A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer

Paul R. Smart[1], Alistair Russell[1], Dave Braines[2], Yannis Kalfoglou[1], Jie Bao[3] and Nigel R. Shadbolt[1]

[1] School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, United Kingdom
ar5, ps02v, yk1, nrs@ecs.soton.ac.uk
[2] Emerging Technology Services, IBM United Kingdom Ltd, Hursley Park, Winchester, Hampshire, SO21 2JN, United Kingdom
dave_braines @uk.ibm.com
[3] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
baojie@cs.rpi.edu

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 NITELIGHT, a Web-based graphical tool for semantic query construction that is based on the W3C SPARQL specification. NITELIGHT combines a number of features to support end-users with respect to the creation of SPARQL queries. These include a columnar ontology browser, an interactive graphical design surface, a SPARQL-compliant visual query language, a SPARQL syntax viewer and an integrated semantic query results browser. The functionality of each of these components is described in the current paper. In addition, we discuss the potential contribution of the NITELIGHT tool to rule creation/editing and semantic integration capabilities.

Keywords: sparql, visual query system, semantic web, graphical query language, ontology, owl, rdf, semantic integration, ontology alignment.

1 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 [1] is an important element of information retrieval capabilities; however, query developers are likely to gain additional benefit from 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) [2]. 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 minimize 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 that is based on the SPARQL language specification [1]. The tool we present is called NITELIGHT and it enables users to create SPARQL queries using a set of graphical notations and GUI-based editing actions. The graphical notations supported by NITELIGHT comprise a SPARQL-compliant Visual Query Language (VQL), called vSPARQL, which covers all syntactic elements of the SPARQL specification. The complexity of this VQL makes the tool largely unsuitable for users who have no prior experience with SPARQL; although this does not preclude the use of the tool in contexts where users are attempting to familiarize themselves with SPARQL-related capabilities. In addition, we suggest that the functional applications of NITELIGHT are not necessarily limited to information retrieval, and that the tool could be used for a variety of other purposes (e.g. ontology alignment, information integration, rule creation) which may serve to broaden the user base (see Section 5).

2 Demonstration Ontology

In order to demonstrate the representational and functional capabilities of the NITELIGHT tool (see Sections 3 and 4) we use an ontology that was developed to support the processing of terrorist incident data. The ontology we have developed is called the E-Defence Terrorism Ontology (EDTO) and it draws on previous ontology design work in the area of terrorist incident analysis [3].

The centre-point of the EDTO ontology is, perhaps not surprisingly, the notion of a terrorist attack (see Fig. 1). Multiple types of terrorist attack are represented as subclasses of the edto:TerroristAttack class and, in most cases, these classes are defined, meaning that they are associated with restrictions that define the necessary and sufficient conditions for membership of the class. One such condition is illustrated in Fig. 1. In this case we see the definition of the edto:MiddleEastTerroristAttack class. The class is defined in terms of a terrorist attack that occurs in a spatial (geographic) region that is either the Middle East or the Persian Gulf.

Each edto:TerroristAttack class is associated with a number of properties that provide further information about the attack. In the context of the EDTO ontology, these properties capture information about the spatial and temporal location of the attack, the organizations and individuals involved in the attack, the number of fatalities and causalities associated with the attack, and so on.

