1 Introduction

Beyond doubt, the world wide web has become central to the business reality of companies and to the personal reality of the majority of people in their everyday lives. Large web communities like Facebook or YouTube make people stay on the internet for a very long time. Companies use at the very least e-mail for communication, while most of them also use other services on the web, e.g. for ordering material or booking business travel. While the rise of web services made several services on the web accessible for machines, the vast majority of content on the web is still made for and only usable by humans. For example, while obtaining information by reading a text or watching a video is natural for humans, a machine would have to use complex text parsing or even video and audio recognition techniques. Even after fulfilling this complex task, the machine would still lack required background knowledge a human might naturally have, e.g. who the person in a video is.

To make the content of the web more meaningful to machines, the idea of a semantic web and linked data was developed. In the semantic web, the content is connected in a machine readable way and delivered as metadata with a web resource. Additionally, background information is also modelled in a machine readable way, so the machine can connect the newly obtained information to information which is already known. To achieve that, a framework for modelling and connecting information is needed. Furthermore, a possibility to search and query the model is needed. A way to model and connect information is the resource description framework (RDF), and a language for obtaining information from RDF-modelled content is called an RDF query language. One such language standardized by the world wide web consortium (W3C) is SPARQL, which is an acronym for SPARQL Protocol And RDF Query Language. SPARQL and its usage will be the central topic of this paper.
1.1 Document Structure

In Section 2, a brief introduction to the topic of linked open data, its ideas and the necessary technologies is given. Following that, the basic structure and usage of RDF is described in Section 3. In the main section of this paper, Section 4, the syntactical and logical structure of SPARQL is presented. Section 5 then describes the usage of SPARQL with a few examples and real world applications. Section 6 closes with a short summary and an outlook on problems and further development of SPARQL.

2 Linked Open Data

The majority of resources on the web is made for human consumption in the form of, for example, HTML or different media types, like video and audio tracks. This being the case, a machine misses most of the structure and semantics of a resource, because neither the resources themselves nor connections like hyperlinks hold any information about their type, and thus about what they express.

To change this, the idea of linked data was developed. In the words of Berners-Lee et. al., linked data: “is simply about using the Web to create typed links between data from different sources.”[2] The concept is connected to the idea of a semantic web, which was also published and supported by Tim Berners-Lee[1]. The connection and typification of resources should of course happen in a machine processable way using standard technologies. The addition of the word open to the concept’s name underlines that the concept was developed for an open web accessible to everyone.

With the example of a video or audio resource, linked data can also be seen as metadata about the resource. The data linked to the resource could contain information about it, like things or persons visible, shown places and so on.

For guaranteeing the uniqueness of resources and connection types, Berners-Lee suggests the usage of Uniform Resource Identifiers (URI) in the “Four rules for Linked Data”, a set of rules for the usage of linked data[2]. With URIs, every describable resource, every type and every description can be uniquely addressed, a fact that is used by RDF, and later SPARQL.

The backbone of linked data is a standardized way of storing and connecting information and resources. The RDF was developed to fulfill these requirements and will be discussed in the following section.

3 Resource Description Framework

Before one can even think about querying information or having a semantic web, a language for describing and connecting information on the world wide web is needed. A XML-based approach to this is RDF, which is a recommendation by the W3C. The following section will give a brief introduction to the basic ideas of RDF.

The RDF Primer by the W3C describes RDF as follows:
The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. [...] However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web. [...]” [5]

Unlike, for example, the hyper text mark-up language (HTML), which is primarily designed for presenting information to humans, RDF is designed to store and present information about a resource in a machine readable way. Contrary to the metadata tag of HTML, which has a similar intention, RDF data preserves its meaning by storing the type of information in a standardized format alongside the information itself. This meaning is also a resource identified by an URI, e.g. http://www.w3.org/2000/10/swap/pim/contact#Person for determining something as a person.

The base idea behind the information connection in RDF is that something (the resource) can have properties, which in turn have values. In the RDF Primer by the W3C, a terminology of subject, predicate and object is introduced. The subject is the resource or thing that will be described, the predicate is the property that will be specified and the object is the value of that property. To keep this structure machine readable, subjects, predicates and objects are given as URIs. However, not every subject, predicate or object has to be another resource referred to by an URI. For example, constant values (which are also called literals) can only be objects, while the predicate always has to be a resource which describes the kind of information attached to the object (as in the “person” example above). There are several ways of displaying or storing RDF, e.g. as a graph or in its XML representation. Furthermore, it can be written in triples: `<subject-URI> <predicate-URI> <object-URI>` [5].

Figure 3.1 shows three representations - or views - on an RDF knowledge base. Car8231 represents a real world resource. The first line of the triple representation
specifies its property Type as a value Mustang68. In the next line, Mustang68 is the resource with its property Manufacturer being described. With all of this information, connected knowledge is created, which can be regarded as a graph. This is shown in Figure 3.1(c).

