

Visual SPARQL Querying based on Extended Filter/Flow Graphs

Florian Haag Steffen Lohmann Steffen Bold Thomas Ertl
Institute for Visualization and Interactive Systems, University of Stuttgart
Universitätsstraße 38 · 70569 Stuttgart · Germany
{florian.haag, steffen.lohmann, thomas.ertl}@vis.uni-stuttgart.de

ABSTRACT

SPARQL is currently the major query language for the Semantic Web. However, writing SPARQL queries is not an easy task and requires some understanding of technologies like RDF. In order to enable users without this knowledge to query linked data, visual interfaces are required that hide the SPARQL syntax and provide graphical support for query building. Based on the concept of extended filter/flow graphs, we present a novel approach for visual querying that addresses the unique specifics of SPARQL and RDF. In particular, it enables the creation of *SELECT* and *ASK* queries, though it can also be used for other query forms. In contrast to related work, the users do not need to provide any structured text input but can create the queries entirely with graphical elements. Our approach supports most features of SPARQL and hence also the construction of complex query expressions. It has been implemented in a visual querying framework and tested on different RDF datasets, including DBpedia that is used as an example in this paper. Since the filter/flow concept is empirically well-founded, we expect our approach to be very usable, which is additionally supported by the results of a qualitative user study we conducted.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—*information filtering, query formulation*; H.5.2 [Information Interfaces and Presentation]: User Interfaces—*graphical user interfaces (GUI)*; I.3.6 [Computer Graphics]: Methodology and Techniques—*interaction techniques*

General Terms

Human Factors

Keywords

Visual querying, filter/flow, semantic web, query language, visualization, linked data, RDF, SPARQL, OWL, faceted search

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
AVI '14 May 27 - 29 2014, Como, Italy
Copyright is held by the authors. Publication rights licensed to ACM.
ACM 978-1-4503-2775-6/14/05...\$15.00.
<http://dx.doi.org/10.1145/2598153.2598185>.

1. INTRODUCTION

Since being released as an official W3C recommendation in early 2008 [28], SPARQL has evolved into the major query language for the Semantic Web. It has become the de facto standard for querying semantic triple stores and is a key technology of the open data movement [1, 9]. SPARQL is supported by nearly all modern RDF-based storage systems and used within enterprise and public web contexts [10]. Many publicly available datasets provide SPARQL endpoints nowadays [4, 9]. Version 1.1 of SPARQL, released in March 2013, contains several new features that further advance the power of SPARQL, such as aggregate functions and support for subqueries [31].

However, writing SPARQL queries can be a tedious task. The syntax is prone to errors and requires some understanding of RDF. As average users cannot be expected to write textual query expressions, user interfaces are needed that hide the concrete SPARQL syntax from the users. These interfaces must provide means to freely create SPARQL queries without any in-depth knowledge of RDF and other Semantic Web technologies.

Although related attempts have been made in the context of relational databases and SQL, those approaches can only partly be reused, as the data in relational databases is organized in fixed table structures. In contrast, SPARQL is a graph-based language designed for querying across different data sources on the web. Appropriate solutions must therefore address the unique specifics of SPARQL and RDF, such as the schema-independent description of resources and the use of URIs for global identification.

We present an approach for visual SPARQL querying based on the concept of extended filter/flow graphs [15]. In contrast to related work, the users do not need to provide any structured text input but can build the SPARQL queries entirely with graphical elements and simple text strings. In particular, our approach supports the creation of *SELECT* and *ASK* queries, but it may also be used for *CONSTRUCT* and *DESCRIBE* queries with only little variation.

2. RELATED WORK

Several paradigms of visual querying have been proposed and successfully implemented in the past [12]. In the following, we provide a short overview on the relevant paradigms and list works that apply them to SPARQL querying.

2.1 Form-based Querying

A popular paradigm is form-based querying, where query expressions are created from several text fields. For instance, SPARQLViz [11] provides a form-based wizard that guides the users through the query building process. In a first step, they select the query form, followed by the prefix bindings. Then, they add variables, filters and values, and finally any additional properties.

To appear in:

Proceedings of the 12th International Working Conference on Advanced Visual Interfaces (AVI '14)
New York, NY, USA: ACM, 2014.

Other examples are the Graph Pattern Builder of the DBpedia project [6] and Konduit VQB [5], both of which create SPARQL queries by following the RDF triple syntax. Users have to enter or select variables, identifiers or filters for the subject, predicate and object of each RDF triple in text fields or selection lists, which are grouped to simplify specifying multiple statements for the same subject.

Although these approaches support the users in query building, they still represent the queries in a way that is closely related to the SPARQL syntax. Thus, they do not relieve the users from the need to know how SPARQL queries are structured.

2.2 Querying by Browsing

Another paradigm can be found in visual interfaces that implicitly create the queries in the browsing process. Examples of browsers for RDF data are mspace [30], /facet [20], and gFacet [19]. Despite the fact that these browsers are very usable, they have been designed with a particular class of queries in mind and therefore limit the expressiveness of queries compared to generic approaches. All queries are a result of navigating through the RDF graph, and hence the user's degrees of freedom in query building depend on the flexibility of the browsing interface.

