NITELIGHT: A Graphical Tool for Semantic Query Construction

Alistair Russell
School of Electronics and Computer Science
University of Southampton
Southampton
SO17 1BJ, UK
ar5@ecs.soton.ac.uk

Paul R. Smart
School of Electronics and Computer Science
University of Southampton
Southampton
SO17 1BJ, UK
ps02v@ecs.soton.ac.uk

Dave Braines
Emerging Technology Services
IBM United Kingdom Ltd,
Hursley Park, Winchester,
Hampshire,
SO21 2JN, UK
dave_braines@uk.ibm.com

Nigel R. Shadbolt
School of Electronics and Computer Science
University of Southampton
Southampton
SO17 1BJ, UK
nrs@ecs.soton.ac.uk

ABSTRACT
Query formulation is a key aspect of information retrieval, contributing to both the efficiency and usability of many semantic applications. A number of query languages, such as SPARQL, have been developed for the Semantic Web; however, there are, as yet, few tools to support end users with respect to the creation and editing of semantic queries. In this paper we introduce a graphical tool for semantic query construction (NITELIGHT) that is based on the SPARQL query language specification. The tool supports end users by providing a set of graphical notations that represent semantic query language constructs. This language provides a visual query language counterpart to SPARQL that we call vSPARQL. NITELIGHT also provides an interactive graphical editing environment that combines ontology navigation capabilities with graphical query visualization techniques. This paper describes the functionality and user interaction features of the NITELIGHT tool based on our work to date. We also present details of the vSPARQL constructs used to support the graphical representation of SPARQL queries.

Author Keywords
sparql, visual query system, semantic web, graphical query language, ontology, owl.

ACM Classification Keywords
graphical user interfaces, query formulation, query languages

INTRODUCTION
Information retrieval is a key capability on the Semantic Web, contributing to both the efficiency and usability of many semantic applications. The availability of semantic query languages such as SPARQL [20] is an important element of information retrieval capabilities; however, query developers are likely to benefit from the additional availability of tools that assist them with respect to the process of query formulation (i.e. the process of creating or editing a query). Ideally, query formulation tools should avail themselves of user interaction capabilities that contribute to the efficient design of accurate queries while maximally exploiting the power and expressivity provided by the constructs of the target query language.

Most attempts to support the user with respect to query formulation have focused on graphical or visual techniques in the form of Visual Query Systems (VQSs) [7]. VQSs provide a number of advantages relative to simple text editors. Most obviously, such systems support the user in developing syntactically valid queries: they serve to constrain or guide editing actions so as to militate against the risk of lexical or syntactic errors. Other potential advantages include improved efficiency, understanding and reduced training requirements.

In this paper we introduce a graphical tool for semantic query construction based on the SPARQL language specification [20]. SPARQL is one of a number of query

languages that have been proposed for the Semantic Web. Others include RQL [14] and RDQL [22], although only SPARQL benefits from W3C endorsement. The tool we present in this paper (called NITELIGHT) enables users to create SPARQL queries using a set of graphical notations and GUI-based editing actions. The tool is intended primarily for users that already have some familiarity with SPARQL; the close correspondence between the graphical notations and query language constructs makes the tool largely unsuitable for users who have no previous experience with SPARQL.

The rest of this paper is organized as follows. We first provide an overview of the SPARQL query language. The purpose of this overview is to highlight the target set of constructs that need to be supported by any (fully) SPARQL-compliant Visual Query Language (VQL). The following section (Graphical Query Editor) describes the NITELIGHT tool we have developed to support graphical query formulation. We first present the graphical notations that comprise the elements of the VQL supported by NITELIGHT (a language we refer to as vSPARQL); we then go on to describe the tool itself, describing both its general functionality and support for user interaction. Next we present previous work in the area of graphical query formulation, particularly in the context of the Semantic Web. The emphasis in this section is, not surprisingly, on graphical techniques, particularly those provided by Visual Query Systems (VQSs); however, we also describe approaches based on natural language interfaces. Finally, we describe some directions for future work based on our progress to date.

SPARQL QUERY SYNTAX

SPARQL [20] is a semantic query language that exploits the triple-based structure of RDF to perform graph pattern matching and contingent RDF triple assertion. In this sense it is similar to RDQL [22]; however, SPARQL provides a number of features that are not provided by RDQL [see 4 for a review]. These include:

- the ability to create new RDF graphs based on query variable bindings (this is accomplished using the SPARQL CONSTRUCT form)
- the ability to return descriptions of identified resources in the form of an RDF graph (this is accomplished using the SPARQL DESCRIBE form)
- the ability to specify optional query graph patterns (this allows a user to specify that data should contribute to an answer if it is present in the RDF model)
- the ability to test for the presence or absence of specific triple or graph patterns via the SPARQL ASK query form

SPARQL includes facilities to filter result sets using specific tests, e.g. to test whether or not a particular query variable is bound or unbound. It also includes a number of solution sequence modifiers (ORDER BY, DISTINCT, OFFSET, LIMIT, etc.) that modify the sequence of query solutions returned by a SPARQL query processor. SPARQL is, in summary, a highly expressive semantic query language that compares favorably with other RDF query languages, such as RDQL and SeRQL [see 13]. Figure 1 and Figure 16 provide examples of SPARQL queries.