For navigating or retrieving information from RDF data in a comfortable way, a query language is needed. One possible choice is SPARQL, which will be discussed in the following section.

4 SPARQL

This section describes a query language for RDF. The general approach regarding the representations of RDF will be discussed, e.g. which of the formats or views of RDF discussed above will be queried. Then, an explanation of the basic principles of SPARQL will follow, leading to the construction of complex queries.

The syntax and semantics presented here are described in the SPARQL recommendation of the W3C by Prud’hommeaux and Seaborne[6].

4.1 General Approach

When thinking of querying RDF, the first step is to decide on which RDF representation or representations should be used for querying. One could think of using the XML representation, since the problem of finding information within an XML document is well known and already has several solutions, e.g. X-Path1.

Another possibility would be using the triple representation presented in Section 3. The naive approach here would be to perform a full text search, something that is very ineffective and inferior to techniques like navigating via X-Path. One would rather have a more structured way of accessing information stored in triples.

To solve that, the graph view on RDF information was taken as a model of thought. With this, querying RDF is possible by matching graph patterns against an RDF graph.

Of course, there are other approaches for querying RDF structured data, such as the RDF Data Query Language (RDQL) or the Second Generation RDF Query Language (SeRQL). These, however, are outside the scope of this paper.

4.2 Graph Patterns

When thinking of graph patterns, one must first think of a textual representation for them. Since RDF can already be represented by a triple structure which can be viewed as a graph, a similar approach can be chosen to define graph patterns. Thus, triple patterns are introduced. The difference between a triple and a triple pattern is the usage of variables in triple patterns, which may not appear in triples. Any part - resource, property, or value - of a triple can be chosen as a variable, creating

1http://www.w3.org/TR/xpath
Figure 4.1: Graph pattern matching example

{A B ?x . ?x C ?y .} ⇔ {A B ?x .}{?x C ?y .}

Figure 4.2: Group pattern example

A pattern where the non-variable values of the triple pattern are matched against the data. The data then covered by the variables can be part of a result.

Figure 4.1 shows an example of graph pattern matching with the green nodes being the available data and the yellow nodes being the pattern. The pattern is created by a triple pattern like

A B ?variable .

Matching this pattern with our data would return ?variable = C and ?variable = D, since resource and property match in both cases.

To create more complex patterns which go further than simple triple patterns, several possibilities of connecting and manipulating patterns exist. The first option for connecting triple patterns is by using basic graph patterns (BGP). Syntactically, BGPs are a sequence of triple patterns with a full stop after each triple pattern. Semantically, a BGP is regarded as the conjunction of every included triple pattern.

The next level of connection is given by group graph patterns, or, in short, group patterns. Contrary to BGPs, which can be constructed out of triple patterns, group patterns are a set of any graph patterns, like BGPs or other group patterns. Syntactically, group patterns are indicated by curly braces {} around the grouped patterns. As with BGPs, a conjunction of the patterns within a group pattern is implicated, making a logical AND operator obsolete. Figure 4.2 is an example for group patterns which shows two patterns that are semantically equivalent.

The results of SPARQL statements depend on the used query type, but in the common case, the results are bindings to the pattern variables for every matching pattern as seen in the example of Figure 4.1. To refine the patterns created from this, SPARQL introduces several keywords, each giving different manipulation opportunities.

The value facet of a variable can be restricted by the usage of the FILTER keyword in combination with the respective variable. A group pattern can contain any number of FILTER expressions. Variables can be filtered by arithmetical operators such as equal, less, more than, or any combination, such as less-or-equal. Further-
more, string filters can be applied, which consist of regular expressions. The regular expression evaluation of SPARQL is based on the respective part of *XPath*.

Since patterns are implicitly conjuncted, a keyword is needed to express if the match of a pattern is not mandatory for a result. If a group pattern is optional for the result, it can be marked with the **OPTIONAL** keyword. With this, the result of that pattern is added to the total result of the containing group pattern if it matched. If not, only the content of the optional group pattern is excluded from the result, but not the whole pattern. This is especially useful if the knowledge base is not complete or if the user is not sure which information is present.

If seen as cardinalities, the **OPTIONAL** keyword expresses a cardinality of 0 to 1 of the respective pattern. If a cardinality of 1 to \( n \) (with \( n \in \mathbb{N} \)) is to be expressed, the keyword of choice is **UNION**. While being described as alternative-mechanism in the standard, the keyword **UNION** was chosen over, for example, **ALTERNATIVE**. Syntactically, the keyword is written in-between every two group pattern which shall be included in the 1 to \( n \) relation. A result is returned if any of the graph patterns connected by **UNION** matches. If more than one matches, all results of matching patterns are returned.

