query Size	1	**64 B**		68 B	168 B	
	2	**78 B**		82 B	280 B	
	3	**187 B**		202 B	260 B	
	4	**155 B**		160 B	258 B	
Result Size	1	30.1 KB			**17.4 KB**	73.3 KB
	2	10.4 KB			**4.9 KB**	33.4 KB
	3	**5.7 KB**			8.1 KB	22.6 KB
	4	**2.5 MB**			4.7 MB	13 MB
Latency / Response Time	1	139 ms	180 ms	88.4 ms	**10 ms**	19 ms
	2	68.1 ms	103 ms	50.9 ms	**8 ms**	14.5 ms
	3	48.3 ms	64.3 ms	42.7 ms	**7.3 ms**	13.2 ms
	4	3,530 ms	4,640 ms	2,970 ms	**240 ms**	715 ms

- Smaller query sizes than SPARQL
- Result structured and without duplicate data
  - Even smaller results than SPARQL with increasing amount of duplicates

Query Translation Evaluation - Quantitative



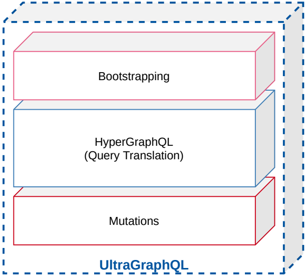
- Overhead depending on query and result size
  - Nested queries have higher impact on the overhead
  - Single digit overhead factor (Across all tests in the evaluation)

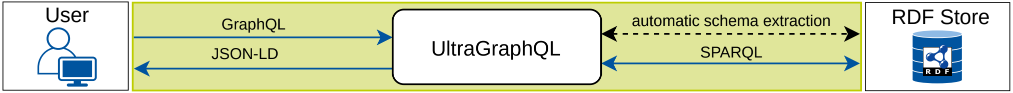
Evaluation - Summary

- Query resolving with a single digit overhead factor
  - Below the responsiveness threshold[8]
    - Except the cases where also SPARQL exceeded the threshold
  - Majority of overhead caused by internal response transformation
- Smaller query and result (formatted) size than SPARQL
  - Reduced data exchange
  - Optimal for mobile applications
- Mutations allow modifications on one service
  - Minimal overhead (around 2)
  - Schema preserving

Conclusion

- Automatic generation of
  - UGQLS for unknown datasets
    - Configurable bootstrapping phase
  - CRUD operations
- Provision of GraphQL endpoints for Linked Data
  - Acceptable overhead
  - Even on frequently changing triple stores
  - No prior knowledge of Semantic Web required
- Reduction of query and result size
  - Negation of the SPARQL criticism
  - Ideal for mobile applications
- Code available under **https://git.rwth-aachen.de/i5/ultragraphql**





Future Work & Outlook

- Improved response transformation **(Done)**
  - Performance improved by up to 70% (compared to HGQL)
  - Implemented in UGQL 1.1.0
- Schema altering mutations
  - Live schema updates at runtime
- Support of more primitive data types
- User study on improved accessibility to Linked Data through UGQL compared to SPARQL
  - Not conducted by the other GQL adapters

Questions?

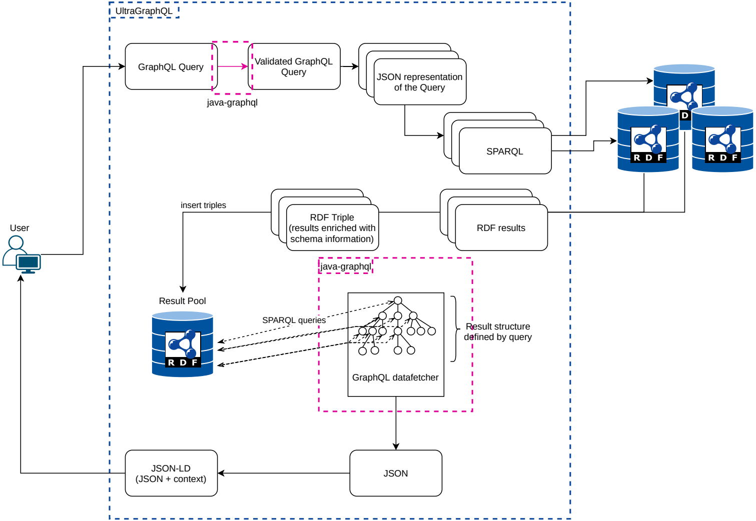
Paper

`https://hobbitdata.informatik.uni-leipzig.de/quweda/2020/quweda2020_paper_2.pdf`