GRAPHICAL QUERY EDITOR

The development of a graphical tool for SPARQL query formulation necessarily entails the development of a set of graphic notations that support the visual representation of SPARQL query components. Following an analysis of the SPARQL syntax specification [20], we developed a set of graphical notations to support the representation of SPARQL queries. These notations comprise the basis of a SPARQL VQL that we refer to as vSPARQL. In the first half of this section we present some features of this language based on our work to date. The graphical query designer, NITELIGHT, was designed to support the user with respect to the formulation of SPARQL queries using vSPARQL constructs. The second half of this section describes the functionality and user interaction features of the NITELIGHT editor.

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX sem: <http://www.edefence.org/semiotiks.owl#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>

SELECT ?incident, ?type, ?certainty
WHERE
{
    ?incident rdf:type sem:MineIncident .
    ?incident sem:isLocatedIn sem:HelmandProvince .
    ?incident sem:involves ?mine .
    ?mine rdf:type sem:Mine .
    ?mine sem:usesExplosive sem:TNT .
    ?mine sem:hasType ?type .
    ?mine sem:hasTypeP ?certainty .
    FILTER (?certainty > .50)
}
```

Figure 1. SPARQL SELECT Query

Graphical Notations

Because SPARQL queries exploit the triple-based structure of RDF models, graph-based representations comprising a sequence of nodes and links can be used to represent the core of most SPARQL queries, i.e. the basic triple patterns that are matched against the RDF data model. The nodes in this case correspond to the subject and object elements of an RDF triple; the links correspond to RDF predicates.

Basic Triple Patterns

In terms of the vSPARQL language, nodes and links correspond to URIs, literal values or variables (bound or unbound). Nodes are represented graphically as a geometric

object exploiting both color and shape to indicate the node type (e.g. unbound variable). Nodes are also associated with a label that indicates the URI, literal value or query variable represented by the node (see Figure 2).

Links are represented as simple lines. They are also associated with a label that indicates the predicate represented by the link or the name of a query variable. Directional arrows indicate which node represents the subject and which node represents the object in a triple pattern (see Figure 2).

Multiple Triple Patterns

The introduction of multiple triple patterns into a query is represented by the addition of multiple nodes and links (see Figure 3). If there are any shared variables or literal values across the triple patterns, then these are represented using a common graphical node with multiple link connections.

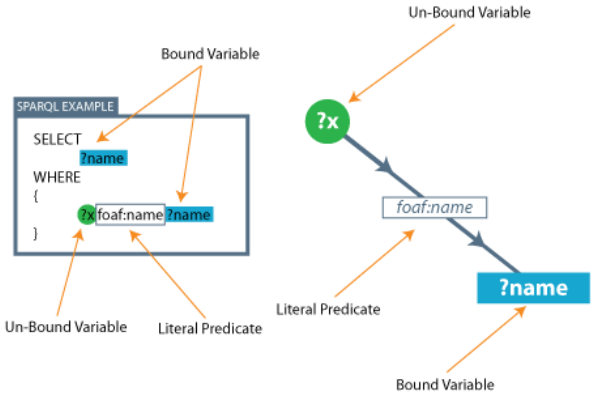

Figure 2. Basic Triple Pattern

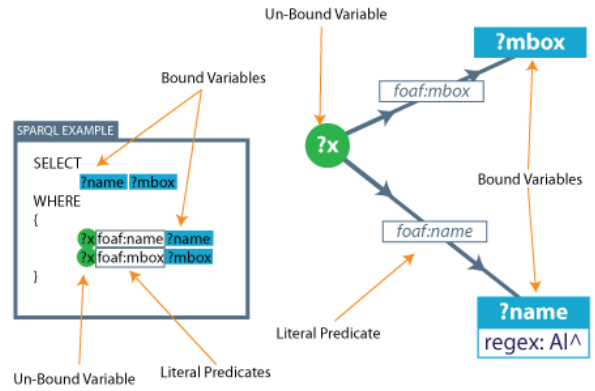

Figure 3. Multiple Triple Patterns

Variable and Triple Ordering

For some SPARQL queries, the ordering of triple patterns and bound variables is important. In order to support the user with respect to the ordering of variables and triple patterns, a numeric value is displayed in the top left corner of both node and link labels (see Figure 4). Any nodes that are duplicated across graph pattern groups will share the same order indicator.

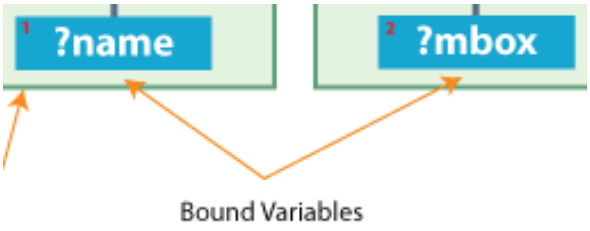

Figure 4. Variable and Triple Ordering

Graph Patterns

In SPARQL, a graph pattern consists of one or more triple patterns that are matched against the entire RDF graph. Graph patterns influence variable bindings because each variable has local scope with respect to the graph pattern in which it is contained. This means that the same variable could be bound to different values in different graph patterns. Using graph patterns means that the triples within a graph pattern are matched against the entire RDF graph and are not affected by any previous graph patterns.

Graphical support for the representation of graph patterns in vSPARQL is accomplished by organizing node-link-node collections into groups (see Figure 5).

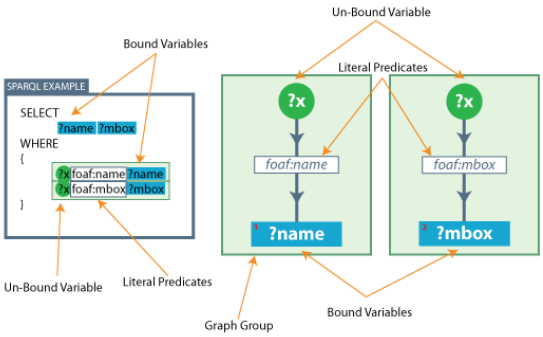

Figure 5. Graph Patterns

When shared nodes appear in multiple graph patterns, the nodes are duplicated graphically. Internally, however, duplicated nodes are treated as the same node.

Optional Graph Patterns

The representation of optional graph patterns is accomplished by visually highlighting the relevant triple groups within the optional graph pattern (Figure 6).

Union Graph Patterns

The visual representation of union graph patterns (i.e. graph patterns where either one of two graph patterns could be considered as part of a query solution) is accomplished by