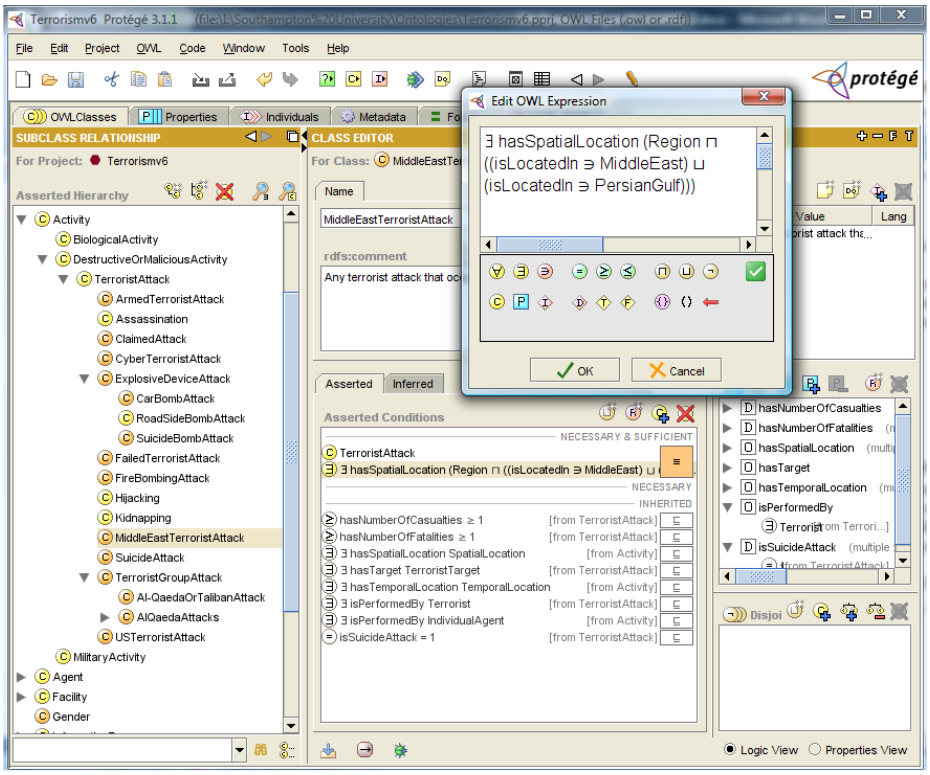

Fig. 1. Protégé-OWL editor showing the taxonomic hierarchy associated with terrorist incidents and the definition of one particular type of edto:TerroristAttack, namely edto:MiddleEastTerroristAttack.

3 vSPARQL Visual Query Language

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 [1], 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.

3.1 Core SPARQL Features

Because SPARQL queries exploit the triple-based structure of RDF models, graph-based representations comprising a sequence of graphical nodes and links can be used to represent the core of most SPARQL queries. The nodes in this case correspond to the subject and object elements of an RDF triple, while the links correspond to the predicates.

vSPARQL uses colour to differentiate between the three types of graphical node (i.e. Bound Variable Node, Unbound Variable Node and Non-Variable Node) that are

used by vSPARQL to represent the subject or object elements of a triple (see Fig. 2). Bound Variables, in this case, represent variables whose value bindings are returned as part of the query resultset, Unbound Variables are variables that are not returned in the resultset (they are used as part of the query execution process) and Non-Variable Nodes are nodes that represent a URI, literal value or blank node. Nodes are associated with a label that indicates the URI, literal value or query variable represented by the node.

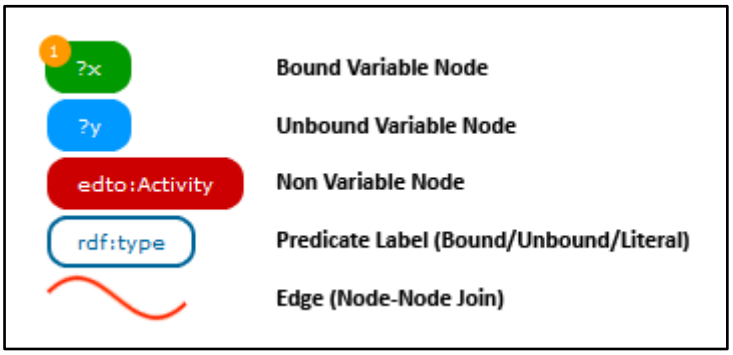

Fig. 2. Core vSPARQL graphical notations.

The predicate part of a triple is visually represented by a graphic link between the subject and object nodes. As with the graphic objects representing the subject or object parts of the triple, the graphical object representing the predicate is associated with a text label that specifies either the URI of the predicate or the query variable (see Fig. 2).

3.2 Triple Patterns

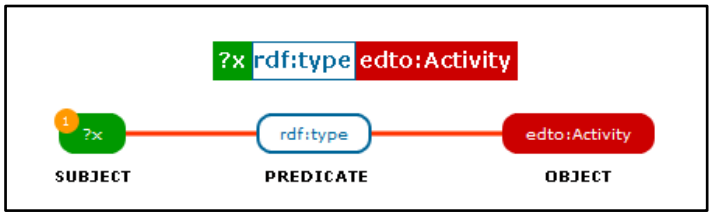

Fig. 3. vSPARQL representation of a basic triple pattern.