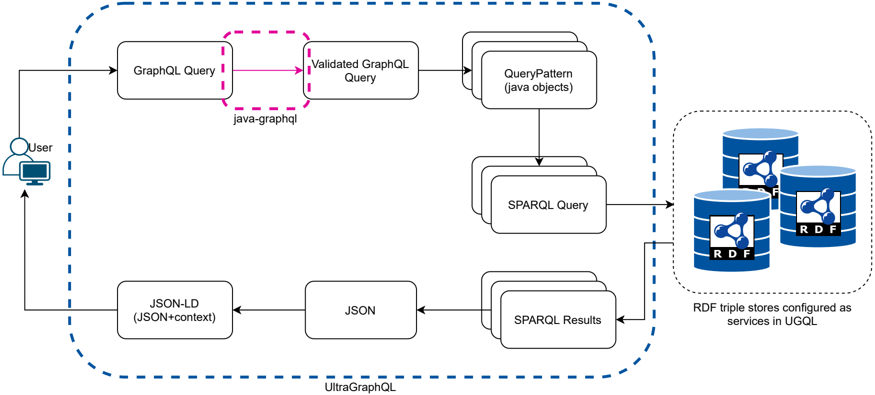
Code

`https://git.rwth-aachen.de/i5/ultragraphql`

Overhead Reasons



UGQL 1.1.0 - Improved Result Transformation

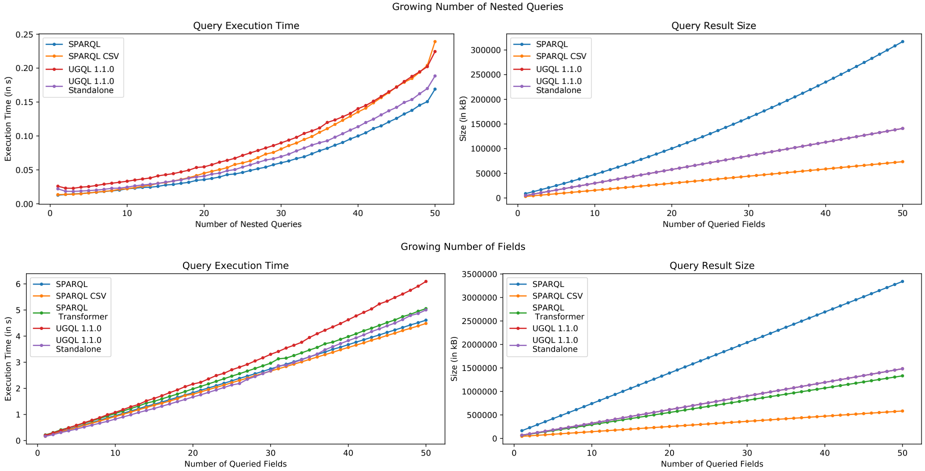


UGQL 1.1.0 - Qualitative Evaluation

Metric	Query	UGQL 1.1.0		UGQL 1.0.0		HGQL	SPARQL	
		Standalone	Fuseki	Standalone	Fuseki	Fuseki	CSV	JSON
Query Size	1	**64 B**				68 B	168 B	
	2	**78 B**				82 B	280 B	
	3	**187 B**				202 B	260 B	
	4	**155 B**				160 B	258 B	
Result Size	1	30.1 KB					**17.4 KB**	73.3 KB
	2	10.4 KB					**4.9 KB**	33.4 KB
	3	**5.7 KB**					8.1 KB	22.6 KB
	4	**2.5 MB**					4.7 MB	13 MB
Latency / Response Time	1	39.1 ms	26.0 ms	139 ms	180 ms	88.4 ms	**10 ms**	19 ms
	2	25.4 ms	18.7 ms	68.1 ms	103 ms	50.9 ms	**8 ms**	14.5 ms
	3	24.8 ms	17.8 ms	48.3 ms	64.3 ms	42.7 ms	**7.3 ms**	13.2 ms
	4	2,105 ms	1,108 ms	3,530 ms	4,640 ms	2,970 ms	**240 ms**	715 ms