linking two graph pattern groups with a union label indicator (see Figure 7).

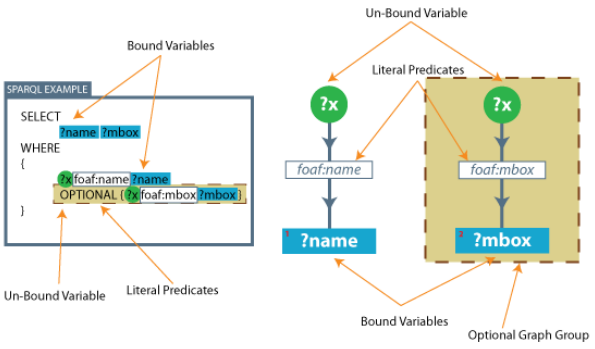

Figure 6. Optional Graph Patterns

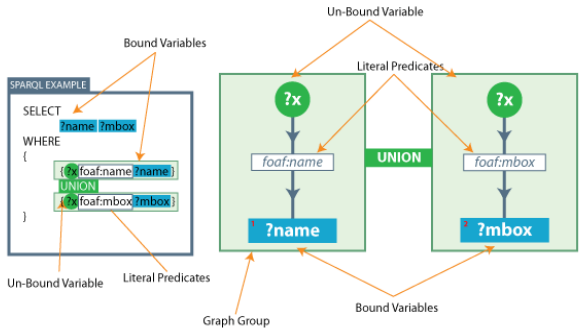

Figure 7. Union Graph Patterns

Graph Specification

The specification of a default RDF graph, or the retrieval of a graph as part of a query, is represented by assigning a variable or literal value to a graph pattern group (see Figure 8).

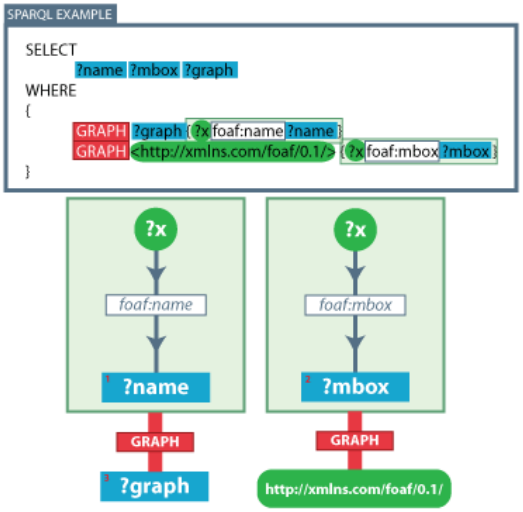

Figure 8. Graph Specification

Result Ordering

SPARQL query results can be ordered by any variable, bound or unbound. To represent this visually, a numerical indicator, similar to the variable/triple pattern order number indicator (see Figure 4), is used. In this case the numeric value appears on the right-hand-side of the variable nodes. The graphical indicator also provides information about the sort order (i.e. ascending or descending) using a directional arrow (see Figure 9). If no indicator is present, the variable is not used for the purposes of ordering the query result set.

Figure 9. Result Ordering

Variable Filter

SPARQL filtering is used to restrict the result sets returned by a query using numerical and regular expressions. The visual representation of a filter expression is based on the addition of a filter field box to the node or link that participates in the filter expression (see Figure 10).

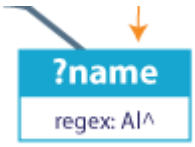

Figure 10. Filter Expressions

Distinct, Limit and Offset

SPARQL also provides functions for retrieving distinct result sets, as well as limiting result sets to a specified number of solutions. These functions are all global to the current query, and can be viewed or changed using the GQE.

Query Editor Prototype

To test and evaluate the features of vSPARQL, we developed a Java-based prototype application, called NITELIGHT, using a combination of Jena [19] and Standard Widget Toolkit (SWT) components.

NITELIGHT (see Figure 11) provides 5 distinct components, each of which works together to give the user an intuitive interface for graphical query creation.

The centerpiece of the NITELIGHT tools is the Query Design Canvas (see Figure 12). The functionality of this component is supplemented by an Ontology Browser component (see Figure 13), a SPARQL Syntax Viewer, a Query Results Viewer and a Quick Toolbar.

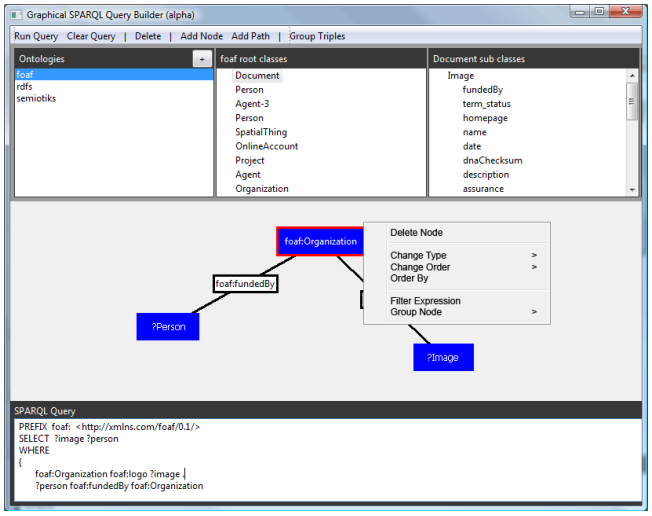

Figure 11. Query Editor Prototype Interface

Query Design Canvas

The Query Design Canvas (see Figure 12) is the centerpiece for user interaction and query construction in the NITELIGHT tool. It provides a canvas for the graphical rendering of SPARQL queries using vSPARQL constructs. It also includes a number of user interaction features that allow users to create and refine semantic queries.

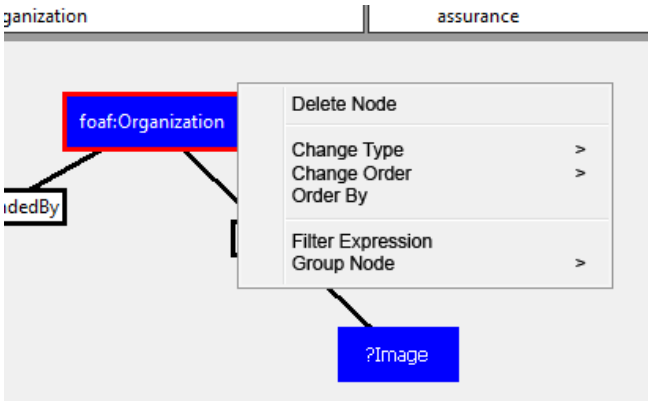

Figure 12. Query Design Canvas

Triples are drawn as two polygon nodes joined with a single link. To allow for more complex queries, the polygon nodes can be moved around the canvas freely, and the canvas itself can be zoomed and panned to view the entire query at different levels of visuo-spatial resolution.

Both the nodes and links are selectable objects that can be edited using either the Quick Toolbar or a context menu. Both the Quick Toolbar and the context menu allow users to define filtering, ordering and grouping information for the selected object. The support for defining filter expressions is currently limited, consisting of a simple text entry form. Our future development plans aim to provide better support for filter expression definition, perhaps using a wizard-like utility.

Ontology Browser

To facilitate the process of query formulation, and to provide users with a starting point for query specification, the NITELIGHT editor includes an Ontology Browser component (see Figure 13). The first column of the Ontology Browser is a persistent list of currently loaded ontologies (the Source Ontologies Column). New ontologies can be loaded into the browser, and the selection of one of the loaded ontologies will result in the enumeration of top-level classes (root classes) in the second column of the Ontology Browser.

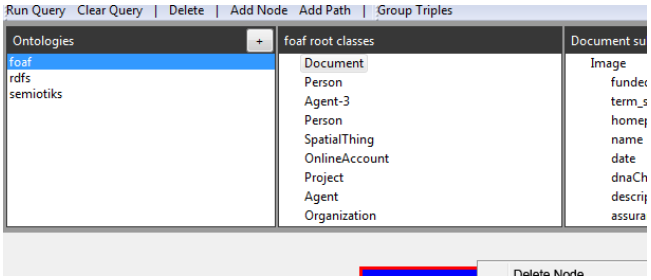

Figure 13. Ontology Browser

The Ontology Browser consists of a series of columns that display the classes and subclasses of an ontology with more abstract classes situated to the left. The column immediately to the right of the Source Ontologies Column is always populated with the root classes of the currently selected ontology. Selecting a class from this column causes an adjacent column to appear to the right of the root classes column. This new column contains the subclasses of the currently selected root class. The pattern of subclass enumeration is repeated as the user progressively selects classes from the right-most column.