### 4.3 Queries

To build syntactically correct queries from the patterns described above, some additional keywords for a query structure are needed. The structure of the most common query type is borrowed from the database query language SQL. The variables which shall be included in the result are specified after the **SELECT** keyword. If all bound variables are wanted, one uses **SELECT \***. The patterns which are matched to create the result are given after the **SELECT**, starting with the **WHERE** keyword.

Since RDF knows namespaces like XML, it is additionally required to add the namespace to the identifiers used in the patterns. Similar to XML, a prefix mechanism is introduced in SPARQL for better readability. The namespaces have to be included in every query and are specified at the beginning of a query, using the **PREFIX** keyword. The structure of a prefix definition is:

```
PREFIX <prefix-identifier> <prefix-uri>
```

Unlike SQL, SPARQL offers query types different than **SELECT**-queries. The **CONSTRUCT** query type is used to create a new RDF graph from the data given by the patterns in the **WHERE** statement. The structure of the new graph is given as a group pattern after the **CONSTRUCT** keyword. This can be useful to rearrange data for further processing, or to create new connections among the extracted information.

Another, more simple query type is the **ASK** query. It matches the pattern given after **ASK** with the graph and returns **yes** if it matches at least once, and **no** otherwise.

Similar to SQL, the result of a query can be manipulated after the patterns themselves were already matched. The first option for this is the **ORDER BY** keyword,
which orders the results by one or more variables. The order can be ascending (ASC) or descending (DESC).

To exclude duplicates from the result, one uses the DISTINCT keyword after SELECT. The SPARQL standard also specifies the REDUCED keyword, which permits the query processor to remove any number of duplicates (even 0) from the result, without the possibility to further specify how many duplicates should be excluded.

If the query user wants to display a large result step by step or simply wants to have a limited result set, the LIMIT keyword can be used. A positive integer after the keyword specifies how many results are returned. With this, equal queries can return different results if executed on the same data. This is because no order is defined on the result. To avoid this, the ORDER BY keyword should be used. Especially when combined with the OFFSET keyword, which skips the first $n$ results, if $n$ is a positive integer given after the keyword.

5 Usage and Examples

After presenting the basics of query construction in Section 4, this section will cover the usage of SPARQL. In the first part, a few examples on a simple dataset will be presented. In the second half, an outline of some real world applications and their SPARQL background will be shown.

5.1 Query Examples

For the following examples, the dataset shown in Figure 5.1 will be used. The namespace prefixes given there represent a fictional car namespace (cars) with the URI http://example.com/cars and the GoodRelations (gr) namespace of purl.org, which describes companies and product relations.

The first example shown in Figure 5.2 features a basic query, using the basic query structure and a group pattern. It will give the manufacturer of a certain type of car, the cars:Mustang68.

Although the results of a SELECT query are bindings to the given variables, the most common way of displaying SPARQL results is a table. Thus, the result of Figure 5.2 is shown in a table in Figure 5.3.

```sparql
cars:Car8231 gr:Type cars:Mustang68 .
cars:Car4642 gr:Type cars:Camaro .
cars:Mustang68 gr:hasManufacturer cars:Ford .
cars:Car8231 cars:LicensePlate "Lic42" .
cars:Camaro gr:hasManufacturer cars:GM
```

Figure 5.1: An example RDF dataset in triple representation
PREFIX cars: <http://example.com/cars>
PREFIX gr: <http://purl.org/goodrelations/v1>
SELECT ?name
WHERE { cars:mustang68 gr:hasManufacturer ?name . }

Figure 5.2: A basic query

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cars:Ford</td>
</tr>
</tbody>
</table>

Figure 5.3: The result of the query in Figure 5.2

Figure 5.4 shows a more complex query, using the OPTIONAL and UNION keywords. It will return the types of cars constructed by Ford or General Motors and, if present, the license plates of all registered cars of that types. Since the actual car itself is not of interest, a blank node is used to create the connection but ignore the value.

The result of that query is shown in Figure 5.5. Note that the type cars:Camaro is included in the result, although no license plate was specified. The variable is simply unset.

5.2 Real World Usage

Although SPARQL is a rather young technology, there are several real world examples for its usage. One example is the usage of RDF for a triple store, creating a schema-less database. Having this triple store, SPARQL can be used to query it, like SQL would be used to query relational databases. An advantage of this over relational databases is that it can be used without knowledge about the stored data, since triple patterns with only variables can be used. Whether this advantage is a weighty argument for the usage of SPARQL over SQL is doubtful, though, since the schema of a database is known in the common case. A commercial product using

PREFIX cars: <http://example.com/cars>
PREFIX gr: <http://purl.org/goodrelations/v1>
SELECT ?car, ?license
WHERE {
  { ?car gr:hasManufacturer "Ford" . }
  UNION { ?car gr:hasManufacturer "GM" . }
  OPTIONAL { _:x gr:Type ?car . _:x cars:LicensePlate ?license } }

Figure 5.4: A query using OPTIONAL and UNION
this approach is AllegroGraph\footnote{\url{http://www.franz.com/agraph/allegrograph}}.

