Facet Graphs: Complex Semantic Querying Made Easy

Philipp Heim[1], Thomas Ertl[1], and Jürgen Ziegler[2]

[1] Visualization and Interactive Systems Group (VIS), University of Stuttgart, Germany
{Philipp.Heim,Thomas.Ertl}@vis.uni-stuttgart.de
[2] Interactive Systems and Interaction Design, University of Duisburg-Essen, Germany
Juergen.Ziegler@uni-due.de

Abstract. While the Semantic Web is rapidly filling up, appropriate tools for searching it are still at infancy. In this paper we describe an approach that allows humans to access information contained in the Semantic Web according to its semantics and thus to leverage the specific characteristic of this Web. To avoid the ambiguity of natural language queries, users only select already defined attributes organized in facets to build their search queries. The facets are represented as nodes in a graph visualization and can be interactively added and removed by the users in order to produce individual search interfaces. This provides the possibility to generate interfaces in arbitrary complexities and access arbitrary domains. Even multiple and distantly connected facets can be integrated in the graph facilitating the access of information from different user-defined perspectives. Challenges include massive amounts of data, massive semantic relations within the data, highly complex search queries and users' unfamiliarity with the Semantic Web.

Keywords: Graph visualization, faceted search, query building, SPARQL, hierarchical facets, pivot operation.

1 Introduction

Ten years ago, Tim Berners-Lee planted the seed for the Semantic Web [1] which thereupon started to grow and expand over the years. At first, information was manually translated into semantic structures and added to the Semantic Web resulting in a rather moderate growth. The automation of the translation process at a later date allowed large amounts of existing information as available e.g., in encyclopedias, medical databases or in knowledge bases of other domains to be integrated in the Semantic Web causing a rapid acceleration of its growth rate. Today, the Semantic Web contains so much semantically annotated information that it seems ready for broader exploration. Alone appropriate tools are missing.

In this paper, such tools are defined as something, which allows humans to access information contained in the Semantic Web according to its semantic descriptions. Finding the right information, however, requires the semantic of what should be searched to be specified by the user. For this purpose it is not sufficient to enter words in an input field as it is usual for search engines in the common Web since natural language is ambiguous. A basic strategy to avoid the ambiguity of natural language

L. Aroyo et al. (Eds.): ESWC 2010, Part I, LNCS 6088, pp. 288–302, 2010.

are artificial query languages like *SPARQL*[1] that are uniquely defined. A growing amount of information in the Semantic Web (e.g. *DBpedia* [2] or the *LOD cloud* [3]) can be queried via SPARQL endpoints that are freely accessible over the Internet. However, building search queries in such an artificial language, as for example possible with *SNORQL*[2], requires the language to be learned by the user and is thus rather a task for experts. The vast majority of users need intuitive interfaces to express search queries that are semantically unique but do not require any extra knowledge.

A popular approach for such interfaces is based on the concept of faceted search [4]. In faceted search, the search space gets partitioned using orthogonal conceptual dimensions whereas one acts as the result set and the others as facets. The facets can then be used to filter the result set by different attributes that can be selected independently from each other. Selecting an attribute in a facet adds it to the query and hence filters the result set accordingly. Whenever the result set gets changed, all the facets are updated to only those attributes that can be used for further filtering (Fig. 1). Therefore, users always see only attributes that can cause a reduction of the result set, but can never cause an empty set when selected. Moreover, since all attributes that can be used for filtering are extracted from the semantic structures of the Semantic Web, all search queries contain only uniquely defined objects, classes and properties and thus completely avoid ambiguity.

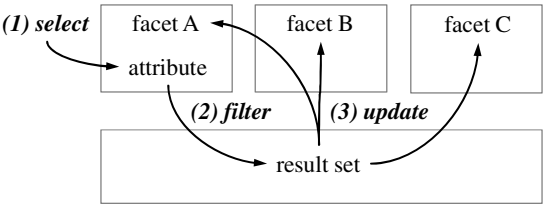

Fig. 1. In faceted search, selecting an attribute in a facet (1) filters the result set (2), which thereupon updates all facets (3)

In order to support a better understanding of our theoretical explanations throughout this paper, we use concrete examples and tasks from the field of football (soccer). In this domain, a faceted search would allow, for example, football players to be filtered by their clubs, their birthplaces, or their ages. Even so the concept of faceted search is successfully applied in popular applications such as *Apple's iTunes*[3], using it to seek information in the Semantic Web entails several new problems that are not yet sufficiently solved. Most of the problems result from the vast number of objects, classes and properties that are contained in the Semantic Web. According to [5], the LOD cloud alone contained approximately two billion statements in 2008 already.