Similar constraints are present in visual approaches that create SPARQL queries for specific purposes, such as relationship discovery [24] or the querying of geographic data [8]. While they often provide well-designed user interfaces, the SPARQL queries follow a fixed structure and can only be configured but not fundamentally changed by the user.

2.3 Visual Query Languages

Visual query languages provide more degrees of freedom by supporting the query building with geometric primitives that represent the elements of the query language. There are several works that apply visual query languages to SPARQL, such as NITELIGHT [29], iSPARQL [2], RDF-GL [21], or the visual SPARQL editor presented by Groppe et al. [14]. The graphical elements are typically connected by edges to form simple node-link diagrams that describe the query expressions.

A slightly higher degree of abstraction is provided by visual interfaces based on UML-like diagrams. One such approach is presented by Barzdins et al. [7], who let the users create SPARQL queries in a UML profile for RDF and OWL. While such visual query languages help to lower the barrier of writing correct queries, they still require some knowledge of the query language, its structure and syntax.

Other approaches use visual pipes to represent the flow of data, in a similar way as flow charts. Two such attempts are DERI Pipes [25] and MashQL [22]. They are both inspired by the mashup framework Yahoo! Pipes [3], which has received a lot of attention also outside of academia. However, these mashup frameworks are not visual query languages in the narrow sense: While they do provide nodes for filtering data, their overall focus is rather on rearranging, sorting and transforming data. Filter restrictions are still displayed in the form of triples and require some knowledge of RDF and SPARQL.

3. FILTER/FLOW MODEL

The filter/flow model is similar to the aforementioned pipeline approaches of Yahoo! Pipes, DERI pipes and MashQL. In fact, it was developed long before these works by Young and Shneiderman in the context of SQL querying [33]. In contrast to the pipeline approaches, it does not address data transformation but has been developed purely for visual querying.

Several extensions to the original concept of Young and Shneiderman have been proposed over the years, which have led to our development of an extended filter/flow model that incorporates the most common modifications [15]. This extended model is very suitable to serve as a basis for visual SPARQL querying, as we will show in this paper. Before that, we will briefly introduce the basic and the extended filter/flow model in this section.

3.1 Basic Filter/Flow Model

The basic filter/flow model describes a visual representation of Boolean expressions that can be used for data filtering. The expressions are depicted as directed acyclic graphs, where the nodes define filter criteria and the edges represent the flow of data. This flow is modeled analogously to circuit diagrams, that is, conjunctions are represented by sequential paths while disjunctions form parallel paths. Figure 1 sketches an example of a filter/flow graph.

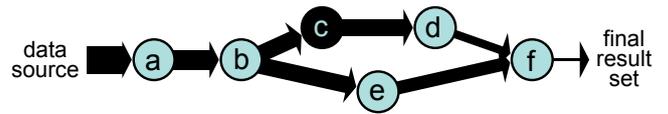


Figure 1: Sketch of a filter/flow graph representing the Boolean expression $a \wedge b \wedge ((\neg c \wedge d) \vee e) \wedge f$. The nodes define filter functions, while the edges depict the flow of data. The flow thickness represents the number of items that have passed the preceding filter nodes.

The filtering starts in the first node(s) of the graph (node a in Figure 1). Conceptually, these first node(s) retrieve the complete set of items from the data source. The final result set contains all data items that have traversed the whole graph on at least one path without being blocked by the filter nodes. The thickness of the flows represents the number of items that passed the preceding filter nodes. Thus, flows get noticeably thinner at nodes that filter out a large number of items (e.g. node d in Figure 1).

An extension of the basic filter/flow model is to allow more than one outbound edge at the “end” of the graph (node f in Figure 1). That way, several result sets based on different criteria can be returned in a single filter/flow graph. Likewise, intermediate results could be displayed for any of the edges in the graph, as it is done in FindFlow [18]. Further extensions include the coloring and free positioning of filter nodes [13, 18]. There are also some adaptations of the concept specific to the domain of geographic information systems [26] or tabletop display environments [23, 32].

3.2 Extended Filter/Flow Model

Based upon the proposed improvements, we have formally defined an extended filter/flow model [15]. In that model, the filter nodes have one or more *receptors* and *emitters* serving as the connection points for flows. This allows the filter nodes to receive data from several inbound flows that can be processed in different ways. Likewise, there can be several outbound flows, each representing a different filter function. The general structure of an exemplary filter node in the extended model is shown in Figure 2.

Filter nodes do not necessarily represent atomic filter functions in the extended model but can incorporate several filtering parameters into one node. Special node types are used to display the result of filtering. They can be added at any point in the graph, thus enabling not only the display of the final result set but also of intermediate results.

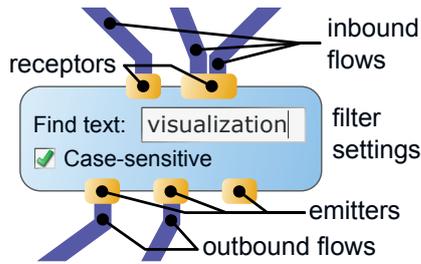


Figure 2: A node in the extended filter/flow model can have several receptors and emitters that serve as connection points for the flows [15].