Table: Qualitative comparison of UGQL 1.1.0, UGQL 1.0.0, HGQL and SPARQL

UGQL 1.1.0 - Quantitative Evaluation



Schema Mapping - Default Vocabulary Mapping

RDF	UGQL
Class	Object Type + Interface Type
Property	Field
Doamin	Domain(Object) of Field
Range	Output Type of Field
Literal	String
SubClassOf	Interface Type + implements
SubPropertyOf	Add Domain and Output Type of Sub-Property to Super-Property
EquivalentClass	Mutual implements + directive
EquivalentProperty	Merging Domain and Output Type of Both Fields + directive

Example Query

```
{
    ex_Person(offset: 2,  limit : 3){
        _id
        ex_name(limit:2)
        ex_relatedWith(_id:["http :// example.org/alice"]){
            ex_name
            ex_age
            ex_drives{
                ex_model(lang:"en")
            }
        }
    }
}
```

Query Translation - Example I

UltraGraphQL Schema

```
type ex_Person{
  _id: ID
  ex_name: [String]
  ex_age: [String]
  ex_address [ex_Address]
}

type ex_Address{
  _id: ID
  ex_street: [String]
}
```

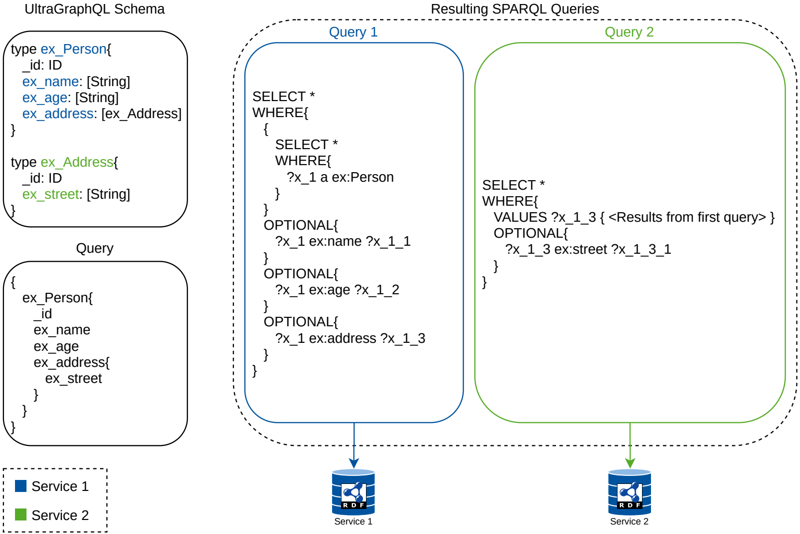
Query

```
{
  ex_Person{
    _id
    ex_name
    ex_age
    ex_address{
      ex_street
    }
  }
}
```

SPARQL Query

```
SELECT *
WHERE{
  {
    SELECT *
    WHERE{
      ?x_1 a ex:Person
    }
  }
  OPTIONAL{
    ?x_1 ex:name ?x_1_1
  }
  OPTIONAL{
    ?x_1 ex_age ?x_1_2
  }
  OPTIONAL{
    ?x_1 ex:address ?x_1_3
    OPTIONAL{
      ?x_1_3 ex:street ?x_1_3_1
    }
  }
}
```

Query Translation - Example II



UltraGraphQL Schema

```
type ex_Person{
  _id: ID
  ex_name: [String]
  ex_age: [String]
  ex_address: [ex_Address]
}

type ex_Address{
  _id: ID
  ex_street: [String]
}
```

Query

```
{
  ex_Person{
    _id
    ex_name
    ex_age
    ex_address{
      ex_street
    }
  }
}
```