In this paper we introduce *Facet Graphs*, a new approach based on the concept of faceted search which allows the intuitive formulation of semantically unique search queries. Users can choose the result set as well as the facets to filter it and thereby

[1] http://www.w3.org/TR/rdf-sparql-query/
[2] http://dbpedia.org/snorql/
[3] http://www.apple.com/de/itunes/

produce a personalized interface to build search queries. Facets and result set are represented as nodes in a graph visualization and are connected by labeled edges that fosters users' understanding of the relationships between the facets and also allows for an easy extension by further nodes and edges. The graph provides a coherent representation of multiple, even distantly connected facets on one page that completely avoids any browsing activities and hence prevents users from getting 'Lost in Hyperspace'. Each node contains a list that provides sorting, paging and scrolling functionalities and thereby enables the easy handling of even large amounts of objects.

In Facet Graphs, users can build queries just by clicking certain objects in the nodes. All filtering effects caused by such selections are highlighted by different colors in the graph that can be used for both, a better understanding of the caused effects or a better traceability of reasons for filtering effects on a certain node. Since users may change their minds while searching, the result set can be changed at any time enabling a rather exploratory search when the domain, the way to achieve the goal, or the goal itself is not completely clear in advance. Altogether, the combination of graph visualization with faceted search allows for an easy formulation of even complex semantically unique search queries and thus to seek for information according to its semantic without expert knowledge.

The rest of this paper is organized as follows. First we review related work and use the identified weak points for the motivation of our approach of Facet Graphs. We then describe our approach in detail and evaluate its usability in a comparative user study. Since the problem of building semantically unique queries using intuitive interfaces based on the concept of faceted search is comparatively new and has not yet been solved sufficiently, we then propose a list of requirements that need to be fulfilled by such systems in order to support the information seeking process in general. These requirements are then used to better discuss the advantages and limitations of our own approach and those mentioned in the section about related work. Finally, we give a conclusion and an outlook on future work.

2 Related Work

A number of approaches to build semantically unique search queries using faceted search have been described in the literature. Most of the approaches display the facets as well as the result set as lists at different positions on the screen. Examples are tools like *mSpace* [6], *Flamenco* [4], *Longwell* [7] and *Haystack* [8]. Facets provided by these tools are limited to directly connected ones exclusively. So, football players could be filtered by facets containing the clubs they are playing for, the city they are living in or the number on their shirts. By selecting, for example, the football club 'FC Chelsea' in the facet with all existing clubs, the result set with all football players could be filtered to only those players that are playing for this club. Hierarchical filtering, however, that allows also indirectly connected facets to be used for query building (e.g. get only players playing for clubs in the Premier League) is not supported by these tools.

Tools like *Parallax* [9], *Humboldt* [10], *Tabulator* [11] and the *Nested Faceted Browser* [12], by comparison, allow for hierarchical filtering. Therefore, football players can be filtered by the cities where clubs they are playing for are located. Thus

the possible options for building a query are not restricted to the direct periphery around a result set but can include dimensions that are distantly connected by other facets. Depending on the directness of their connections to the current result set, they can be integrated in a hierarchy and are therefore called *hierarchical facets*.

Fig. 2. Screenshots of Parallax (left) and Tabulator (right)

In Parallax (Fig. 2, left), Flamenco and Humboldt, the hierarchy of facets is never completely visible to the user but only parts of it. By providing ways to browse through the hierarchy, users are able to include attributes of also distantly connected facets in their search query. With the growth of the complexity of their queries, it is getting more difficult for users to keep an overview of all included attributes since the corresponding facets are often scattered over several different pages.

Tools like Tabulator (Fig. 2, right) and the Nested Faceted Browser, on the contrary, allow the whole hierarchy to be displayed on one page. By using a tree structure to display all available hierarchical facets, users can open and close even distantly connected facets in one coherent view and thus keep an overview of all attributes that are included in their queries. Since tree structures are used to depict all kinds of taxonomies in a wide range of well known applications, users need no extensive period of training to gain an understanding of how to use them.

In Tabulator and the Nested Faceted Browser, every attribute defines its own subtree that can be expanded by the user in order to see distantly connected facets and their attributes, which again can be expanded and so on (cp. Fig. 2, right). So for the football players, one of the clubs they are playing for (e.g. 'FC Chelsea'), could be expanded by the user to see, for example, the city of its location (here: 'London'). Selecting 'London' would cause the list of players to be filtered to only those that are playing for 'FC Chelsea' or any other club located in this city. A combined list of all the cities that could be selected in order to filter the clubs and also the players, however, is not provided by the tree structure. The cities are partitioned in different subtrees, each for every club, that all need to be expanded in order to see all available cities. Having many subtrees opened, however, places attributes that actually belong to one facet at many different positions in the tree, leading to an increased tree structure that can possibly not be displayed on one screen. If an attribute is shared by more than one object (e.g. many clubs are located in London), it also occurs repeatedly in different subtrees.

Altogether, a tree structure is a well known paradigm to visualize and interact with hierarchical data; however, when used for hierarchical facets, it tends to produce large and highly subdivided structures that cannot easily be overviewed by users. In this paper we therefore propose an alternative, graph-based approach to visualize and interact with hierarchical facets that aims at preventing large and confusing tree structures and hence facilitates an easy generation of semantically unique queries by the user.