The Ontology Browser also provides access to information about the properties associated with each class. In this case, the user can expand a class node in the Ontology Browser to view a list of properties associated with the class.

The Ontology Browser enables a user to drag and drop classes and properties onto the Query Design Canvas. A new node can be created by dragging a class item from the Ontology Browser onto the canvas. A new link can be created by dragging a property from the Ontology Browser and attaching it to a node on the canvas.

SPARQL Syntax Viewer

The SPARQL Syntax Viewer component provides a text-based view of the query that is dynamically updated to reflect any changes made using the Query Design Canvas. At the present time, the SPARQL Syntax Viewer is read-only, i.e. the user cannot edit the SPARQL syntax directly; they must implement any changes to the query via the Query Design Canvas. Future work could explore the possibility of bi-directional translation capabilities in which the user would be permitted to modify the graphical representation of a SPARQL query by interacting directly with the SPARQL Syntax Viewer. This would be of

particular benefit to users who wanted to visualize existing text-based SPARQL queries for the purposes of query refinement or improved understanding.

Query Results Viewer

The Query Results Viewer allows a user to execute a vSPARQL query against any SPARQL endpoint. In the current version of the tool the results are presented in the form of a simple table; however, one could imagine a variety of alternative output formats that might be more suited to the processing capabilities of human end-users. Examples include map-based visualizations, timelines and natural language serializations of query result sets. Since these output formats are often tied to a particular application context, we do not intend to explore the use of these richer visualizations as part of the current development effort.

Quick Toolbar

The Quick Toolbar provides access to commonly used tools for manipulating the Query Design Canvas and its graphical query contents. Example tools include pan and zoom buttons, grouping functions and node editing utilities.

RELATED WORK

A number of approaches to query formulation have been described in the literature. This section provides an overview of some of the approaches that are related to the work described in this paper, or that impact on future extensions to the NITELIGHT query designer tool.

Visual Query Systems

Most attempts to support the user with respect to query formulation have focused on graphical or visual techniques in the form of VQSs [7]. VQSs are systems that use visual representations to depict the domain of interest and express related queries. Often they provide a language, a VQL, which defines both a set of graphical notations to represent query constructs and a compositional semantics for using the notations in the context of query formulation.

Perhaps the best known example of a VQS is the Query-By Example (QBE) system that was developed by IBM in the 1970s [23]. Since then many VQSs have been developed. Catarci et al [7] present a classification scheme for VQSs based on the kind of visual formalism (see [11]) used for query representation. They identify 4 categories of VQSs:

1. form-based systems: these are systems that provide structured representations corresponding to conventional paper-based forms. The aforementioned QBE system was one of the first systems to adopt a form-based approach.
2. diagram-based systems: these are systems that depict relationships between components using simple geometrical figures, such as squares, rectangles, circles, etc. Typically, a diagram-based system will use visual components that have a one-to-one correspondence with specific concepts, with lines between the components representing logical relationships between the concepts.
3. icon-based systems: these are systems that use icons to represent the concepts defined in the domain of discourse. Iconic representations have the advantage that they serve as a pictorial or metaphorical reminder of the concepts being represented; however, VQSs often need to represent entities that have no natural visual counterpart, e.g. an action, command or design specification.
4. hybrid systems: these are systems that comprise two or more of the aforementioned categories.

Of these systems, diagram-based systems tend to be the most popular. In fact, the tool we describe in this paper belongs to this particular category of VQS.