The fundamental component of a SPARQL query is the triple pattern. Collections of triple patterns within a query are matched in sequence against the target RDF model in order to establish variable bindings and return query resultsets. Graphically, a triple pattern can be represented by a subject node connected to an object node by a predicate link. An example of this graphical representation using vSPARQL constructs is shown in Fig. 3. The variable '?x', in this case, matches against any object in the EDTO ontology that is an instance of edto:Activity.

Subject and object nodes within the triple pattern are identified by their connection with the Predicate Label: a graph edge protruding from the right hand side of a node into the left hand side of the Predicate Label is the 'subject' of the RDF triple; a graph edge protruding from the left hand side of a node to the right hand side of the Predicate Label is the 'object' of the RDF triple.

3.3 Simple Select Query

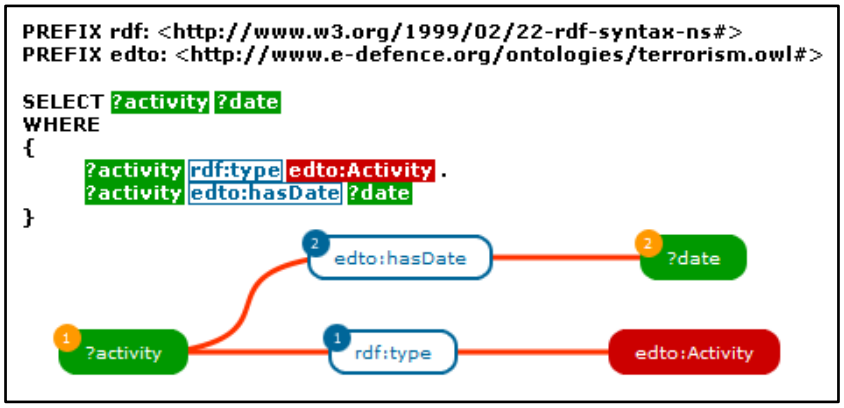

Fig. 4. vSPARQL representation of a SPARQL SELECT query. Note that nodes that occur in more than one triple pattern (e.g.?activity) are represented using a single graphic node.

In vSPARQL a SELECT query comprises graphical representations of the triple patterns that are ultimately matched against the target RDF model (see Fig. 4). The query variables that are returned as part of the SELECT query are represented by the Bound Variable Nodes (coloured green), while the query variables that are used internally as part of the vSPARQL query are represented by Unbound Variable Nodes (coloured blue, but not shown in Fig. 4).

The order in which Bound Variables are returned in query resultsets can sometimes be important. This ordering information is represented in vSPARQL using a numeric value in an orange circle added to the top left of the (Bound or Unbound) Variable Node. The order in which triple patterns appear within the SPARQL WHERE clause is defined by a similar order indicator on the label associated with predicate link (see Fig. 4).

3.4 Graph Patterns

In SPARQL, there are multiple types of graph patterns (e.g. basic graph pattern, group graph pattern, etc.). The query presented in Fig. 4 is an example of a basic graph pattern that comprises one or more triple patterns. Graph patterns influence variable bindings because each variable has local scope with respect to the (basic) graph pattern in which it is contained. This means that the same variable could be bound to different values in different graph patterns.

In SPARQL, a group graph pattern is a collection of two or more basic graph patterns. Graphical support for the representation of group graph patterns in

vSPARQL is accomplished by grouping triple patterns into separate graphical groups (see Fig. 5).