Present RDF knowledge bases can also be accessed over the web. To standardize the way this is done, the concept of SPARQL endpoints was introduced. SPARQL endpoints are web services which accept SPARQL requests and match them against their underlying knowledge base. The endpoints are also available to humans, although the interface of the most prominent endpoints is a simple text editor. Furthermore, the results are presented as tables, as shown in the examples above. Thus, only experts can use SPARQL endpoints directly, requiring more user friendly interfaces for offering endpoints to end-users. Many ontologies from linked open data offer endpoints, such as BioGateway, data.gov, or DBPedia.

DBPedia is also a good example for a large existing knowledge base storing its data in RDF. This project, started by the Free University of Berlin, the University of Leipzig and OpenLink Software, has the goal of storing all structured information present in Wikipedia in RDF. Structured information are present in several Wikipedia articles and are often displayed in a small box at the beginning of an article. As of September 2011, the DBPedia knowledge base contains about one billion RDF triples\footnote{\url{http://dbpedia.org/about}}.

Due to its wide range of domains, data from DBPedia is often referred to from other Linked Open Data hubs, giving DBPedia a central role in a web of connected data\footnote{\url{http://dbpedia.org/about}}.

Another application using SPARQL and linked data is RelFinder\footnote{\url{http://www.visualdataweb.org/relfinder.php}}. It uses SPARQL queries and the DBPedia knowledge base to find connections that exist between any two or more resources present in the DBPedia data set. To achieve that, the resources which are to be investigated are used as endpoints of patterns, with paths in the RDF graph discovered between them. Let \( r_1 \) and \( r_2 \) be the investigated resources and \( n \in \mathbb{N}\setminus\{0\} \), then such a pattern path could be constructed like this:

\[
\begin{align*}
\quad r_1 \ ?\text{prop1} \ ?\text{val1} . \ ?\text{val1} \ ?\text{prop2} \ ?\text{val2} . \ldots \ r_2 \ ?\text{propn} \ ?\text{valn}
\end{align*}
\]

The query construction of RelFinder uses a similar approach, but adds a little more complexity for better results. For example, several \textsc{filter} statements are added to the query for excluding properties and values which are redundant or not wanted in the result.

The RelFinder web application is more comfortable to use for an end-user than the SPARQL endpoints mentioned above. The resources to investigate are entered
in text fields with auto-completion, and the results are presented as an animated graph that is expanded as new results are discovered over time.

There are a lot more real world applications using SPARQL and RDF which go beyond this short overview. The W3C presents a list of SPARQL endpoints, which can be used as a starting point for exploring these applications.

6 Conclusion

A short introduction to the concepts of linked open data and semantic web were given, stating the problems and the technologies that would be needed to solve those problems. Following that, the resource description framework was presented, which offers a standard for storing structured information. The SPARQL query language for RDF was presented, showing basic concepts, syntax, semantics, and several rules and hints for query construction. Afterwards, the usage of SPARQL was demonstrated with some small example queries on a small data set. Some real world application of the presented technologies were shown to give a better idea of the practical use of SPARQL.

6.1 Problems and Future Work

Since RDF, SPARQL and other linked data technologies are rather young, especially in comparison to other database technologies such as relational databases and SQL, several unsolved problems and many ideas for improvement exist.

A major problem in handling and querying knowledge bases is their sheer size. The one billion triples of DBpedia were mentioned above, querying them will lead to large runtimes. Schmidt et. al. benchmarked the performance of several SPARQL engines and discovered deficits and optimization potential which might be considered in a future update of SPARQL.

Schmidt et. al. investigated SPARQL and subsets of the language regarding their complexity, discovering that pattern matching even with small subsets of the language is NP-complete.

Another approach for a runtime reduction could be the partitioning of the data space and the usage of parallel or massively parallel execution models and hardware, such as CUDA or FPGAs. Groppe and Groppe developed a parallel query execution for multi-core CPUs, discovering that the overhead of parallelisation let their approach only outrun the basic approach for large data sets.

Beside the performance issues, future work is open for the benefit of end-users and a general improvement of the usage of SPARQL. In particular, tools for the generation of queries and the display of results are needed. Russell and Smart tried to solve this by implementing a graphical query editor. The vision of a distant

5http://www.w3.org/wiki/SparqlEndpoints
6http://www.nvidia.com/object/cuda_home_new.html
7http://en.wikipedia.org/wiki/Field-programmable_gate_array
future could be a speech processor which generates queries from natural language and synthesizes answers from query results.

7 References