3 Facet Graphs

In Facet Graphs, facets and result set are represented as nodes in a graph visualization. The semantic relations that exist between facets and result set as well as facets and other facets are represented by labeled directed edges between the nodes. Fig. 3 shows *gFacet*[4], a prototypical implementation of our approach of Facet Graphs.

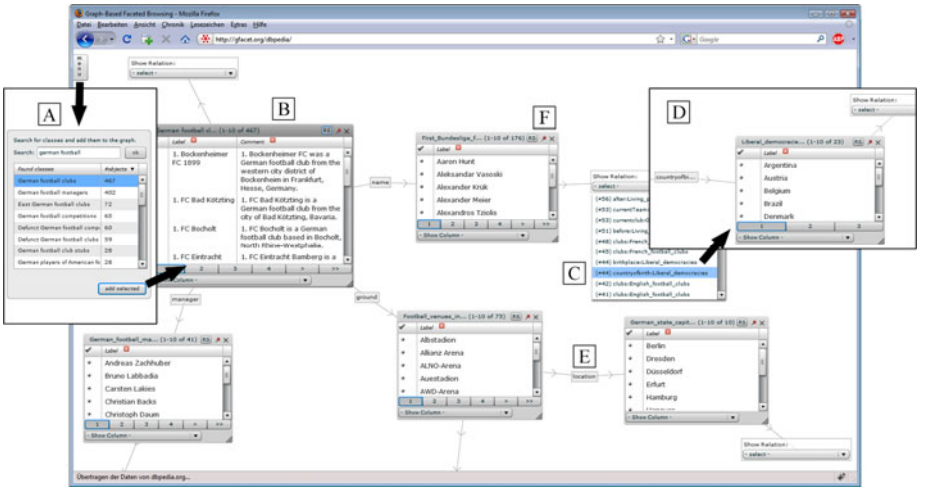

Fig. 3. In gFacet, the result set is defined by the user (A) and represented by the initial node in the graph (B). By selecting properties out of drop-down-lists (C), facets can be added as new nodes (D) that get connected by labeled edges (E).

In gFacet, the node representing the result set is marked by a darker background color (cp. Fig. 3, B) to better distinguish it from the nodes representing facets. The attributes of each facet are not located at different positions on the screen but are grouped within one single node (e.g. the cities where football venues are located in Fig. 3). Every node in the graph contains a list that can be scrolled, paged and sorted by the user allowing even large amounts of objects to be clearly arranged. If an attribute is shared by more than one object (e.g. several players are born in Poland) it still occurs only once in the list in comparison to the tree-based approaches described in related work.

[4] A description of an early version of this prototype with limited functionalities and access to dummy data only can be found in [13].

The graph can interactively be expanded by additional nodes with further facets that are not represented yet. For each node in the graph, facets that can be added are available via a drop-down list (e.g. facets for the 'First Bundesliga Footballers' in Fig. 3, C). The facets in the drop-down lists are ordered by the number of their attributes, with the largest numbers presented first. Selecting one of the facets in a list (e.g. the one containing the countries where footballers are born), adds this facet as a new node to the graph and connects it to the existing node by a labeled edge (cp. Fig. 3, D). Thus, the user can iteratively add and remove facets to the graph and hence produce a personalized interface to build search queries.

The nodes in the graph are positioned in an aesthetically pleasing way by performing a force-directed layout algorithm [14]. This can cause the nodes to often change their position, possibly making it hard to maintain visual focus and thus confusing the user. We therefore apply a pinning mechanism that forces nodes to hold their position. When a new node is added to the graph, the pinning of this node is executed after a short time period. This delay allows the force-directed algorithm to position new nodes in an appropriate way and at the same time prevents already existing information to change their location. Whether a node is pinned or not is indicated by the color of the needle symbol at the upper right corner of each node (cp. Fig. 3, F). This pinning can also be controlled by the user by clicking at the needle symbol. Thus, the user can decide whether a node should stay on a fixed position or should be rearranged by the force-directed algorithm in order to improve the overall appearance of the graph.

The general benefits of representing hierarchical facets as nodes in a graph are:

1. The attributes for each facet are grouped into one node.
2. All nodes are shown in a coherent presentation on one page.
3. Semantic relations between the nodes are represented by labeled edges (Fig. 3, E).
4. Facets can be added and removed by the user (Fig. 3, D).

3.1 Extracting Hierarchical Facets

In order to provide facets in drop-down lists to expand the graph, they first have to be extracted from the underlying data structure. In gFacet, we build SPARQL queries on the client side, send them to SPARQL endpoints using HTTP requests and extract the facets from the resulting XML data. The client-server communication as well as the graphical user interface are implemented in *Adobe Flex*[5] and thus compiled to a Flash movie, which runs in all Web browsers with Flash Player installed. An exemplary implementation of gFacet that uses DBpedia's SPARQL endpoint is accessible online[6]. Since gFacet does not need any modification on the server side, it can be used to access information from other SPARQL endpoints as well.