The overall size and complexity of the filter/flow graphs is reduced in the extended model, with positive effects on the readability. The extended filter/flow graphs were read faster and were preferred over the basic ones by the participants of a user study we conducted [16].

4. FILTER/FLOW GRAPHS FOR SPARQL

Our approach for visual SPARQL querying is based upon the extended filter/flow model. It provides a representation of the filter restrictions rather than a representation of the complete query. This means that the visualization can be used to display combinations of filters in a way that does not require users to know anything about the SPARQL syntax or the structure of a SPARQL query. Users do not have to specify all query-related information, such as the projected variables of a `SELECT` query. Instead, this information is supplied automatically where needed by nodes that display intermediate or final results. Thus, once the desired restrictions have been defined in a given filter/flow graph, users can add nodes that display the result to apply those restrictions in `SELECT` queries (for displaying the resulting items), in `ASK` queries (for displaying the information whether there are any results) or possibly in other query forms such as `CONSTRUCT` queries (though we have not looked into how to graphically specify the graph templates required for that query form)—or even several at a time.

The filter criteria that can be expressed with our approach consider a variety of properties of the RDF data. All of these are directly built upon the filter/flow concept so as to be intuitively comprehensible to users. Figure 3 shows an exemplary SPARQL query represented with our approach, filtering a set of island based on various criteria. DBpedia [6] is used as the data source in all examples, i.e. the topmost node receives the data items from the DBpedia dataset. The numbers of data items that reach the final result sets are displayed in the bottommost node of the graph. The single elements that the filter/flow graph consists of will be explained step by step in the following paragraphs.

4.1 Filter/Flow Graph Elements

The basic group of filters performs comparisons of data values and excludes data items from the result set that do not meet the comparison criteria. These comparisons refer to the data stored in the RDF graph, mostly by indicating a property path that is evaluated relatively to each data item. Filters in this group include comparisons of ordinal values (such as numbers or dates, by operators like equality, greater or less than), strings as well as resource identifiers (URIs). Examples for ordinal comparisons are the filters for the *area* and *population* properties in Figure 3. String comparisons are applied by the first filter that checks whether the label of the *lo-*

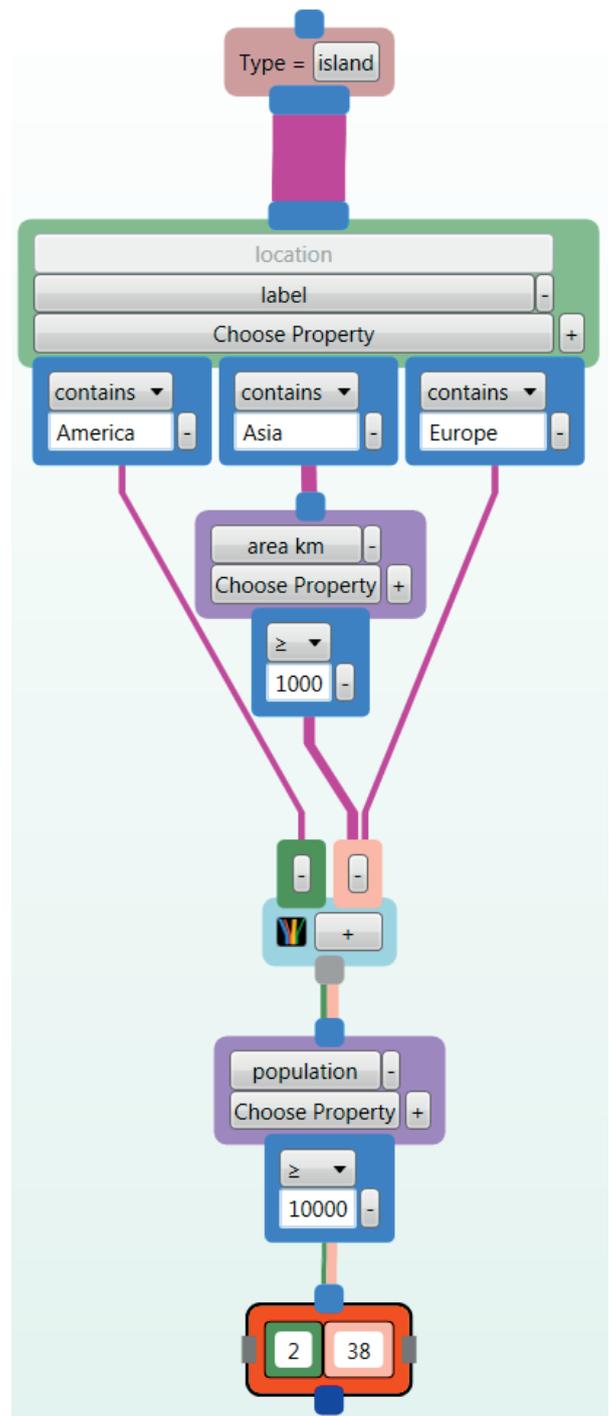


Figure 3: An exemplary SPARQL query expressed with our filter/flow-based approach. Data about islands provided by DBpedia is filtered based on various criteria. Several result sets can be displayed in one flow by using different colors, as shown in the bottom half of the graph.

cation property contains one of three strings. Besides comparisons of actual string values, the filters can also refer to other attributes of a literal, such as its language tag or the number of characters it consists of.