There have been a number of previous attempts to support graphical modes of query formulation in the context of the Semantic Web. Notable examples include OntoVQL [9], SEWASIE [8], SPARQLViz [6], and iSPARQL [2]. OntoVQL [9] is a graphical query language for OWL DL ontologies that maps onto the query language supported by the DL reasoner, Racer. One problem with OntoVQL concerns its expressive power, which is somewhat limited compared to conventional semantic query languages, such as SPARQL. In addition, there is, as yet, no one-to-one correspondence between the visual components of OntoVQL and the elements of a textual query language. This makes OntoVQL somewhat unsuitable as a graphical representational language for SPARQL.

SEWASIE [8] is a graphical query generation environment that co-opts natural language representations and graph-based visualizations of the domain ontology. The user is able to extend and customize an initial query by adding property constraints to selected classes or by replacing classes in the query with another compatible class, such as a subclass or superclass. This process of query refinement is accomplished by selecting terms in the sentential structure of a text-based representation of the query, and then interacting with a graphical visualization of a relevant part of the ontology infrastructure. As the user selects different parts of the query sentence, the graphical visualization of the ontology fragment is updated to reflect the kinds of editing actions that may be performed.

SPARQLViz [6] is a plugin for IsaViz [1] that provides a GUI for the graphical construction of SPARQL queries. SPARQLViz aims to support the user with respect to query formulation, and its aims are therefore similar to those of the work described herein. Significant differences emerge, however, in terms of the approach to user interface design. SPARQLViz relies on a wizard-like interface that presents the user with a sequence of forms such as that presented in Figure 14. This approach differs significantly from that

adopted in the current paper. In terms of Catarci et al's [7] classification scheme SPARQLViz is an instance of a form-based VQS; in contrast, NITELIGHT is an instance of a diagram-based system that co-opts ontology browsing and drag-and-drop functionality with a graph-based visualization of query graph patterns. In the absence of any empirical studies it is difficult to comment on the relative merits of these two approaches (i.e. form-based vs. diagram-based); however, comparisons between SPARQLViz and NITELIGHT could (and should) constitute the basis of future experimental studies.

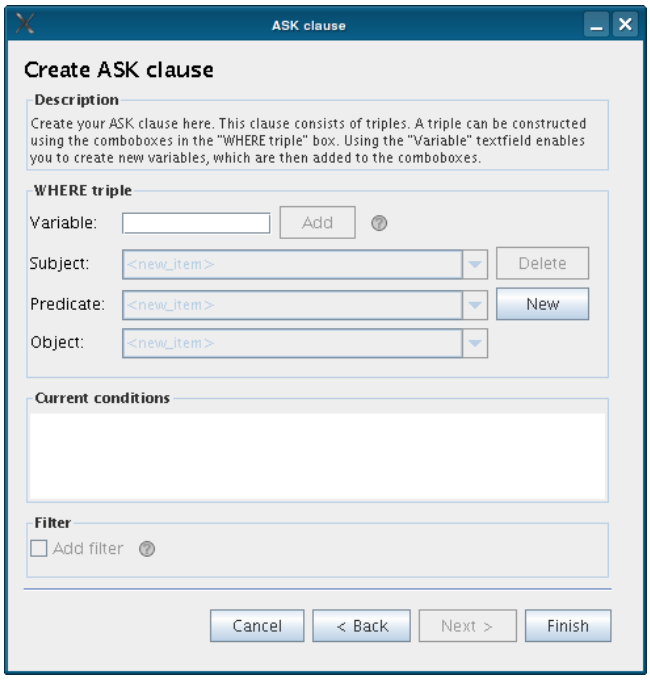

Figure 14. SPARQLViz User Form

One tool that does bear much in common with NITELIGHT is the visual query builder associated with the iSPARQL framework [2] (see Figure 15). The iSPARQL Visual Query Builder supports the user with respect to the specification of all SPARQL query result forms (i.e. SELECT, CONSTRUCT, etc.). It also supports the creation of optional graph patterns as well as UNION combinations of graph patterns in a manner similar to that described for vSPARQL in the present paper. Despite these similarities, differences do exist between the iSPARQL Visual Query Builder and NITELIGHT. Firstly, the visual query language described in this paper (i.e. vSPARQL) is somewhat richer compared to the VQL supported by the iSPARQL Visual Query Builder. vSPARQL supports filter expressions and result ordering as an intrinsic part of its notational syntax, but this information is not available from the set of graphical notations used by iSPARQL (the information is instead provided at the level of editor interface). A second difference concerns the way in which the user is able to access information about target ontologies. The iSPARQL tool relies on a Treeview component that groups ontology elements into 'Concepts' and 'Properties'. NITELIGHT similarly provides access to concepts and properties, but does so using a columnar format that is sensitive to the taxonomic structure of the ontology (see the Ontology Browser section above).

In the absence of empirical studies it is difficult to comment on the significance of the differences between iSPARQL and NITELIGHT in terms of their impact on (e.g.) user approval ratings and query formulation efficiency variables. We would expect the notational differences of the two VQLs to have a relatively minor impact on performance metrics; however, the differences with respect to the tools themselves (e.g. the different ways in which the content of target ontologies is accessed and utilized) may be somewhat more significant. In our experience, understanding the structure of the target ontology as well as the intended meaning of target ontology elements is often the hardest part of the query formulation process.