Our general strategy of how to extract facets from semantic structures by querying SPARQL endpoints is based on the links within these structures. The links are defined by properties like 'plays for' and connect objects like 'Franck Ribéry' with, for example, 'FC Bayern Munich'. So given a list with several objects including 'Franck

[5] `http://www.adobe.com/products/flex`
[6] `http://gfacet.semanticweb.org`

Ribéry', a possible facet to filter this list would be the football clubs at least one of the objects in the list are playing for. Other facets representing other properties of even other objects could also be used for filtering. With all the objects that exist in the underlying data structure (e.g. DBpedia) put together in one list, however, would result in so many possible facets to filter this list that they could not be read by the users. So, before selecting facets to filter the list, the users first have to restrict the list to only objects that are of a certain ontological class. This is always the first task when searching with gFacet and defines the result set for the current search.

By entering words in an input field, users can search for ontological classes that meet their interests (e.g. 'German football' in Fig. 3, A). The corresponding SPARQL query, which returns all the classes with labels containing the words entered by the users (here: "German Football"), is given in the following:

```
SELECT DISTINCT ?class ?label COUNT(?o) AS ?numOfObj
WHERE { ?class rdf:type skos:Concept .
?o skos:subject ?class .
?class rdfs:label ?label .
?label bif:contains "german and football" .
FILTER (lang(?label) = "en") }
ORDER BY DESC(?numOfObj) LIMIT 30
```

The found classes are shown in a list and are initially ordered according to the numbers of objects contained in each class, with the largest counts presented first. One of the suggested classes can then be selected by the user to become the first node in the graph and also the current result set (e.g. 'German football clubs'). The first node contains all the objects of the selected class and is the starting point for all further nodes in the graph (cp. Fig. 3, B).

Based on the properties of the objects in the result set, available facets to filter them are extracted automatically and are displayed in a drop-down list next to the result set. Each facet in the drop-down list consists of a property (e.g. 'ground') and one class of objects this property is leading to (e.g. 'Football venues in Germany'). So it is possible to have several facets with the same property but with different classes, or different properties but with the same class. The corresponding query looks like the following:

```
SELECT DISTINCT ?prop ?newClass
COUNT(DISTINCT ?objNewClass) AS ?numOfObj
WHERE {
?objCurrClass skos:subject <URIofGermanFootbalClubs> .
?objCurrClass ?prop ?objNewClass .
?objNewClass skos:subject ?newClass .
?newClass rdf:type skos:Concept .}
ORDER BY DESC(?objNewClass) ?prop ?newClass LIMIT 40
```

Similarly to adding first order facets to the result set, also facets of second or higher order can be added to the graph and thus used for filtering. The number of visible objects in higher order facets is restricted to only those that are connected to objects in other visible facets that are in turn directly or indirectly connected to objects in the result set. Fig. 4 shows a result set on the left, a first order facet in the middle and a

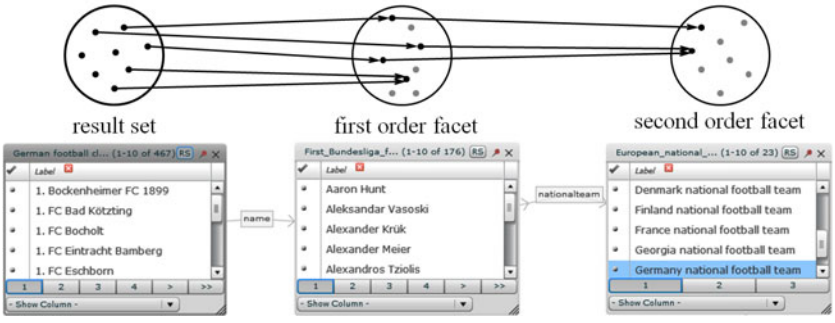

Fig. 4. Only objects that are connected to the result set are visible in the hierarchical facets: Here the national teams for which players of German clubs are playing

second order facet on the right together with a schematic representation of the visible and invisible objects. Black dots are visible objects and gray dots are invisible objects. The gray dots in the schematic representation of the second order facet represent objects that are of the same class but are not connected to an object in the result set and therefore not visible for the users. In this way, all the facets in the graph only contain objects that can be used for filtering but will never result in an empty result set.