The second group of filters does not only help to analyze the data stored in the RDF graph, but also the graph structure itself. Thus, the concept takes advantage of an inherent feature of linked data and SPARQL in particular that is not usually found in most other data structures such as relational databases. In linked data, there is no separate schema information required to retrieve data from an RDF graph; the actual schema information—what properties do actually exist for a given resource, how many values are there for each of them—is implicitly contained in the results returned on any SPARQL query. Like this, in the same way as defining criteria that the data values in the RDF graph must meet, criteria that must be met by the underlying schema can be defined. Therefore, we have provided filters that check for the existence of a given property. Some planned additions to this group of filters are a node that determines which ontologies the properties defined for a given data item belong to, and a node that imposes numerical constraints on the number of values that can be found by following a given property path.

The third and last group of filters helps to arrange the flow of data through the filter/flow graph. Aside from mere layout-based helper nodes (that can bundle and thus unite flows before they actually arrive at their destination node, in order to reduce visual clutter), this relates primarily to the additional feature of our approach that permits running several distinct result sets through a single flow. Instead of interpreting each flow between an emitter of one filter node and a receptor of another as exactly one set of data items, we have defined flows in a way so they can represent any number of data sets.

Visually, a flow made up of more than one set of items is shown as consisting of several parallel differently colored lines that follow the path of the flow (lower half of Figure 3). Logically, each of these sets is processed separately. Like this, several sets that essentially run through the same filter nodes—but, for example, are distinguished by different filters in the very beginning of the graph, or taken from two different data sources—can be compared to each other. Nodes that deal with the whole flow of data—for example, to show intermediate results—consequently display the individuals sets of data items separately (cf. the bottom node in Figure 3 showing the size of the result sets). The only exception to this are some elements from the third group of filter nodes, which can break up a single flow of data into a flow with several sets of data based on arbitrary criteria, or which can combine several flows into a single flow with a separate sets (cf. the node with the icon, located in the middle of Figure 3).

By closely following the original filter/flow idea, we have incorporated the feature to use the flow thickness as an indicator of how many items there are in an intermediate result set at a given point in the filter graph. This helps users determine whether—based on the currently available data—a given filter node has a significant effect on the dataset (if the thickness of the outbound flows is visibly reduced compared to the inbound flows) and whether a given filter path needs to be followed no further as the dataset is empty anyway. This aspect of the visualization is directly applied to the aforementioned flows that represent several sets of data, as the thickness of the colored parallel lines indicates the size of the respective set of data (cf. Figure 3).

4.2 Structural Considerations

Besides this general idea of the components and features of our filter/flow graphs, we would like to highlight some properties that were included specifically to make query construction and comprehension easier for users.

The restrictions in SPARQL queries are provided as triple patterns that form a graph. The components of the triples can be further restricted by applying filter conditions expressed in separate `FILTER` clauses. In order to maintain the filter/flow principle of showing the combination of all restrictions in one coherent graph, our approach treats restrictions expressed as triple patterns and restrictions expressed within a SPARQL `FILTER` clause the same way. Thus, conjunctions and disjunctions of arbitrary restrictions can be combined by users, while the underlying SPARQL generation algorithm makes sure restrictions are grouped in `FILTER` clauses where appropriate. This can be observed in the following SPARQL text query, which is generated to determine the size of the result set displayed on the right-hand side in the bottommost node in Figure 3:

```

1 PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
2 PREFIX dbpprop: <http://dbpedia.org/property/>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4
5 SELECT DISTINCT (COUNT(DISTINCT ?a) AS ?f)
6 WHERE {
7   ?a dbpedia-owl:Island.
8   ?a dbpprop:population ?b.
9   FILTER(?b >= 10000).
10  {
11    ?a dbpprop:location/rdfs:label ?c.
12    ?a dbpprop:areaKm ?d.
13    FILTER(contains(ucase(?c), ucase("Asia")))
14    && ?d >= 1000).
15  } UNION {
16    ?a dbpprop:location/rdfs:label ?e.
17    FILTER(contains(ucase(?e), ucase("Europe"))).
18  }.
19 }

```

Like in many programming and query languages, SPARQL defines some frequently used functions as operators that can be written in infix notation, while other functions can be called only in prefix notation with an argument list. Although that distinction is useful in textual source code, it has no bearing for users who see only a visual representation of the queries. Consequently, our approach does not distinguish between mere operators (e.g., comparing the equality of two values) and function calls (e.g., retrieving the length of a string for a comparison).

As our approach is based on the extended filter/flow model, some filter nodes can be rather complex. This applies to generic comparison nodes that provide for a high degree of flexibility. On the other hand, in the case of frequently applied restrictions, users would have to configure all the settings in such a generic node over and over again. Therefore, such repeatedly used restrictions are integrated in predefined filter nodes statically set to, for example, a particular property (e.g., the *type* restriction in Figure 3).