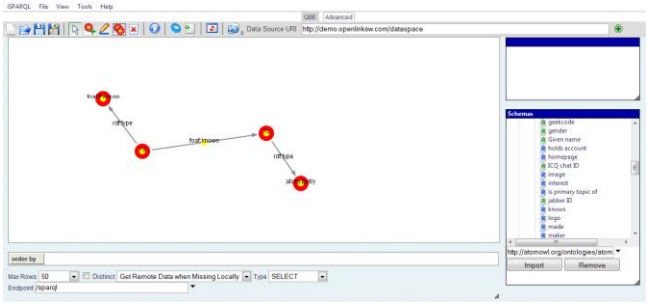

Figure 15: iSPARQL Visual Query Builder

Many of the graphical tools encountered in the literature do not aim to support an underlying text-based language. OntoVQL, for example, does not aim to support query formulation with regard to any specific textual query language (although it does have a partial mapping to nRQL). The tool we describe in this paper does aim to support a specific query language and this motivates a distinction between the current work and some previous studies. We suggest the term Graphical Query Construction System (GQCS) be used to selectively refer to systems that support the visual construction of queries expressed in some other, textual, query language. Systems of this type form a subset of the systems described as VQSs by Catarci et al [7].

Natural Language Query Interfaces

Natural language interfaces provide an alternative to graphical methods of query formulation. These interfaces enable a user to formulate a query using natural language expressions, and they therefore obviate much of the difficulty that novice users may have in terms of creating syntactically valid semantic queries. There have been a number of attempts in the database community to develop systems that use natural language interfaces to support information retrieval [3]. In the context of the Semantic Web, Controlled English interfaces have been used to support information retrieval from semantic repositories [5,

15]. Other systems, such as Aqua-Log [17, 18], provide a question-answering capability that takes queries expressed in natural language and returns answers derived from query execution against a domain ontology. In contrast to the vSPARQL specification described above, natural language interfaces may be more appropriate to users with little or no familiarity with SPARQL.

Semantic Information Browsers

All VQSs aim to support the user with respect to the deliberate creation of queries. The realization of a user's information retrieval goals need not, however, involve the deliberate creation of queries. In some cases, queries can be created and executed (invisibly) as part of an ongoing sequence of goal-directed browsing actions. Systems, such as mSpace [21], for example, support the retrieval of information based on a set of relatively simple and intuitive user interactions, none of which are specifically geared towards query formulation. The question that arises with respect to such systems is whether they undermine the need for tools that explicitly support the query formulation process: couldn't all information retrieval goals be better supported in a system that conflates query generation with episodes of exploratory activity?

While it is certainty true that not every instance of information retrieval necessitates deliberate query formulation, there are, we suggest, cases were users will want to specify information retrieval requests independent of a user interaction context. This is the case when users want to rapidly (re)use the query for information retrieval in multiple contexts, or when they want to distribute a query to other users of a system for the joint evaluation of common result sets. Explicit query design is also required in cases where the query is particularly complex, for example, in cases involving the evaluation of (multiple) variable bindings or disjunctive graph patterns.

FUTURE WORK

The tool described herein was developed as part of an ongoing research program to support human end-users with respect to information retrieval processes in a Semantic Web context. Our future work in this area consists of three activities: extensions to the current tool, development of additional query formulation interfaces and user evaluation studies.

Tool Extensions

The tool described in this paper represents an initial prototype that does not fully support the SPARQL specification. As part of our continued development efforts we aim to extend the functionality of NITELIGHT to include graphical support for all aspects of the SPARQL query language. Of particular interest is the support we aim to provide for the SPARQL CONSTRUCT form. This form of SPARQL query can be viewed as a deductive rule because the query is being used to derive new knowledge from previously asserted facts (see Figure 16). Support for the creation of SPARQL CONSTRUCT queries therefore adds rule editing capabilities to what was originally a tool intended solely for query formulation.