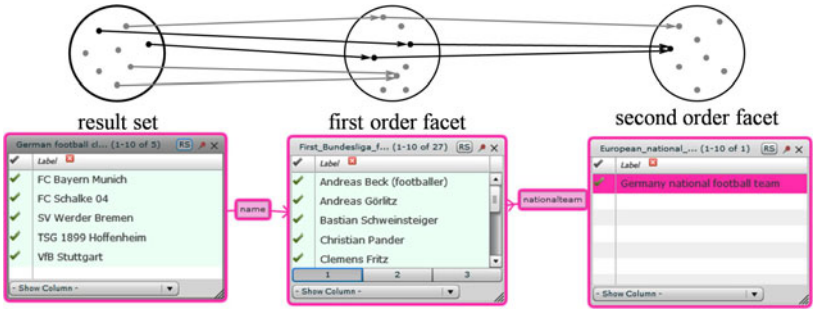

Fig. 5. Hierarchical facets can be used for filtering

3.2 Building Search Queries

Having chosen at least one facet to be represented as node in the graph, it can be used to build semantically unique search queries. By selecting one of the objects in a facet, the result set can be filtered to only objects that are directly or indirectly connected to the selected one. E.g. the selection of 'Germany national football team' in the second order facet in Fig. 4 filters the football clubs to only those with players playing for this national team (see the new result set in Fig. 5). If a selection actually filters the result set, all lists get updated within a few seconds.

To support the traceability of filtering effects we use color-coding. Every node has a certain color assigned to, which marks all filtering effects caused by selections in this node. If an object gets selected by the user, the selected object itself, all the filtered nodes and all the edges between them are marked with the same color (Fig. 5).

Nodes containing the same number of objects after and before the selection are not filtered and hence are not marked additionally. This way, the colors in the graph support an understanding of not only what effects are caused by certain selections but also what reasons exist for certain restrictions. For every restricted node, all relevant selections can be traced back by the user just by following the colors in the graph even long after the selections took place.

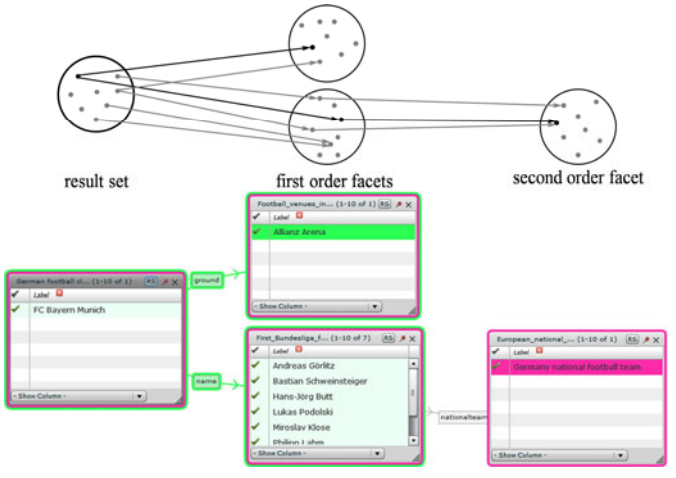

Fig. 6. Multiple selections are possible

This is especially useful in case of multiple selections in different facets because every selection gets marked by a different color and thus supports users to distinguish between the different filtering effects in the graph[7]. As can be seen in Fig. 6, filtered nodes get surrounded by colored rings for each selection that affects them. Like tree-rings, new colored rings are added around the existing ones and thus allow users to understand all filtering effects by looking at the colored ring growth patterns in the graph. However, unlike tree-rings, clicking at an already selected object deselects it and reverses the corresponding filtering effects with all their color-coding including also rings around nodes. In the current implementation, multiple selections are connected by a logical 'AND' and thus narrow down the result set to only objects that are connected to all selections. Thus by selecting a certain ground, the clubs in the result set in Fig. 6 are restricted to only those that have players playing in a certain national team and are tenants of a certain venue.

3.3 Pivot Operation

While choosing facets to add them to the graph or selecting objects to filter the result set, users may change their minds about what they want to search. When, for example, searching for 'German football clubs', users could realize that they rather want to

[7] A screencast of a typical usage scenario is available at `http://gfacet.semanticweb.org`

search for 'First Bundesliga footballers' and want to use the football clubs to filter the footballers instead of the other way around. This operation is called *pivot operation* since it allows leaving the used path and following another direction. The pivot operation is based on the operation in data drilling [15], where data can be represented according to different dimensions.

The Nested Faceted Browser [12] and Humboldt [10] are the only tools mentioned in the section about related work that allow users to perform a pivot operation and thus to change the focus of their search. In Humboldt, the user can replace the current result set, displayed in the centre of the screen, by one of its visible facets, arranged among each other on the right-hand side. The chosen facet becomes the new result set and vice versa, the replaced result set becomes a new facet. In Humboldt, only directly related facets are shown next to the result set and thus hierarchical facets can only be reached by operating pivots. Whenever a pivot is operated and a new result set is shown in the centre of the screen, the list of directly related facets is updated accordingly. This way, information is partitioned over multiple pages, placing substantial cognitive load on the users to keep track of former result sets and facets, which are not visible yet.