The filter/flow concept in its original form and in various of its later extensions was used to filter a set of items based on restrictions on their scalar attribute values. For example, Young and Shneiderman's original work filtered employees based on attributes such as their title and salary [33], while Facet-Streams uses single-value attributes, such as the number of stars, the rating or the price per night, to filter hotel offers [23]. The filter/flow visualization is fairly straightforward, as the flow represents the set of items, and property values are directly accessible on each item in that set. This gets more complicated once a given property can refer to any number of complex objects, as it is the case in RDF graphs. A filter/flow graph that uses two filter nodes to impose two restrictions on authors of a book (cf. Figure 4a) will retrieve all books that have at least one author satisfying one of the restrictions, and at least one author satisfying the other one. Modeled like that, there is no guarantee that both restrictions are enforced for the same author.

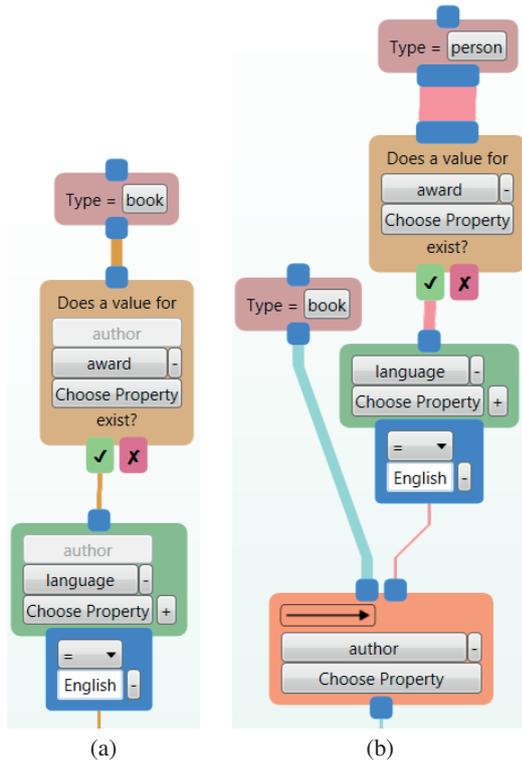


Figure 4: (a) “Retrieve all books with at least one author who has won an award and at least one author whose language is English.” and (b) “Retrieve all books with at least one author to whom both conditions apply.”

As a solution, we propose to use an additional graph entry point that produces a flow representing the set of eligible persons who satisfy both requirements, like it is shown in Figure 4b. As defined in the extended filter/flow model, filter nodes can have several receptors with different meanings to the filter function. That way, a comparison filter node that receives the total set of books on its default receptor can narrow that set down to contain only books whose author is in the previously filtered set of persons—represented by a flow connected to the second receptor of said filter node. This representation maintains the general filter/flow metaphor in that the effect of filter nodes is only significant for parts of the graph that are located “downstream”. Meanwhile, users do not have to handle any variable names, as all relevant connections are expressed by visible flows. Users can also obtain some good insight into every step of the filtering process, as the thickness of the flow representing the eligible persons indicates the effect of each of the person-related filters independently of the set of books up to the point where the books are actually filtered based on their authors.

The last problematic structure in queries that we have considered in our approach builds upon the support for object properties. As shown above, filtering data based on matching property values with a set retrieved from another flow is feasible; however, checking whether two given properties refer to exactly the same object from another set, rather than just to any two objects from that set, is challenging. As an example, a query that selects movies directed by one of their actors would *not* be constructed as shown in Figure 5a: According to the common filter/flow logic, the *starring* and the *director* attributes would just require *any* person, but not neces-

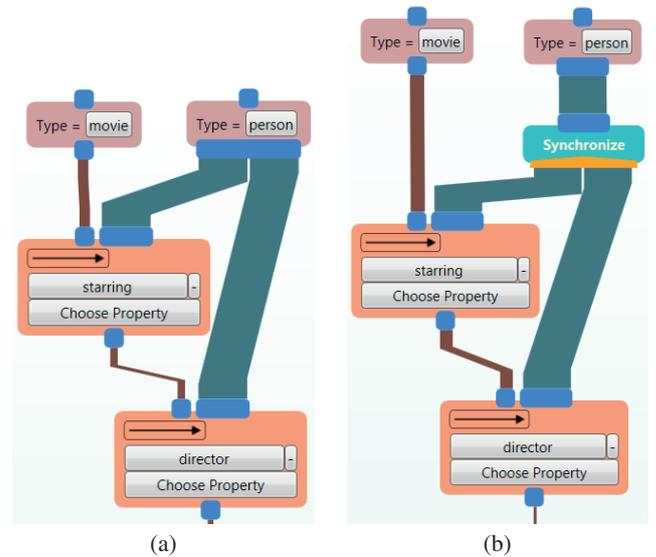


Figure 5: (a) “Retrieve all movies that have at least one actor and one director (from the total set of known persons),” and (b) “Retrieve all movies where one of the actors was also a director of the movie.”

sarily the same person. Therefore, we have incorporated the special *Synchronize* node into our concept, which ensures that for any single result, the same item from each of its outbound flows will be used by the next receptor (cf. Figure 5b). Again, this avoids confronting users with the notion of variables that are not represented by any visual connections. It also enables the visualization of complex queries for linked and nested data structures.

