Smeagol: a "specific-to-general" Semantic Web query interface paradigm for novices

Aaron Clemmer, Stephen Davies

University of Mary Washington, Fredericksburg, VA 22401, USA
aclemmer@umw.edu, stephen@umw.edu

Abstract. Most Semantic Web query interfaces let the user give an abstract specification of the desired results (perhaps using facets, or a natural language query.) We introduce the Smeagol visual query interface which, by contrast, guides the user from a specific example to a general result set. Users begin the query process with navigation and exploration activities, building a concrete subgraph of interest from the larger data set. They then generalize this subgraph to find other subgraphs similar in some way to the one identified. Among other advantages, this approach also lends itself quite naturally to querying on instance-based data; i.e., triples in which the predicate is not part of a defined ontology. We provide an analysis of this specific-to-general approach, contrasting it with existing systems. We also present the results of a usability experiment comparing novices' use of Smeagol with that of a standard Linked Data browser.

Keywords: Semantic Web, linked data, user interface, query building, pivot operation, graph visualization

1 Introduction

In the last few years, Semantic Web researchers have begun to produce interfaces that enable novices to pose queries without the use of a formal query language. Some of these applications accept natural language queries; others let the user directly manipulate a graphical representation.

One thing these diverse systems share is an interface paradigm that progresses from the general to the specific. Users give an abstract specification of the desired results, whether by using facets, natural language description, or some other means. Although the mechanism used to express the query varies widely, the user's task is ultimately still to express some variation of the general formula "find resources that satisfy these criteria."

A subtle problem is that this approach often does not mimic the user's thought process. Sometimes the user may begin with an abstract question in mind, but often he does not. Instead of knowing at the outset that he wants to ask "Who are all the famous athletes who dated celebrities?" a user may be browsing a David Beckham page, discover that Beckham dated Victoria Adams, and think, "Interesting! I wonder what other athletes were similar?" Only after

asking that question, in those circumstances, will he discover that Derek Jeter dated Mariah Carey. In other words, a user often does not even realize that he *has* a question until an intriguing concrete example is found.

We believe there may be an advantage to an interface that explicitly enables this process. With such a tool, users could roam and explore a Semantic Web data set without regard to any possible future query. As they browse, they mark out features of interest along the way, building a subgraph which illuminates a small subset of resources and relationships. Then, they generalize from this example in whatever way(s) they choose in order to see analogous resources.

Even when users *do* begin with a question, they may benefit from being able to express that question by means of an example. Rather than having to begin abstractly with a list of types and predicates, users can find a concrete example of what they are looking for, and construct the query "in place." This seems less error-prone, since the user is directly working with the very predicates and graph structure for which they want results, rather than having to describe the desired instances in general terms. This approach also naturally supports querying on *any* predicate, not merely those defined by an ontology. It is possible, of course (see, e.g., VisiNav[6]) to design a faceted interface that exposes non-ontology-based predicates, but it seems that the context in which one first discovered the existence of a predicate is a quite natural place from which to select that predicate and find other examples.

We define the term "general-to-specific" to refer to a query interface (like Humboldt[12] or gFacet[7]) that allows the user to specify abstract criteria for a result set. By contrast, we define "specific-to-general" to refer to an interface that explicitly supports starting with an example and generalizing it to find other similar examples. Put another way, a "general-to-specific" interface is based on reduction: adding criteria to the query progressively narrows down the results from the set of all resources to the desired set of answers. A "specific-to-general" interface, on the other hand, is based on expansion: aspects of a concrete example are progressively generalized to find other results that match a pattern.

To continue the above example, a general-to-specific interface would allow a user to find the class of Footballers (or Professional Athletes) in a data set, then choose from predicates like "dated," "marriedTo," or "inARelationshipWith." The user could form a query based on such predicates, and add additional constraints, such as that the object of the triple must be of a certain type (Celebrity.) This would allow the user to find both the Beckham-Adams and Jeter-Carey query results, but only by beginning with (and knowing about) the abstract types and predicates, and anticipating that there would be some result(s) that satisfied them. By contrast, a specific-to-general interface would allow the user to browse the data set, and upon reaching a subgraph reflecting the Beckham-Adams relationship, immediately request to generalize that relationship to others that followed the same pattern. No top-down selection of classes, predicates, or desired subgraph patterns is required, since these are directly under the user's nose at the time the query is generated. There is also

no need for the user to guess what the "correct" predicates or classes are, since the example immediately in front of the user already contains them.