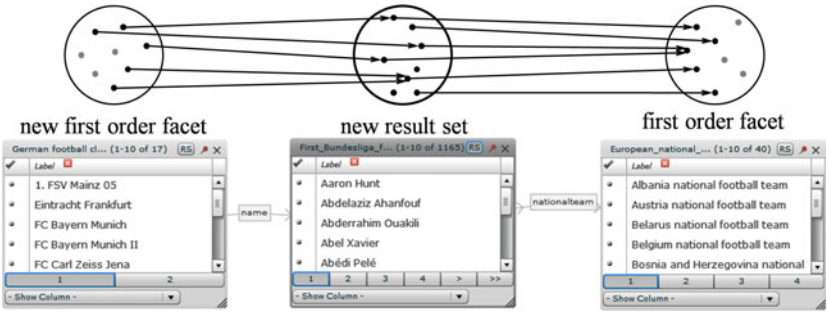

Fig. 7. Hierarchical facets can be used for filtering

Our graph-based approach, by contrast, allows a pivot operation to be performed without any changes of the displayed information structure and thus reduces the cognitive load to keep track of them. Clicking the 'RS'-Button (RS = result set) next to the pinning needle of any of the facets represented in the graph turns this facet into the new result set (e.g. 'First Bundesliga footballers' in the middle of Fig. 7) and vice versa the current result set into a new facet. Whereas other approaches have to rebuild their complete interface to keep the displayed facets up to date, in our approach even distantly related information can be shown on one page and hence can remain at their position while operating pivots. The only aspect that can change in gFacet when operating a pivot is the number of objects in both the result set and the visible facets.

In Fig.7, the current result set has moved from the node on the left to the node in the middle. Therefore, the number of objects in the new first order facet containing 'German football clubs' decreased from 467 (cp. Fig. 4) to 17 in Fig. 7. This is because the new result set contains only first Bundesliga footballers and thus the number of clubs is limited to only those of the first division (i.e. some of the dots are gray in the schematic representation of the left facet in Fig. 7). Because the data provided by

DBpedia is automatically extracted from Wikipedia and thus entails some minor errors, not all first Bundesliga footballers are connected to a German football club and therefore only 176 players were visible in Fig. 4 as it was a first order facet. Since this facet became the new result set, all German football players got visible without any restrictions and so the number of objects increased to 1165 (i.e. all dots are black in the schematic representation of the new result set in Fig. 7).

4 User study

In order to evaluate our approach and thus get a first opinion about its usability, we conducted a user study that compared gFacet with another tool that supports the building of semantically unique search queries based on the concept of faceted search. For both tools we measured to what extent participants were able to solve the following three task types of different levels of difficulty:

1. Find two players who are playing for a certain club.
2. Find two cities where players who are playing for a certain club are born.
3. Find one player who is playing for a certain club and is born in a certain city.

We applied a 2x3 (*tool type* x *task type*) within-subject design to compare gFacet with Parallax [9]. To control learning effects, each participant was assigned to one of two groups. Participants in the one group used gFacet first and then Parallax, while participants in the other group used Parallax first and then gFacet. After completing all three tasks with one tool, the participants were asked to fill out an evaluation sheet to rate this tool. In a final questionnaire, participants had to directly compare both tool types with each other.

Ten participants took part in the study, with an average age of 28.3 (ranging from 24 to 31). Eight of them were male; two were female with all ten participants having normal or corrected to normal vision and no color blindness. Education level of the participants was at least general qualification for university entrance and they were all familiar with computers. The functionalities of both, gFacet and Parallax were introduced by videos in which sample tasks were solved.

4.1 Results

Overall, gFacet performed very well for complex tasks. However, it performed less well for rather simple tasks that could also be accomplished just by following links.

The left bar chart in Fig. 8 shows the number of right solutions for each of the three different task types using the two different tool types. For task 1 and task 2, Parallax performed equally or even better than gFacet. This was mainly because in Parallax participants could accomplish both tasks just by following links and did not have to filter at all. In gFacet, by contrast, they had to filter despite the simplicity of the first two tasks and hence did not accomplish task 2 in four cases. The unfamiliar approach of graph-based facets seemed partly too complex to be properly used in average exploratory tasks. However, in order to find the right solution for more complex tasks, as for example for task 3, gFacet performed significantly better than Parallax. It was almost impossible so to find the right solution for task 3 just by following links in

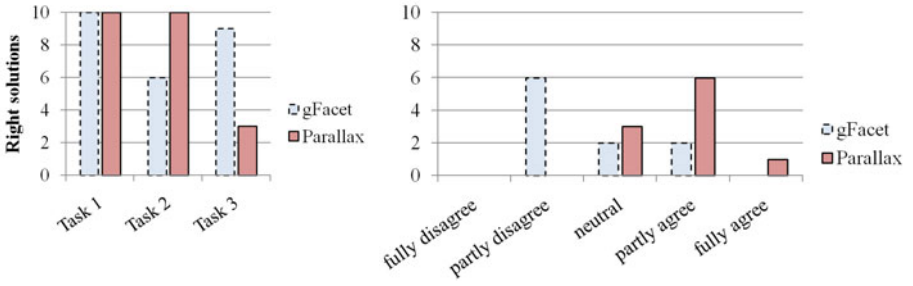

Fig. 8. Number of right solutions to the given tasks (left) and comments to the statements 'It was difficult to understand the relations between the information' (right)

Parallax. All the links had to be checked in a trial-and-error method to actually find the player that is born in the given city. Since this is quite time-consuming, participants were forced to apply filters in Parallax as well.