4.3 Implemented Prototypes

We have implemented the approach as a stand-alone application in C# based on the visual querying framework for filter/flow graphs we developed [17]. The application uses the Microsoft Windows Presentation Foundation (WPF) toolkit for the graphical user interface. Users can insert and delete filter nodes and connect them with flows. The filter nodes can be freely placed, while flows are automatically positioned.

An expression generator component traverses the graph and gradually constructs a SPARQL query that comprises of all restrictions defined in the considered part of the graph, usually but not exclusively by adding statements to the *WHERE* clause of the resulting query. This process can be invoked at any emitter in the graph. The resulting SPARQL expression represents the query to retrieve the intermediate result set returned by that particular emitter.

We have also tested the approach on a variety of other platforms: A student project has shown that the prototype can be transformed into a multi-touch-enabled application for Microsoft PixelSense with slight modifications to the existing prototype. Another such project has successfully implemented a modified front-end with some of the filter nodes that works on Android Smartphones. Moreover, we have created a web prototype of the approach for demonstration purposes. It provides limited functionality and lacks many of the key features, such as chaining different filter nodes in an arbitrary order, or splitting up a flow into several result sets. However, it gives an idea of the approach and is publicly available at <http://sparql.visualdataweb.org>.

5. DISCUSSION AND EVALUATION

The general filter/flow concept is empirically well-founded, as it has been successfully applied and evaluated in a number of works.

5.1 Empirical Background

Some of the works examined the filter/flow concept in user studies and expert reviews. For instance, Young and Shneiderman [33] evaluated the comprehension and composition of the filter/flow concept by comparing it with a text-only SQL interface. They performed a user study with twenty participants who had to read and to create Boolean expressions with both interfaces. The participants had to solve different tasks and the number of errors and correct queries were counted. Significant differences in both the comprehension and composition tasks clearly showed the effectiveness of the filter/flow concept.

The effectiveness of the filter/flow concept was also demonstrated in the work of Jetter et al. [23], which adopts the filter/flow concept for tabletop displays by adding tangible components. They observed that the concept was very effective with small failure rates in two user studies. The study participants quickly learned and applied the visual metaphor and were able to create complex queries with it.

We have conducted another user study with 28 participants in which we compared the basic and extended model [16]. We found that the extended filter/flow graphs are at least as readable as the basic ones despite their smaller size. They were preferred by the participants who could read them faster and perceived them as more clear than the basic ones.

5.2 User Study

While the aforementioned evaluations have shown the general applicability and usability of the filter/flow concept, we have conducted a qualitative user study to gather feedback on our particular adaptation of the filter/flow approach for SPARQL querying. Similar to Young and Shneiderman [33], we have evaluated both the comprehension and composition of SPARQL queries with our approach, using the prototype described in Section 4.3. In the following, we will report on the method and results of the user study.

5.2.1 Participants

Half of the ten participants who took part in the study were female, even though we did not expect any gender effects. They were within an age range from 24 to 44 years (mean age of 28 years). All had normal or corrected to normal vision. 80 percent of them had a university degree, which reflects the user group of academic professionals who may be required to create complex queries as part of their work. All participants had good English skills and half of them were already superficially familiar with the Semantic Web, though none considered themselves experts in the topic.

5.2.2 Tasks

We prepared three comprehension and three composition tasks for the study that were presented in an alternating order. In the comprehension tasks, the study participants had to correctly read visual queries created with the prototype. They were shown a screenshot of a visual query similar to the one depicted in Figure 3 and had to describe in their own words what it does. An example for a such a task was to correctly read and interpret a filter/flow graph querying all movies with a budget of more than 15 million dollars. In the composition tasks, they had to create given queries with the prototype on their own. The queries were provided as natural language text, which had to be replicated as visual queries with the inter-

Table 1: Mean values and standard deviations (SD) of the ratings from the study participants regarding comprehension, composition and the overall concept.

	Mean	SD
Comprehension	9.0	0.8
Composition	8.4	1.4
Overall Concept	9.0	1.2

active prototype we developed (cf. Section 4.3). For example, in one of the composition tasks, the study participants had to create a graph that queries cities in Brazil or Peru that have more than one million inhabitants. DBpedia was used as data source in all cases.

5.2.3 Procedure

First, participants were asked to provide some demographic data in a questionnaire. Subsequently, they were made familiar with the basic ideas of the Semantic Web as well as the filter/flow concept, and the prototype was briefly explained to them. After a few training tasks that were not included in the evaluation, participants were given written descriptions of the tasks they had to solve with the prototype. They were asked to verbalize their thoughts as they were solving the tasks, and their statements were noted. Inquiries about the concept were allowed, unless they were directly related to the task. Finally, participants had the opportunity to state their impression of the concept and the tasks in a concluding questionnaire and a structured interview.

5.2.4 Results

All tasks were correctly solved by the participants and no one reported any problems in understanding the tasks. The participants quickly got used to the prototype and were able to recognize and correct mistakes without any help. Common mistakes were to set an incorrect connection between two nodes or to select the wrong class from the menu. Often, the participants recognized the mistakes already by the flow thickness. Otherwise, they realized that there were incorrect values in the result set and corrected the graph before they answered the task.