This paper introduces Smeagol, a query interface that directly supports the specific-to-general paradigm. In J.R.R. Tolkien's mythology, Smeagol was a creature who found something of great value without deliberately setting out to search for it. Similarly, our application allows the user to begin the query process with navigation and exploration activities, building a concrete subgraph of interest from the larger data set. They then generalize this subgraph to find other subgraphs similar in some aspect(s) to the one identified. Our theory is that ordinarily, a novice user's "queries" arise in exactly this way: not as a quest for general results satisfying some abstract criteria, but as a search for other items analogous to a concrete particular.

Note that our work is about identifying, designing, and empirically evaluating a new user interface paradigm. There are many other open questions about querying the Web of Linked Data which we do not address. These include: performance and scalability of complex queries; evaluating queries that span multiple data sets and hence require data integration to satisfy; and discovering relevant data sets. These are all important open problems, and we are aware that others are working on them. They are, however, outside the scope of our research. Our focus is on enabling novice users to effectively pose complex queries against a Semantic Web data set, a challenging task given the complex nature of graph-based data and the difficulty many humans have visualizing and articulating patterns in it.

2 Related Work

There are a variety of user interfaces that enable novices to query the Semantic Web. These differ from search interfaces such as Sindice[16], Swoogle[5], and Falcons[2], whose purpose is to find resources matching a keyword or property, in that they enable the user to answer complex questions expressed as a graph pattern, such as "Who are all of the authors of books published in Germany in the year 1974?"

Within the domain of query interfaces, Natural Language Interface (NLI) systems such as FREyA[4], PowerAqua[13], PANTO[17], and SerFR[1] focus on helping the user find an answer to an *a priori* question, as opposed to supporting an iterative process of domain discovery and query building.

Visual query builder (VQB) systems, such as iSPARQL[15], Semantic Crystal[11], NIGHTLIGHT[14], and SPARQLinG[8], manifest the query as a graph (as Smeagol does), but the top-down emphasis on constructing formal graph patterns from an ontology expects significant knowledge of both the problem domain and SPARQL. Neither NLIs nor top-down VQBs are oriented towards scenarios where a formal ontology does not exist.

Faceted interfaces, such as Humboldt[12], Parallax[9], gFacet[7], and VisiNav[6], are more oriented towards novices than VQB systems. Like Smeagol, they enable novice users to both explore the problem domain and also itera-

tively build a query, particularly when they do not begin with a clear *a priori* question. However, faceted interfaces are intrinsically general-to-specific, because the user starts with a generalized set of resources (perhaps all resources of a given type) that is reduced by selecting facets which filter the set. That is, facets are abstract criteria that narrow the result set.

In addition to letting a user filter results on simple properties, Humboldt and Parallax allow the user to "pivot" to properties of related objects (of different types.) For example, a user querying for automobile manufacturers can pivot to the set of automobiles manufactured by Toyota. This implementation of the pivot concept is powerful, but has three important limitations. (1) Only a portion of the query specification is visible to the user at any given time, due to the selected facets only being displayed on the respective pages they were selected from. This was noted by [7], and we believe that grouping the constraints of the entire query together would place less cognitive burden on the user, since they could then see a unified presentation of the query. (VisiNav[6] is an example of a faceted interface which addresses this limitation.) (2) Not all types of queries (which we term "query topologies"; see Section 5) are supported by these interfaces because they utilize a linear history model (i.e. the past sequence of user pivots.) This limitation was noted by [12] for Humboldt but applies to Parallax as well. An example of a non-linear query that cannot be posed by these interfaces is "What musicians contributed to a 2010 album, and also wrote a book of poetry?" which requires a branching sequence of pivots to specify all relations and facets. (3) When the user wants to view results across multiple pivots, he must visit each relevant page in the history to assemble the results. That is, the user can only view one column of the query results at a time. Smeagol addresses all three of the above concerns by (1) manifesting the entire query in a single display, (2) supporting arbitrary branching topologies, and (3) presenting all query results as a set of tuples in a unified display.

gFacet[7] is a facet graph interface, and so uses the general-to-specific paradigm as all faceted interfaces do. It is similar to Smeagol in that the user's query is represented visually as a graph, but there are several key differences. First, in gFacet the facets are derived solely from ontology[1], so it is not possible to express queries involving arbitrary predicates. Second, it does not provide a facility for viewing all statements made about a given resource, limiting the ability of the user to explore the domain to discover what kinds of queries can be posed. Third, like Humboldt and Parallax, the user cannot see a unified, assembled result set across multiple pivots.[2]

[1] For DBpedia, the facets are `skos:subject` objects of type `skos:Concept`, paired with the predicates relating them back to the subject of the triple whose resources have `skos:subject` as the current facet.