To find the right solution to task 3, participants had to filter the list of all football players to only players that are playing for a certain club and are born in a certain city. Even though Parallax generally supports applying multiple filters, participants felt uncertain about the solutions they found. A reason for this is the strict separation in Parallax between exploration, which can be realized by clicking links on the right side of the content, and filtering, which can be done by clicking links on the left side (cp. Fig. 2, left). If users explore a list of objects (e.g. cities where players are born) these objects cannot be used for filtering.

On the other hand, if users look at the filter options on the left side, they cannot further explore these options for hierarchical filters. To filter hierarchically, users first have to explore objects via links on the right side (e.g. cities where players are born) and then have to filter this list via links on the left side (e.g. to only cities located in Germany). Having to change from one side to the other can confuse users and thus can decrease their confidence in the found solutions. Moreover, clicking links to explore objects completely replaces the current content as well as the links at both sides. That way information gets partitioned over multiple pages and thus hampers users' understanding of the relations that exist between information (cp. right bar chart in Fig. 8). In gFacet, by contrast, all objects in all lists can be used for both filtering and the exploration of further facets and are all visible on one page.

5 Discussion

In order to better comprehend the advantages and limitations of our approach and also to better compare it to the existing approaches described in the section about related work, we go through the general information seeking process (ISP) as described in [16] and determine whether the different approaches are suitable to support it or not. The ISP from a human's perspective can be described in six stages: task initiation, selection, exploration, focus formulation, collection and presentation [16]. In a user study described in [16], different tasks were revealed that are considered most appropriate by

Table 1. A list of system requirements in order to support the six stages of the information seeking process and their fulfillments by the different approaches

ISP Stages	Task initiation		Selec -tion		Exploration					Focus formulation						Collec- tion		Presenta -tion
Require- ments	R1.1	R1.2	R2.1	R2.2	R3.1	R3.2	R3.3	R3.4	R3.5	R4.1	R4.2	R4.3	R4.4	R4.5	R4.6	R5.1	R5.2	R6.1
mSpace	-	-	-	-	+	+	-	-	-	+	+	+	-	-	-	-	-	-
Humboldt	-	-	-	-	/	-	-	-	-	+	+	+	+	-	+	-	-	-
Parallax	-	-	-	+	+	-	+	+	-	+	+	+	+	-	-	-	-	-
Tabulator	-	-	-	+	/	-	+	+	-	-	-	+	+	-	-	-	-	+
gFacet	-	-	-	+	+	+	+	+	-	+	+	+	+	+	+	-	-	-

users to move the process on to the subsequent stage of the ISP. In the following, we identify for each of these tasks requirements to support humans in accessing information in the Semantic Web and mark for each approach in Table 1 whether it is fulfilled or not.

The first stage of the ISP, the *task initiation*, begins with awareness of lack of knowledge and leads to a concrete definition of the problem and its relation to prior experience and knowledge. Requirements are: A continuous support of the whole ISP including the problem definition (R1.1) and, based on this definition, suggestions on how to address the information need considering previously operated ISPs (R1.2).

The next stage, the *selection*, includes tasks like the identification and selection of the general topic to be investigated. Requirements are: An overview of all available topics (R2.1), for instance in form of a map that can be zoomed in and out, together with automatic suggestions of topics (R2.2) based on the entered problem description, entered keywords or an auto complete functionality. Both requirements aim at lowering the barrier to start an ISP.

The selection is followed by the *exploration* stage. It includes the investigation of the general topic, locating information and relating it to what is already known. Requirements are: A graphical representation of information that can be understood by the user (R3.1), interaction possibilities that are self-explanatory and easy to use (R3.2), the accessibility of details on demand (R3.3), sorting and paging techniques to handle large datasets (R3.4) and zooming functionalities that are capable of showing information in different levels of detail (R3.5).

The *focus formulation* includes tasks like the identification and the selection of hypotheses that result in the formulation of certain filters and thus allows a focused perspective of the topic. It is rather an iterative process than one that is strictly linear. Requirements are: The interactive and intuitive formulation and change of filters (R4.1), their immediate execution on the data (R4.2), the combination of different filters (R4.3), the possibility to build hierarchical filters (R4.4), the traceability of effects caused by each of them (R4.5) and a possibility to change the focus (R4.6).