A few participants had difficulties thinking in classes, objects and properties. Especially property paths caused problems in some cases. For instance, when asked for cities in Peru or Brasil in one of the tasks, the most efficient strategy would be to first select the city and then the country, whereas some participants tried to first filter the countries and then the cities. However, often there was more than one way to solve a task, so that different filter/flow graphs could represent the same query. While some participants preferred to work with the simple filter nodes, others tried the more complex ones, typically resulting in more efficient solutions and simpler graph structures.

The use of colors was not clear to all participants. While most of them did not think about the coloring any further, some expected connected flows and filters to have the same color. Implementing this would cause other problems, but it shows that the use of colors in filter/flow graphs is an issue that should receive more attention in future work.

Another suggestion for future extensions is a feature that encapsulates queries or query parts in a filter node that can be reused in another query. A related idea was to offer a special filter node for expert users that allows to define parts of the query in the SPARQL text syntax. However, this is already possible by editing the textual SPARQL query that results from the graph, if users know how to interpret it.

Finally, the participants had to rate the comprehension and composition of queries with our approach as well as the overall concept on a scale from 1 to 10 (1 = very bad, 10 = very good). As shown in Table 1, the results of these final ratings were very positive.

6. CONCLUSIONS AND FUTURE WORK

Originally, the filter/flow concept has been developed for relational databases [33]. There are also some adaptations of the concept for object databases [27] and spatial databases [26]. In this paper, we have further developed the concept for RDF databases and SPARQL querying, using the extended filter/flow model as a basis [15]. It allows to define sophisticated filter nodes with several receptors and emitters that are connected to individual sets of flows. As we have shown in this paper, it can effectively be used to represent the features and semantics of SPARQL query restrictions in an intuitive way.

Our approach can be applied to any RDF dataset that provides a SPARQL endpoint, without a need to configure it for a certain database schema. It can handle filtering not only by scalar attribute values, but also based on attributes of linked objects as they appear in RDF graph structures. At the same time, the filter/flow metaphor is completely maintained, providing users with a consistent underlying concept for the visualization of complex queries. While we have tailored our concept to RDF and SPARQL, our findings can basically be applied to any object graph.

Furthermore, we have removed the necessity for users to input or review any structured query source code. No knowledge of SPARQL and other Semantic Web technologies beyond a basic understanding of RDF is required to handle the presented filter/flow graphs. However, we still rely on textual labels, or—if no labels are available—simple strings extracted from resource URIs to name properties in the visual interface. Taking advantage of representative images in some cases, as they are provided in datasets like DBpedia, may be worthwhile both for recognizing single resources and getting a better overview over the complete graph, as different nodes would then be visually more distinguishable.

We have focused on the representation of query restrictions in our work. So far, we have included and tested means to use these restrictions to generate `SELECT` and `ASK` queries. We are considering to add support for `DESCRIBE` and `CONSTRUCT` queries in the future. However, some additional concepts will have to be integrated for that goal, such as an intuitive way to specify the desired structure of the graph that contains the result of the `CONSTRUCT` query.

Moreover, we are planning to make use of the available schema information accessible via SPARQL to aid the construction of filter/flow queries with suggestions about what filters to add next. Information that can be directly retrieved and hence considered for the suggestions includes properties that have a certain range of values across the items of the dataset and thus could noticeably restrict the number of resulting items, without filtering away too much and yielding an empty set. Likewise, the datatype mainly used for a given property can be determined, resulting in a recommendation of the most appropriate filter node type.

Finally, we expect that there is an upper limit for the size of filter/flow graphs that can be displayed at once and remain readable to the user. Determining when that limit is reached and developing context-specific strategies to deal with that situation—for example, grouping parts of the graph in subqueries to reduce the overall number of nodes, or automatically determining reasonable fractions of the graph that could be split into separate views—will be another subject of future work.

7. ACKNOWLEDGMENTS

We would like to thank our student research assistants Sukanya Bhowmik and Hao Li for their assistance with the prototype implementation. This work was partly supported by the BMWi funded project IP-KOM-ÖV (grant no. 19P10003N) and by the EU funded project iPatDoc (grant no. 606163).