[2] For example, suppose the user has a query graph with three facets: song titles in the category `Songs_written_by_John_Lennon`, producers of those songs who are `LivingPeople`, and record labels of those songs that are `RockRecordLabels`. If the user wanted to know all songs produced by Yoko Ono and their respective record labels, he would first have to select Ono from the list of `LivingPeople`. He would then see the songs produced by Ono and the record labels of those songs, but the

The MashQL interface[10] is essentially a dynamic, hierarchical, form-based query builder. It implements pivots and supports both ontology and arbitrary properties, all composed in a tree-based view. The user progressively specifies graph patterns using dynamically constructed dropdowns whose contents reflect the current context (e.g. if a subject dropdown is set to a particular resource, the predicate dropdown will contain all properties used by that subject). Pivots are expressed through the hierarchical representation of the graph patterns, and the user can then specify which elements will be returned in the results. MashQL largely utilizes the general-to-specific paradigm, as the user specifies a general pattern using subject classes as well as predicates, which may return many results. He may then reduce the results to a specific answer by either adding further relations, or by specifying that a property's subject or object has a particular value. Note that the interface also does support the specific-to-general paradigm to a degree, as concrete concepts can be chosen from dropdowns rather than from an ontology. However, the interface's form-based query building encourages users to think from a top-down perspective, and it does not provide a straightforward way to view all statements made about a particular resource, limiting the exploration required to locate conrete resources.

In summary, these query interfaces comprise numerous powerful features that enable users to pose queries in intuitive ways. However, none of them support the specific-to-general paradigm in the way that Smeagol does.

3 The Smeagol user interface paradigm

Smeagol supports a threefold procedure for building queries. Each step is intended to lead naturally to the next.

1. **Exploration.** Users begin by exploring the data graph, traversing from resource to resource and seeing the statements made about each one. This is similar to most Linked Data browsers, but different from most query interfaces. It enables the user to begin with a familiar navigation task, traversing from concrete resource to concrete resource. The user can inspect the triples involving each resource of interest. During this process of inspection and traversal, the user does not face a burdensome cost in terms of backtracking or reorientation when pursuing casual exploration or encountering deadends.

2. **Subgraph building.** As the user begins to identify an area of interest, he can select particular triples and add them to an "example subgraph." This is a connected subset of the overall Web of Data that reflects a user's current focus. It consists of a handful of specific resources and the relationships between them. Building this subgraph serves two purposes: (1) it lets the user select and highlight only the relevant subset of information out of the overwhelming amount of data in the overall graph, and (2) it forms the basis for a future query.

---

two lists would not be correlated with each other. In order to correlate the two, the user would need to click on each record label resource in turn to determine which songs correspond to it.

We believe this step is particularly important as users transition from today's free-text-based Web to the Semantic Web. In the free-text Web, the content of a single page contains so much context that visualizing its neighbors is not as crucial. But when browsing Linked Data, where each node represents a succinct nugget of information, a user can quickly become disoriented if he cannot visualize the contextually pertinent relationships around it.

3. **Subgraph generalizing.** Finally, the interface assists the user in generalizing his specific example into a query, so that he may view the answer(s) to a question he wants to pose. From the specific example, the user can express that certain concrete subjects and objects are actually the resources that he wants to generalize from; that is, they are the variables of the query. The properties and remaining resources are considered to be the constraints.

Though presented here as a sequence, Smeagol users can naturally move back and forth between these activities as they explore the data graph. Adding more triples to the subgraph moves from activity 3 back to activity 2, and navigating to a resource moves back to activity 1. In this way, queries can be modified, expanded, and refined in an interactive process.

3.1 User Interface

In order to begin exploring a graph of Semantic Web data, the user must specify a URI to use as his starting point. Smeagol provides a simple search interface that utilizes DBpedia's URI Lookup web service[3]. The user types one or more keywords into a search box, which when submitted gives a list of suggested URIs for consideration. The user must select the URI from which he wishes to begin exploring; this URI becomes the first resource in his subgraph. Once the user chooses, he is taken to the main Smeagol interface.

The Smeagol interface (Fig. 1) is divided into three sections. The inspector pane (left) displays all triples involving the user's current resource of interest in an "infinite-scrolling" list. When the user decides to add a particular triple to his subgraph of interest, he can identify it as such by selecting it in the list. Conversely, if the user changes his mind, he can remove it from his subgraph by unselecting it. Smeagol currently has no mechanism for intelligently limiting the number of triples in the inspector; this is a difficult problem, and is a topic for future work.