After the focus formulation, the *collection* takes place. Tasks are to gather and select information related to the focused topic. Requirements are: Easy mechanisms to select interesting findings (R5.1) and to export the selected information for further use in other systems (R5.2).

The last stage of the ISP, the *presentation*, consists of the task to present the found information. Requirements are: A broad range of opportunities to visualize the findings (R6.1).

Especially obvious in the pattern shown in Table 1 is the lack of all approaches to fulfill requirements regarding the first and the last stages of the ISP. Besides these pre- and post-processing stages, requirement R3.5 is also not fulfilled by any of them. Even so gFacet fulfills more requirements than every other approach listed in Table 1, not fulfilling requirement R3.5 is particularly problematic for our approach since its graph tends to get too large to fit on one screen and thus needs to be scrolled by the users. Therefore an appropriate zooming technique along with a better support for the pre- and post-processing would be highly preferable.

5.1 Conclusion and Future Work

In this paper we described Facet Graphs, a new approach for building semantically unique queries based on the concept of faceted search in combination with graph visualization. In addition to the general advantages of faceted search, the visualization of facets as nodes in a graph allows the direct representation of relationships between nodes by labeled edges and thus a connected presentation of the result set together with all relevant facets on one page. The user can add and remove facets to the graph to produce a personalized interface including even distantly connected and multiple facets that can be used to filter the result set from different user-defined perspectives. All caused filtering effects are color-coded in the graph making them better understandable and traceable for users.

We introduced gFacet, a prototypical implementation of our approach that can query arbitrary SPARQL endpoints (e.g. DBpedia) to access information according to its semantics. We conducted a user study to compare gFacet with Parallax and found out that our tool is especially applicable in scenarios where multiple aspects from different domains need to be integrated in order to find certain information. Such scenarios seem to be particularly interesting for querying the Semantic Web because of its huge variety of domains with each containing large amounts of different classes, objects and properties. This opens up new and outstanding opportunities for users to access information; however, controlling such powerful opportunities remains a challenging task.

Future work includes the integration of appropriate zooming functionalities in combination with a focus and context technique to foster users to retain an overview even when using massive amounts of facets in one graph. Another interesting idea is to provide an opportunity to save especially helpful combinations of facets and share such search interfaces with other users. This would further lower the barrier of acceptance for using gFacet since users can load existing search interfaces that are built by more experienced users and thus do not need to start from scratch.

References

1. Berners-Lee, T., Fischetti, M.: Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor. Harper, USA (1999)
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)
3. Bizer, C., Heath, T., Kingsley, I., Berners-Lee, T.: Linked data on the Web. In: Proc. WWW 2008 Workshop: LDOW (2008)
4. Hearst, M., English, J., Sinha, R., Swearingen, K., Yee, P.: Finding the Flow in Web Site Search. Communications of the ACM 45(9), 42–49 (2002)
5. Hausenblas, M., Halb, W., Raimond, Y., Heath, T.: What is the size of the Semantic Web? In: Proc. I-SEMANTICS'08, JUCS, pp. 9–16 (2008)
6. Schraefel, m.c., Smith, D., Owens, A., Russell, A., Harris, C., Wilson, M.: The evolving mSpace platform: Leveraging the Semantic Web on the trail of the memex. In: Proc. Hypertext 2005, pp. 174–183. ACM Press, New York (2005)
7. Longwell RDF Browser, SIMILE (2005), `http://simile.mit.edu/longwell/`
8. Quan, D., Huynh, D., Karger, Haystack, D.: A Platform for Authoring End User Semantic Web Applications. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 738–753. Springer, Heidelberg (2003)
9. Huynh, D., Karger, D.: Parallax and companion: Set-based browsing for the Data Web (2009)
10. Kobilarov, G., Dickinson, I.: Humboldt: Exploring Linked Data. In: Proc. WWW 2008 Workshop: LDOW (2008)
11. Berners-Lee, T., Hollenbach, J., Lu, K., Presbrey, J., Prud'ommeaux, E., Schraefel, m.c.: Tabulator Redux: Browsing and writing Linked Data. In: Proc. WWW 2008 Workshop: LDOW (2008)
12. Huynh, D.: Nested Faceted Browser (2009), `http://people.csail.mit.edu/dfhuynh/projects/nfb/`
13. Heim, P., Ziegler, J., Lohmann, S.: gFacet: A Browser for the Web of Data. In: Proc. SAMT 2008 Workshop: IMC-SSW, CEUR-WS, pp. 49–58 (2008)
14. Fruchterman, T., Reingold, E.: Graph drawing by force-directed placement. In: Softw. Pract. Exper. 1991, pp. 1129–1164. John Wiley & Sons, Chichester (1991)
15. Gray, J., Bosworth, A., Layman, A., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In: Proc. ICDE 1996, pp. 152–159. IEEE, Los Alamitos (1996)
16. Kuhlthau, C.C.: Developing a model of the library search process: cognitive and affective aspects. Reference Quarterly, 232–242 (1988)