8. REFERENCES

- [1] Linked data - connect distributed data across the web. <http://linkeddata.org>.
- [2] OpenLink iSPARQL. <http://oat.openlinksw.com/isparql/>.
- [3] Pipes: Rewire the web. <http://pipes.yahoo.com/pipes/>.
- [4] SPARQL endpoints status. <http://sparqls.okfn.org>.
- [5] O. Ambrus, K. Möller, and S. Handschuh. Konduit VQB: a visual query builder for SPARQL on the social semantic desktop. In *Proceedings of the Workshop on Visual Interfaces to the Social and Semantic Web (VISSW '10)*, volume 565 of *CEUR-WS*, 2010.
- [6] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC '07/ASWC '07)*, pages 722–735. Springer, 2007.
- [7] G. Barzdins, S. Rikacovs, and M. Zviedris. Graphical query language as SPARQL frontend. In *Proceedings of the 13th East-European Conference on Advances in Databases and Information Systems (ABDIS '09), Associated Workshops and Doctoral Consortium*, pages 93–107. Riga Technical University, 2009.
- [8] C. Becker and C. Bizer. Exploring the geospatial semantic web with DBpedia mobile. *Web Semantics*, 7(4):278–286, 2009.
- [9] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.
- [10] C. Bizer and A. Schultz. The berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems*, 5(2):1–24, 2009.
- [11] J. Borsje and H. Embregts. Graphical query composition and natural language processing in an RDF visualization interface. Erasmus University Rotterdam, 2006. Bachelor thesis.
- [12] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.
- [13] N. Elmqvist, J. Stasko, and P. Tsigas. DataMeadow: A visual canvas for analysis of large-scale multivariate data. *Information Visualization*, 7(1):18–33, 2008.
- [14] J. Groppe, S. Groppe, and A. Schleifer. Visual query system for analyzing social semantic web. In *Proceedings of the 20th International Conference on World Wide Web (WWW '11), Companion Volume*, pages 217–220. ACM, 2011.
- [15] F. Haag, S. Lohmann, and T. Ertl. Simplifying filter/flow graphs by subgraph substitution. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '12)*, pages 145–148. IEEE, 2012.
- [16] F. Haag, S. Lohmann, and T. Ertl. Evaluating the readability of extended filter/flow graphs. In *Proceedings of the 2013 Graphics Interface Conference (GI '13)*, pages 33–36. CIPS, 2013.

- [17] F. Haag, S. Lohmann, and T. Ertl. A flexible architecture for filter/flow-based visual querying. In *Proceedings of the Graphics Interface Poster Session 2013*, 2013.
- [18] T. Hansaki, B. Shizuki, K. Misue, and J. Tanaka. Findflow: Visual interface for information search based on intermediate results. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation (APVis '06)*, pages 147–152. ACS, 2006.
- [19] P. Heim, J. Ziegler, and S. Lohmann. gFacet: A browser for the web of data. In *Proceedings of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW '08)*, volume 417 of *CEUR-WS*, pages 49–58, 2008.
- [20] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A browser for heterogeneous semantic web repositories. In *Proceedings of the 5th International Semantic Web Conference (ISWC'06)*, pages 272–285. Springer, 2006.
- [21] F. Hogenboom, V. Milea, F. Frasinca, and U. Kaymak. RDF-GL: A SPARQL-based graphical query language for RDF. In R. Chbeir, Y. Badr, A. Abraham, and A.-E. Hassanien, editors, *Emergent Web Intelligence: Advanced Information Retrieval*, Advanced Information and Knowledge Processing, pages 87–116. Springer London, 2010.
- [22] M. Jarrar and M. D. Dikaiakos. MashQL: A query-by-diagram topping SPARQL. In *Proceedings of the 2nd International Workshop on Ontologies and Information Systems for the Semantic Web (ONISW '08)*, pages 89–96. ACM, 2008.
- [23] H.-C. Jetter, J. Gerken, M. Zöllner, H. Reiterer, and N. Milic-Frayling. Materializing the query with facet-streams: A hybrid surface for collaborative search on tabletops. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*, pages 3013–3022. ACM, 2011.
- [24] S. Lohmann, P. Heim, T. Stegemann, and J. Ziegler. The RelFinder user interface: Interactive exploration of relationships between objects of interest. In *Proceedings of the 15th International Conference on Intelligent User Interfaces (IUI '10)*, pages 421–422. ACM, 2010.
- [25] C. Morbidoni, A. Polleres, D. L. Phuoc, and G. Tummarello. Semantic web pipes. Technical Report 2007-11-07, Digital Enterprise Research Institute (DERI), 2007.
- [26] A. Morris, A. Abdelmoty, B. El-Geresy, and C. Jones. A filter flow visual querying language and interface for spatial databases. *GeoInformatica*, 8(2):107–141, 2004.
- [27] N. Murray, N. Paton, and C. Goble. Kaleidoquery: A visual query language for object databases. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '98)*, pages 247–257. ACM, 1998.
- [28] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. <http://www.w3.org/TR/rdf-sparql-query/>, 2008.
- [29] A. Russell, P. Smart, D. Braines, and N. Shadbolt. NITELIGHT: A graphical tool for semantic query construction. In *Proceedings of the 5th International Workshop on Semantic Web User Interaction (SWUI '08)*, volume 543 of *CEUR-WS*, 2008.
- [30] m. c. schraefel, D. A. Smith, A. Owens, A. Russell, C. Harris, and M. Wilson. The evolving mspace platform: Leveraging the semantic web on the trail of the memex. In *Proceedings of the Sixteenth ACM Conference on Hypertext and Hypermedia (HYPERTEXT '05)*, pages 174–183. ACM, 2005.
- [31] The W3C SPARQL Working Group. SPARQL 1.1 overview. <http://www.w3.org/TR/sparql11-overview/>, 2013.
- [32] M. Tobiasz, P. Isenberg, and S. Carpendale. Lark: Coordinating co-located collaboration with information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1065–1072, Nov. 2009.
- [33] D. Young and B. Shneiderman. A graphical filter/flow representation of boolean queries: A prototype implementation and evaluation. *Journal of the American Society for Information Science*, 44(6):327–339, July 1993.