The query visualizer pane (top-right) displays the user's current subgraph. Selections and unselections made in the inspector pane immediately result in an animated update of the visualization. The subgraph is depicted using a radial layout algorithm. The advantage is one of locality: the resource in the center of the visualization is the one currently most relevant to the user; it is also the resource shown in the inspector pane. The distance from the center resource to another resource reflects the degree of separation between them; a distant resource is usually less important to the user than a direct relation. The user may choose to shift his focus and inspect a more distant resource by clicking

[3] `http://lookup.dbpedia.org/`

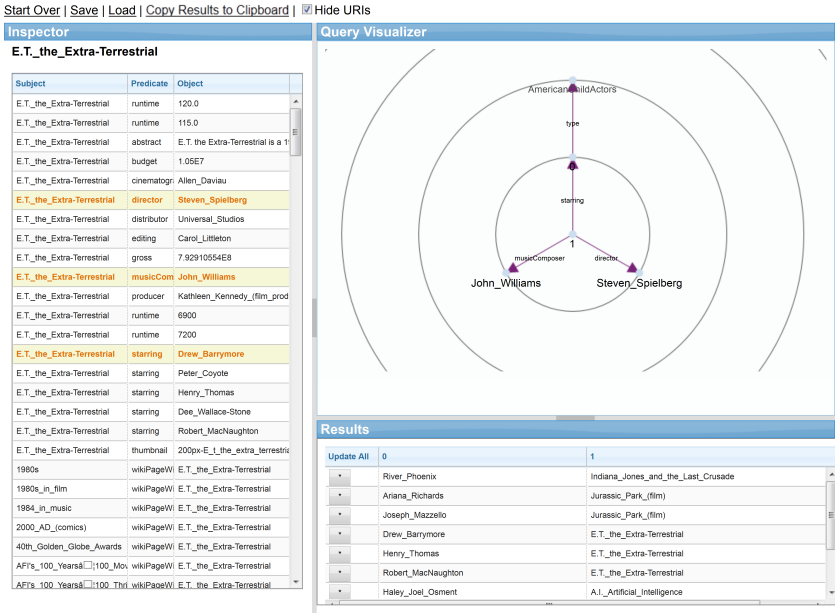

Fig. 1. The Smeagol User Interface.

on its name in the query visualizer; this moves that resource to the center. The radial layout may be panned by clicking and dragging, which provides access to resources too distant to be in immediate view.

A resource in the query visualizer pane can be removed from the subgraph, or "wildcarded" (generalized) via a pop-up menu. Choosing to remove the resource from the query will remove it from the user's subgraph, and also prune the subgraph at that point. This behavior simplifies the process of trimming unnecessary paths of exploration from the subgraph.

Fundamental to the subgraph building procedure, the user can generalize his subgraph at any time by choosing to wildcard a resource; any number of resources may be wildcarded. (When a wildcarded resource is displayed in the inspector, its last associated concrete resource is shown.) The act of wildcarding a resource results in a query being executed.

The results pane (bottom-right) displays a table of tuples corresponding to the query results, which refreshes whenever the wildcarded state of a resource in the query visualizer changes. This state change causes the subgraph to be translated into a SPARQL query, where each variable in the query corresponds to both a wildcarded resource in the query visualizer and a column in the results pane.

There are two operations the user can perform from the results pane. Clicking on a cell in the table replaces the corresponding wildcarded resource in the subgraph with the cell's value. This results in a new query being run and the results pane being refreshed. Clicking on a row's "Update All" button replaces

all wildcarded resources in the subgraph with the respective concrete resources in that row of the results. This effectively replaces the entire query with a chosen concrete example. The benefit of these operations is to aid in the explorative process: if the user discovers an interesting resource in the results, he not only can update the query to restrict on that resource, but can also inspect it and modify the subgraph based on what is seen.

3.2 Example

To provide an example of the process of exploration, subgraph building, and subgraph generalization with Smeagol, consider the following: Suppose the user searches for the 1982 film *E.T.: The Extra-Terrestrial*, and after selecting it, he is taken to the main Smeagol interface. At this point, the center resource in the query visualizer pane would be E.T. (Fig. 2, A), and the inspector pane would show all statements about the film. Browsing through the inspector, the user notices that the film was directed by Steven Spielberg, the music was by John Williams, and it starred Drew Barrymore. Each of these statements interests the user, so he clicks on them in turn in the inspector pane to add them to the query visualizer pane (Fig. 2, B).