🟦 Service 1

🟩 Service 2

Resulting SPARQL Queries

Query 1

```
SELECT *
WHERE{
  {
    SELECT *
    WHERE{
      ?x_1 a ex:Person
    }
  }
  OPTIONAL{
    ?x_1 ex:name ?x_1_1
  }
  OPTIONAL{
    ?x_1 ex:age ?x_1_2
  }
  OPTIONAL{
    ?x_1 ex:address ?x_1_3
  }
}
```

Query 2

```
SELECT *
WHERE{
  VALUES ?x_1_3 { <Results from first query> }
  OPTIONAL{
    ?x_1_3 ex:street ?x_1_3_1
  }
}
```

Query Translation - Example III

UltraGraphQL Schema

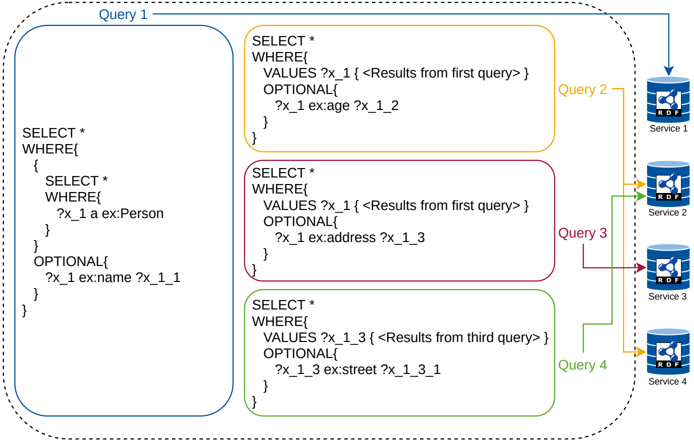
```
type ex_Person{
  _id: ID
  ex_name: [String]
  ex_age: [String]
  ex_address: [ex_Address]
}

type ex_Address{
  _id: ID
  ex_street: [String]
}
```

Query

```
{
  ex_Person{
    _id
    ex_name
    ex_age
    ex_address{
      ex_street
    }
  }
}
```

Resulting SPARQL Queries



Query 1

```
SELECT *
WHERE{
  {
    SELECT *
    WHERE{
      ?x_1 a ex:Person
    }
  }
  OPTIONAL{
    ?x_1 ex:name ?x_1_1
  }
}
```

Query 2

```
SELECT *
WHERE{
  VALUES ?x_1 { <Results from first query> }
  OPTIONAL{
    ?x_1 ex:age ?x_1_2
  }
}
```

Query 3

```
SELECT *
WHERE{
  VALUES ?x_1 { <Results from first query> }
  OPTIONAL{
    ?x_1 ex:address ?x_1_3
  }
}
```

Query 4

```
SELECT *
WHERE{
  VALUES ?x_1_3 { <Results from third query> }
  OPTIONAL{
    ?x_1_3 ex:street ?x_1_3_1
  }
}
```

🟦 Service 1
🟩 Service 2
🟥 Service 3
🟧 Service 2 & 4

Mutations

Insertion

```
mutation{
    insert_ex_Person(
        _id: "https :// example.org/Bob",
        ex_name: "Bob",
        ex_age: "42",
        ex_relatedWith: {
            _id: "https :// example.org/Alice"
        })
    {
        <SelectionSet>
    }
}
```

Resulting SPARQL Update

```
PREFIX ex: <https://example.org/>
INSERT DATA{
    GRAPH <...>{
        ex:Bob a ex:Person;
        ex:name "Bob";
        ex:age "42";
        ex:relatedWith ex:Alice  .
    }
}
```

- Generation of insert and delete mutations for all objects of the schema

Example Insertion

Insertion

```
mutation{
    insert_ex_Person(
        _id:  "https :// example.org/Bob",
        ex_name: "Bob",
        ex_age: "42",
        ex_relatedWith: {
            _id:  "https :// example.org/Alice"
        }
    ){
        _id
        ex_name
        ex_relatedWith{
            ex_name
        }
    }
}
```