```
CONSTRUCT
{
    ?mine sem:hasType sem:APERS_PI-MI-SR .
    ?mine sem:hasTypeP ".50"^^xsd:float .
}
WHERE
{
     ?incident rdf:type sem:MineIncident .
     ?incident sem:isLocatedIn sem:Afghanistan .
     ?incident sem:involves ?mine .
     ?mine rdf:type sem:Mine .
     ?mine sem:usesExplosive sem:TNT.
}
```

Figure 16. SPARQL CONSTRUCT Query

Another possibility for tool extension relates to use of multiple visual formalisms to represent query elements. As discussed earlier in the paper, Catarci et al [7] present a classification scheme for VQSs that distinguishes between form-based, diagram-based, icon-based and hybrid systems. In its current form, NITELIGHT sits most comfortably in the diagram-based category, although it also includes forms to support query specification and refinement. Subsequent development efforts could, however, extend the range of visual formalisms to include icons (e.g. icons representing types of objects contained in the ontology) and forms (e.g. wizard-like capabilities similar to those described by Borsje and Embregts [6]).

Further extensions and refinements to NITELIGHT include support for creating filter expressions and an ability to update vSPARQL graphical representations based on changes to (text-based) SPARQL queries.

Additional Interfaces

As can be seen from our discussion of related work in this area, there are multiple methods of supporting the human end-user when it comes to query formulation. In addition to the examples presented above (i.e. wizards, QBE systems, graphical designers and natural language interfaces) we can also envision systems providing a range of intellisense, code-completion and syntax checking capabilities, similar to those seen in conventional code-editing environments. One potential direction for future work is therefore to provide a syntax-editing capability that supports expert SPARQL users with respect to the creation and specification of textual queries.

Another type of interface is provided by the use of Controlled English [5, 15] and natural language question-answering systems [17, 18]. These types of systems might be particularly beneficial for novice users who are unfamiliar with semantic query languages. In terms of

extending the capabilities of our current tool with respect to these additional interfaces we aim to develop a natural language query formulation system that implements a similar functionality to that provided by systems such as Aqua-Log [17, 18]. A key difference from the work undertaken with respect to Aqua-Log relates to the serialization of sentential query structures to valid SPARQL queries. At present it is unclear how best to implement this capability. One possibility is to constrain user input using an ontology-specific query grammar; another is to adopt a strategy similar to that seen in the SEWASIE [8] system, wherein the user can progressively select terms in a natural language query and substitute these terms with more specific or general terms based on the domain ontology. Finally, we could opt for a solution based on a subset of natural English, such as Attempto Controlled English (ACE) [10], in which a user is able to express information retrieval requirements using familiar language constructs. In this respect it is interesting to note that ACE can be automatically translated into the N3-style semantic query language PQL [16]. Moreover, a user evaluation of this approach suggests that it promotes the design of good queries with very good retrieval performance [5].

User Evaluation

At this stage we have not performed any user evaluation studies; however, we aim to undertake such studies in the near future. Specific focus areas for evaluation include the general usability of the tool, the ability of the tool to support users with regard to query formulation and comparative analyses of the tool with other graphical [e.g. 8] and non-graphical [e.g. 5] query formulation interfaces. Of particular interest are proposed comparisons between NITELIGHT, SEWASIE [8], SPARQLViz [6], and iSPARQL [2].

Clearly, there are a number of dependent variables that might be assessed in the context of user evaluation studies. These include:

- Syntactic Validity: the number of syntactic errors made during query formulation.
- Query Accuracy: the extent to which the query returns the right information.
- Query Comprehensibility: the level of comprehension attained by a user about a specific query.
- User Satisfaction: subjective ratings of the user's satisfaction with the tool.
- Query Formulation Efficiency: the amount of time taken to formulate queries.