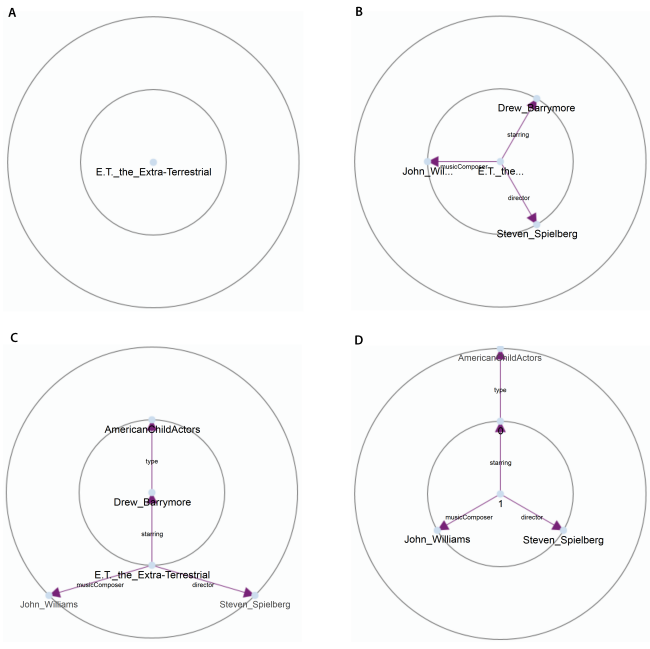

Fig. 2. The query building process.

Deciding to explore Drew Barrymore, the user clicks on that resource in the query visualizer pane, which causes the graph to move the resource to the center and the inspector pane to load information about the actress. Browsing through the inspector, the user sees that she is of type `AmericanChildActors`. Recalling that Barrymore was quite young in E.T., a question occurs to the user: did Stephen Spielberg direct any other films which starred an American who at one time was a child actor? Moreover, did Spielberg and Williams collaborate on any such films? To answer this query, the user begins by adding the `AmericanChildActors` resource to the query visualizer (Fig. 2, C). He then chooses to wildcard both the E.T. and Drew Barrymore resources (Fig. 2, D), as he wishes to see more films and actors meeting the above criteria. After doing so, he is presented with the results as seen in Fig. 1.

4 Architecture

Smeagol is a Rich Internet Application (RIA) that communicates via REST web services to a Java server application. The server application provides proxying to SPARQL endpoints, query result caching, and persistence of users' query graphs.

Smeagol has been tested against DBpedia's SPARQL endpoint, but is not architecturally limited to it. The SPARQL query used by the inspector makes no assumptions about ontology or the presence of any DBpedia-specific resources:

```
SELECT DISTINCT ?subject ?predicate ?object WHERE {
  {<resource> ?predicate ?object}
  UNION
  {?subject ?predicate <resource>}
  FILTER (lang(?object) = "en" || lang(?object) = "")
} ORDER BY ?predicate ?subject
```

The SPARQL queries generated by the query builder from the user's example subgraph are similarly independent of DBpedia. Note that Smeagol could be adapted to data sources other than SPARQL endpoints if those sources could be accessed in a programmatic way, since the user does not deal explicitly with SPARQL.

The query builder currently supports a subset of SPARQL syntax: the user may specify triple patterns and bind variables. More advanced syntax was not necessary for the initial validation of the specific-to-general query paradigm.

Performance of the application is determined by the responsiveness of the SPARQL endpoint and the complexity of the queries the user chooses to construct within the query visualizer. The inspector queries are simple, an advantage afforded by the specific-to-general approach is a reduction in ontology-related queries needed to drive user exploration and query formulation, compared to general-to-specific interfaces. Finally, paging is used to manage large result sets returned by inspector and user queries.

5 Query topologies

In order to identify which kinds of queries Smeagol confers an advantage for, we define the notion of a *query topology*. A query topology describes the general structure of a subgraph in terms of the nodes and the relationships between them, including which of the nodes are wildcarded. It essentially characterizes a certain class of triple patterns.

We depict query topologies visually as a graph of nodes, where "R" indicates a particular concrete resource, and "*" indicates a wildcard. (See Fig. 3.) This is similar to a SPARQL triple pattern – with R's as URIs and *'s as variables – except that we are generalizing from any particular pattern to a category of all structurally similar patterns. Here is one query that conforms to this topology:

```
SELECT * WHERE {
    :Kenneth_Branagh  :starringIn  ?film
    ?film  :writer  :J._K._Rowling
}
```

This query conforms to the topology since it contains one variable present in two triples, each of which also contains one concrete resource. Note that a query topology diagram is an *un*directed graph, since from a complexity standpoint it turns out to be immaterial whether a given node in a triple is a subject or an object. (True, the Semantic Web is a directed graph, since each triple has a subject and object, but the queries "Spielberg directed ?x" and "?x directedBy Spielberg" are equally difficult to pose and to evaluate, which is all we are concerned with.)