Resulting SPARQL Update

```
PREFIX ex: <https://example.org/>
INSERT DATA{
    GRAPH <...>{
        ex:Bob a ex:Person;
                  ex:name "Bob";
                  ex:age "42";
                  ex:relatedWith ex:Alice  .
    }
}
```

Resulting SPARQL Query

```
PREFIX ex: <https://example.org/>
SELECT *
WHERE{
    {
        SELECT *
        WHERE{
            ?x_1 a ex:Person
        }
    }
    OPTIONAL{
        ?x_1 ex:name ?x_1_1
    }
    OPTIONAL{
        ?x_1 ex:relatedWith ?x_1_2
        OPTIONAL{
            ?x_1_2 ex:name ?x_1_2_1
        }
    }
```

RWTH AACHEN UNIVERSITY

Tim Holzheim
RWTH Aachen University
QuWeDa'20 @ ISWS
2020-11-01

Example Deletion I

Deletion by ID

```
mutation{
    delete_ex_Person(_id: "https :// example.org/Bob"){
        <SelectionSet>
    }
}
```

Resulting SPARQL Update

```
PREFIX ex: <https://example.org/>
WITH <...>
DELETE{
    ?person ?p1 ?o .
    ?s ?p2 ?person .
}
WHERE{
    ?person a ex:Person .
    OPTIONAL{
        ?person ?p1 ?o .
    }
    OPTIONAL{
        ?s ?p2 ?person .
    }
}
```

Example Deletion II

Deletion by Match

```
mutation{
    delete_ex_Person(
        ex_name: "Bob",
        ex_age: "42",
        ex_relatedWith:{
            _id:  "https :// example.org/Alice"
        }
    ){
        <SelectionSet>
}
```

Resulting SPARQL Update

```
PREFIX ex: <https://example.org/>
WITH <...>
DELETE{
    ?person ?p1 ?o .
    ?s ?p2 ?person .
}
WHERE{
    ?person a ex:Person;
              ex:name "Bob";
              ex:age "42";
              ex:relatedWith ex:Alice  .
    OPTIONAL{
        ?person ?p1 ?o .
    }
    OPTIONAL{
        ?s ?p2 ?person .
    }
}
```

Example Deletion III

Deletion of Data

```
mutation{
    delete_ex_Person(
        _id: "https :// example.org/Bob",
        ex_name: "Bob",
        ex_age: "42",
        ex_relatedWith: {
            _id: "https :// example.org/Alice"
        }
    ){
        <SelectionSet>
    }
}
```

Resulting SPARQL Update

```
PREFIX ex: <https://example.org/>
DELETE DATA{
    GRAPH <...>{
        ex:Bob a ex:Person;
                 ex:name "Bob";
                 ex:age "42";
                 ex:relatedWith ex:Alice  .
    }
}
```

References I

- John P. McCrae, "The Linked Open Data Cloud." `https://lod-cloud.net`. Accessed on: 2020-09-29.
- C. Farré, J. Varga, and R. Almar, "GraphQL Schema Generation for Data-Intensive Web APIs," in *Model and Data Engineering* (K.-D. Schewe and N. K. Singh, eds.), (Cham), pp. 184–194, Springer International Publishing, 2019.
- D. Chaves-Fraga, F. Priyatna, A. Alobaid, and O. Corcho, "Exploiting Declarative Mapping Rules for Generating GraphQL Servers with Morph-GraphQL," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 06, pp. 785–803, 2020.
- R. Taelman, M. Vander Sande, and R. Verborgh, "GraphQL-LD: Linked Data querying with GraphQL," in *Proceedings of the 17th International Semantic Web Conference*, pp. 1–4, October 2018.
- Stardog Union, "Stardog 7: The Manual." `https://www.stardog.com/docs/#_graphql_queries`, 2020.

References II

TopQuadrant, “Updating RDF Graphs with GraphQL.” https://www.topquadrant.com/technology/graphql/graphql-mutations/. Accessed: 2020-02-08.

S. I. Ltd., “HyperGraphQL Git Respository.” https://github.com/hypergraphql/hypergraphql, 2018. Accessed: 23.06.2020.

J. Nielsen, *Usability Engineering*. 1994.

The End