The initial evaluation of NITELIGHT will be based on our target user community (viz., experienced SPARQL users).

CONCLUSION

This paper has presented a graphical editing environment for the construction of semantic queries based on the SPARQL language specification. The tool, called NITELIGHT, is primarily intended for use by those with previous experience of SPARQL (although it could also potentially serve as a support tool for novice users who aim to acquire SPARQL expertise). NITELIGHT is a type of VQS that specifically supports an existing text-based query language; namely SPARQL. In contrast to the recommendations of some commentators [12] we do not propose to develop a simplified query language for end-users; rather we aim to support end-users with respect to the creation of complex queries using supportive user interfaces and user interaction mechanisms. Our tool is one of growing number of VQSs that are being developed to support information retrieval in the context of the Semantic Web.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K.Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

1 http://www.w3.org/2001/11/IsaViz/ - "IsaViz: A Visual Authoring Tool for RDF"

2 http://demo.openlinksw.com/isparql/ - "OpenLink iSPARQL"

3 Androutsopoulos, I., Ritchie, G. D., and Thanisch, P. Natural Language Interfaces to Databases–An Introduction. *Natural Language Engineering 1, 1* (1995), 29-81.

4 Bailey, J., Bry, F., Furche, T., and Schaffert, S., "Web and Semantic Web Query Languages: ASurvey," in *Reasoning Web: First International Summer School*. Msida, Malta, 2005.

5 Bernstein, A., Kaufmann, E., Gohring, A., and Kiefer, C. Querying ontologies: A controlled english interface for end-users. *Proc. 4th International Semantic Web Conference (ISWC05)* (2005), 112–126.

6 Borsje, J., and Embregts, H., "Graphical Query Composition and Natural Language Processing in an RDF Visualization Interface," in *Erasmus School of Economics and Business Economics*, vol. Bachelor. Rotterdam: Erasmus University, 2006.

7 Catarci, T., Costabile, M. F., Levialdi, S., and Batini, C. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing 8, 2* (1997), 215-260.

8 Catarci, T., Dongilli, P., Mascio, T. D., Franconi, E., Santucci, G., and Tessaris, S. An ontology based visual tool for query formulation support. In *16th European Conference on Artificial Intelligence* (2004)

9 Fadhil, A., and Haarslev, V., "OntoVQL: A Graphical Query Language for OWL Ontologies," in *International Workshop on Description Logics (DL-2007)*. Brixen-Bressanone, Italy, 2007.

10 Fuchs, N. E., and Schwitter, R., "Attempto Controlled English (ACE)," in *1st International Workshop on Controlled Language Applications (CLAW 96)* Leuven, Belgium, 1996.

11 Harel, D. On visual formalisms. *Communications of the ACM 31, 5* (1988), 514-530.

12 Hoang, H. H., and Tjoa, A. M., "The virtual query language for information retrieval in the semanticLIFE framework," in *International Workshop on Web Information Systems Modeling*. Trondheim, Norway 2006.

13 Hutt, K., "A Comparison of RDF Query Languages," 2005.

14 Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., and Scholl, M., "RQL: a declarative query language for RDF," in *11th International World Wide Web Conference*. Budapest, Hungary, 2002, pp. 592-603.

15 Kaufmann, E., and Bernstein, A., "How Useful are Natural Language Interfaces to the Semantic Web for Casual End-Users?," in *6th International Symantic Web Conference (ISWC 2007)*. Busan, Korea, 2007.

16 Klein, M., and Bernstein, A. Toward high-precision service retrieval. *Internet Computing, IEEE 8, 1* (2004), 30-36.

17 Lopez, V., and Motta, E. Ontology-driven question answering in AquaLog. In *9th International Conference on Applications of Natural Language to Information Systems* (2004)

18 Lopez, V., Pasin, M., and Motta, E., "AquaLog: An Ontology-portable Question Answering System for the Semantic Web," in *2nd European Semantic Web Conference (ESWC 2005)*. Heraklion, Greece, 2005, pp. 546–562.

19 McBride, B. Jena: a Semantic Web toolkit. *Internet Computing, IEEE 6, 6* (2002), 55-59.

20 http://www.w3.org/2001/sw/DataAccess/rq23/ - "SPARQL Query Language for RDF"

21 schraefel, m. c., Smith, D. A., Owens, A., Russell, A., Harris, C., and Wilson, M. L. The evolving mSpace platform: leveraging the Semantic Web on the trail of the memex. In *Hypertext 2005* (2005)

22 http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/ - "RDQL - A Query Language for RDF"

23 Zloof, M. M. Query-by-Example: A Data Base Language. *IBM Systems Journal 16, 4* (1977), 324-343.