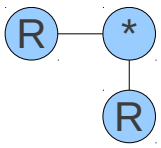

Fig. 3. A query topology.

We characterize a query topology by a three-numbered designation *(n-w-l)*, where n is the total number of nodes in the graph, w is the total number of wildcards, and l is the length of the longest path of consecutively wildcarded nodes, excluding branches. We choose these three measures, omitting other features of the topology (for example, whether two R nodes are attached to the same * node, or to different ones) because they represent likely elements of user difficulty. The number of nodes, the number of generalized nodes, and the "density" with which the generalized nodes are glued together all represent different aspects of a query's complexity. We hypothesize that the third of these three quantities will be particularly significant, since it essentially captures the num-

ber of pivots necessary to execute the query. The topology in Fig. 3 is of class (3-1-1).

Further examples of query topologies, along with their topological designations, sample SPARQL queries, and sample English questions, are in Fig. 4.

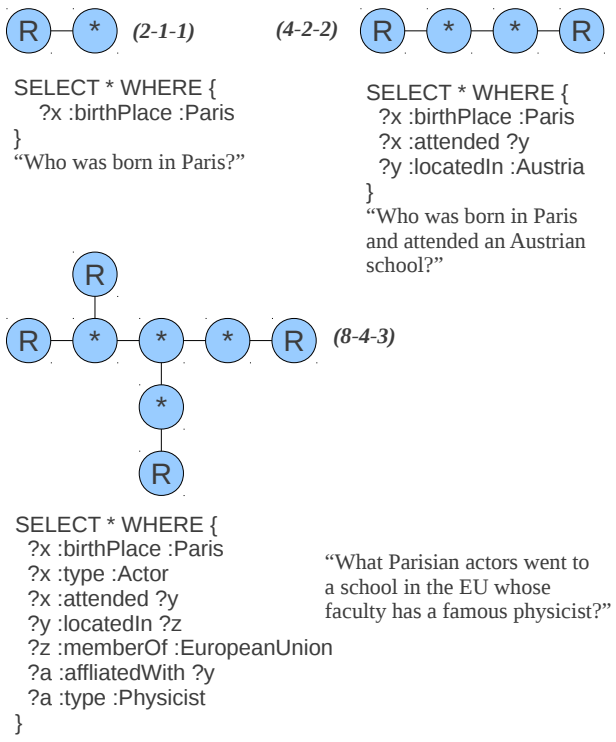

Fig. 4. Some query topologies, with designations, sample SPARQL queries, and sample English questions.

6 Usability experiment

From a user's perspective, Smeagol supports the same kind of open-ended exploration that a Semantic Web browser does, but adds the ability to naturally transition to a query task. Hence, to test whether this added ability yields any benefits, we conducted a usability experiment to compare novices' performance in using Smeagol with that of a Linked Data browser. Our goals were to determine whether the Smeagol subgraph-building and wildcarding paradigm was operable by novice users, and to identify which query topologies it gave an advantage for.

Our subject pool consisted of 43 undergraduate college students, ranging from 18 to 22 years of age and containing roughly an even split between genders. All students were enrolled at the University of Mary Washington during the Fall 2010 semester and were of many diverse majors.

Participants took the one-hour experiment using the Firefox Internet browser on a Windows workstation. They first received a ten-minute explanation of Semantic Web concepts, and a demonstration of the particular navigation tool they were to use (depending on the group; see below.) They then received a packet of materials containing twelve timed query tasks, and were directed to a URL to begin. Both navigation tools used the DBpedia SPARQL endpoint as the sole data source.

Our control group used the Pubby Linked Data frontend[3], which provides a simple HTML interface to the DBpedia SPARQL endpoint. Pubby's interface is similar to many Linked Data browsers in that it manifests each resource as a page, and shows all triples that have that resource as either a subject or object. This is nearly identical to Smeagol's inspector pane, except that full URIs are shown for each resource rather than just its label. Our experimental group used the Smeagol application as described in Section 3.1.

Each item in the packet contained an English language question which the participant was instructed to find the answer to. To begin, the participant was directed to a certain resource that was one of many "answers" to the question. For the control group, this was simply the Pubby page for the starting resource. For the experimental group, the participants began by loading a stored Smeagol subgraph with that node present. For example, item D asked "What famous authors went to Harvard Law School?" and users began the item from the Barack Obama Pubby page (control) or from a subgraph containing the Barack Obama node (experimental). In this way, we hoped to simulate the process described at the beginning of this paper: a user beginning with a concrete example and wanting to generalize it to other examples.

The items were grouped into two sections. The first section contained "scripted" items which explicitly provided the user with the required subgraph. In item D, for example, the control group was told which predicates and resources were relevant (`:almaMater`, `:Harvard_Law_School`, `:occupation`, and `:Author`) and explicitly directed how to find them and what to click on. The experimental group's starting subgraph contained all of these nodes. The participant's task, then, was not to determine how to build the subgraph, but simply to generalize the subgraph: either by navigating and filtering (control) or by wildcarding nodes and noting the results (experimental.)

The second section of the packet contained *non*-scripted items. For example, item J asked "What U.S. Democrats who participated in World War II battles went to a college in the Ivy League athletic conference?" and simply started each participant on the John F. Kennedy page (control) or with a subgraph containing only the John F. Kennedy node (experimental.)

Both sections featured items of various topological classes. Item D (above), for instance, was of class (3-1-1), and item J was (5-2-2). Through this variety

we aimed to isolate the different cases and quantify how well Smeagol improved user performance in various scenarios.

7 Results

The results for each item in our hour-long experiment are presented in Table 1. The "control" column indicates how many users in the Pubby group got each item correct in the time frame allotted (generally 4-6 minutes, depending on item complexity) and the "exp" column shows the same for the Smeagol users. An answer was deemed to be "correct" if its list of resources satisfactorily answered the question for some reasonable choice of predicates. In some cases, more than one reasonable choice existed (such as `:sports` and `:affiliation` for "Ivy League" schools in question J), and so correct answers sometimes varied from one another.

Item	Topology class	Scripted?	control	%	exp	%	p-value
A	(2-1-1)	yes	17/23	73.9%	19/20	95.0%	.100
B	(2-1-1)	yes	19/23	82.6%	20/20	100.0%	.111
C	(2-1-1)	yes	19/23	82.6%	18/20	90.0%	.669
D	(3-1-1)	yes	0/23	0.0%	17/20	85.0%	**<.0001**
E	(4-2-2)	yes	0/23	0.0%	7/20	35.0%	**.002**
F	(2-1-1)	no	17/23	73.9%	18/20	90.0%	.250
G	(2-1-1)	no	20/23	87.0%	20/20	100.0%	.236
H	(3-1-1)	no	0/23	0.0%	15/20	75.0%	**<.0001**
I	(3-1-1)	no	4/23	17.4%	12/20	60.0%	**.005**
J	(5-2-2)	no	0/23	0.0%	13/20	65.0%	**<.0001**

Table 1. Results for each experimental item, including tallies and percentages for each group (control and experimental) who correctly completed the item. Bold p-values indicate statistically significant results (to $\alpha = .05$ by two-tailed Fisher's exact test.)

The first and most obvious finding is that the Smeagol group outperformed the Pubby group on every item. However, this was statistically significant (using Fisher's exact test rather than χ^2 due to small sample sizes) only for items in which the number of nodes is greater than two. A simple thought experiment reveals the likely reason for this. In a (2-1-1) topology – for instance, "Which films did Tom Hanks produce?" (item C) – a Pubby user can navigate to a single page (Tom Hanks) and examine the predicates to find a list of results. On the other hand, in a (3-1-1) topology – for instance, "What famous authors went to Harvard Law School" – a Pubby user faces a nearly hopeless navigation task. Starting from either the "Author" page or the "Harvard_Law_School" page, they can only identify a list of *possible* results. They must then manually navigate to each author (or Harvard alumnus) to determine whether the other criterion

is satisfied. Smeagol, of course, makes such navigational legwork unnecessary through the use of wildcards.

We suspect, but did not verify in this experiment, that the problem for Pubby users would be exacerbated for queries involving a longer chain of wildcards (i.e., those with a larger value of l in the *(n-w-l)* designation.) This effectively multiplies the number of traversals exponentially. For instance, item E – "What Chicago Bears football players went to college in the Big East?" – requires a Pubby user to find players from the Chicago Bears page, but thereafter to face a combinatorial explosion. Each player cannot simply be checked for a property; rather, *all* schools that player was affiliated with must *each* be checked for a property, requiring a second set of multiple traversals. The total size of the problem varies with the average number of triples per predicate, of course, but it quickly becomes unmanageable (to say nothing of time-consuming) even for a disciplined user. We plan to investigate this in future research.

8 Conclusions and Future Work

The specific-to-general query paradigm seems to be understandable by novices and beneficial to them. Equipping users with the ability to identify a subgraph and generalize it allows them to answer a much wider variety of questions than they could with only a standard navigational browser. This appears to be particularly true for queries that reach a certain threshold of complexity, where "complexity" involves both the number of nodes in a subgraph and the number and arrangement of wildcard nodes. Also, there is a benefit to users explicitly building and visualizing a concrete subgraph, so that they can better comprehend the immediate context of their inquiry.

Smeagol itself could be improved by enabling quantitative comparisons in queries, and by expanding the set of logical primitives to include unions and "ors." Additionally, we believe there may be benefit to organizing the predicates available for users to choose from by leveraging available ontology – this would complement the strengths Smeagol has in supporting queries built from arbitrary triples.

Having established a baseline against the standard Linked Data browser model, we plan in future work to compare novices' performance with Smeagol versus general-to-specific query interfaces like gFacet[7]. This should help us understand which use cases have the greatest benefit from a specific-to-general model; it is very possible that different sorts of user scenarios are better served by different approaches.

Smeagol is completely open source under the GPL license and source code is available at `http://bitbucket.org/aclemmer/smeagol`. A live demo of the application can be accessed at `http://rosemary.umw.edu/smeagol`.

References

1. Al-Muhammed, M., Embley, D.: Ontology-based constraint recognition for free-form service requests. In: Proceedings of the 23rd IEEE International Conference

on Data Engineering. pp. 366–375 (2007)
2. Cheng, G., Ge, W., Qu, Y.: Falcons: searching and browsing entities on the semantic web. In: Proceedings of the 17th International Conference on World Wide Web at WWW2008 (2008)
3. Cyganiak, R., Bizer, C.: Pubby – a linked data frontend for sparql endpoints. http://www4.wiwiss.fu-berlin.de/pubby/ (2007), http://www4.wiwiss.fu-berlin.de/pubby/
4. Damljanovic, D., Agatonovic, M., Cunningham, H.: Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. The Semantic Web: Research and Applications pp. 106–120 (2010)
5. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. In: Proceedings of the thirteenth ACM international conference on Information and knowledge management. pp. 652–659 (2004)
6. Harth, A., Buitelaar, P.: Exploring Semantic Web Datasets with VisiNav. In: The 6th Annual European Semantic Web Conference (ESWC2009) (2009)
7. Heim, P., Ertl, T., Ziegler, J.: Facet graphs: Complex semantic querying made easy. The Semantic Web: Research and Applications pp. 288–302 (2010)
8. Hogenboom, F., Milea, V., Frasincar, F., Kaymak, U.: RDF-GL: a SPARQL-Based graphical query language for RDF. Emergent Web Intelligence: Advanced Information Retrieval pp. 87–116 (2010)
9. Huynh, D., Karger, D.: Parallax and companion: Set-based browsing for the data web (2009)
10. Jarrar, M., Dikaiakos, M.: A Data Mashup Language for the Data Web. In: Proc. of Linked Data on the Web (LDOW2009) Workshop at WWW2009 (2009)
11. Kaufmann, E., Bernstein, A.: How useful are natural language interfaces to the semantic web for casual end-users? In: Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference. pp. 281–294 (2007)
12. Kobilarov, G., Dickinson, I.: Humboldt: Exploring linked data. In: Proc. of Linked Data on the Web (LDOW2008) Workshop at WWW2008 (2008)
13. Lopez, V., Motta, E., Uren, V.: Poweraqua: Fishing the semantic web. The Semantic Web: Research and Applications pp. 393–410 (2006)
14. Smart, P., Russell, A., Braines, D., Kalfoglou, Y., Bao, J., Shadbolt, N.: A visual approach to semantic query design using a web-based graphical query designer. Knowledge Engineering: Practice and Patterns pp. 275–291 (2008)
15. Software, O.: OpenLink iSPARQL. http://demo.openlinksw.com/isparql/ (2010), http://demo.openlinksw.com/isparql/
16. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: Weaving the open linked data. In: Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference. pp. 552–565 (2007)
17. Wang, C., Xiong, M., Zhou, Q., Yu, Y.: Panto: A portable natural language interface to ontologies. The Semantic Web: Research and Applications pp. 473–487 (2007)
18. Zloof, M.M.: Query-by-example: a data base language. IBM Syst. J. 16(4), 324–343 (1977), http://portal.acm.org.ezproxy.umw.edu:2048/citation.cfm?id=1662134