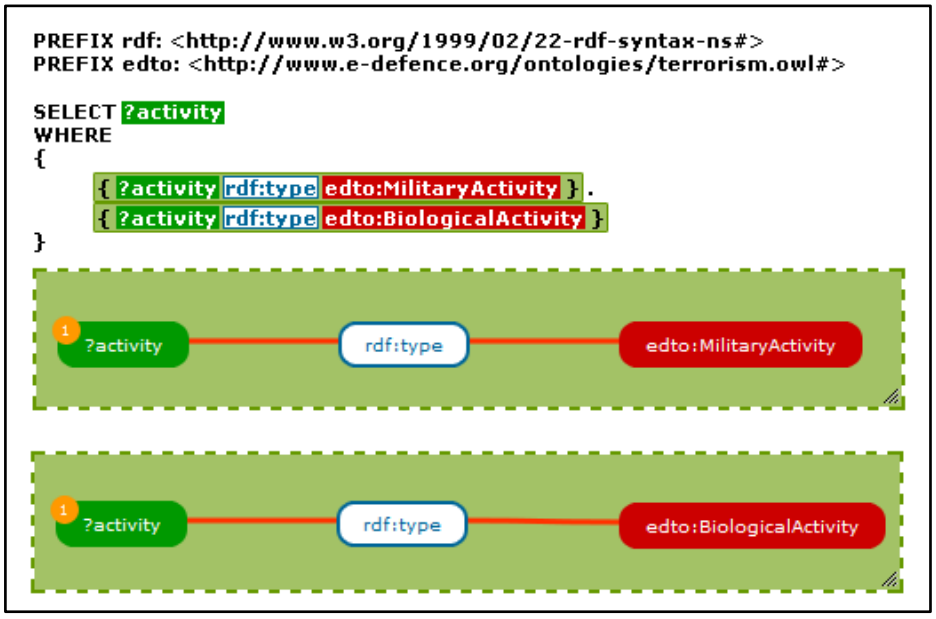

Fig. 5. vSPARQL representation of group graph patterns. Note that in contrast to the strategy adopted with recurring nodes in basic graph patterns, nodes that appear multiple times in multiple basic graph patterns are not represented by a common graphic node; they are duplicated within each basic graph pattern.

Two further types of graph pattern are encountered in SPARQL: optional graph patterns and union graph patterns. Optional graph patterns, as their name suggests, are optional; they allow a user to extend the query solution with respect to additional triple patterns that may or may not match against the RDF model. Union graph patterns (or alternative graph patterns) allow a user to specify alternatives for graph pattern matching. In this case, one of several graph patterns may match the target graph; the failure of one graph pattern to match successfully will not necessarily result in the failure of the query, as a whole, to return a solution. Optional graph patterns are represented in vSPARQL by graphically grouping triple patterns and assigning a unique colour (brown) to the group (see Fig. 6). Union graph patterns are represented using a graphic link between two graph patterns.

The specification of a default RDF graph, or the retrieval of a graph as part of a query, is represented in vSPARQL by using a link to a (Bound/Unbound) Variable (graph retrieval/specification) Node or Non-Variable Node (graph specification) (see Fig. 6).

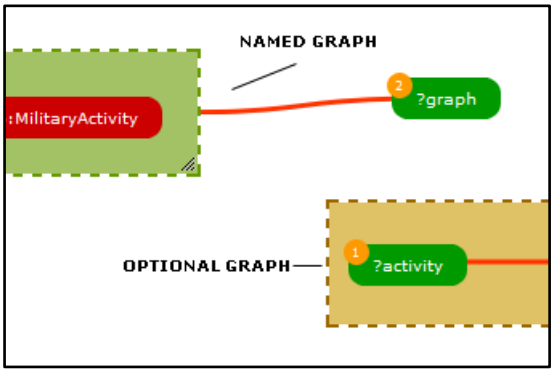

Fig. 6. vSPARQL representation of union graph patterns.

3.5 Solution Sequence Ordering

In SPARQL, the ORDER BY clause establishes the order of a solution sequence, i.e. the order in which the elements of the query resultset are returned. A direction indicator (either Ascending or Descending) specifies whether the query resultset should be ordered in an ascending or descending sequence with respect to the relevant ordering variable. In vSPARQL, solution sequencing is realized by the use of an arrow icon within a (Bound/Unbound) Variable Node (see Fig. 7). The arrow icon uses a numeric value to indicate the order in which variables will be evaluated with respect to the ORDER BY clause; the direction of the arrow specifies the order direction, Ascending (up) or Descending (down).

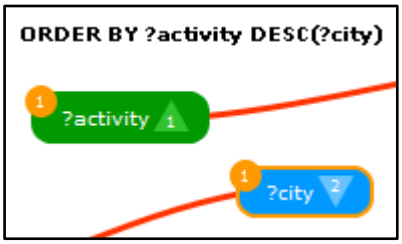

Fig. 7. vSPARQL representation of the SPARQL ORDER BY clause.

3.6 Filtering

SPARQL filtering is used to restrict the resultsets returned by a query using a variety of expressions, e.g. SPARQL operators, SPARQL functions and XPath casting functions [1]. The visual representation of a filter expression is based on the addition of 'Filter Field Boxes' beneath (Bound/Unbound) Variable Nodes (see Fig. 8). Because of the complexity of some SPARQL filters expressions, it is not always practical to display all the terms of the filter expression in the Filter Field Box. Instead, the Filter Field Box displays a short summary of the filter expression which is subsequently expanded in the NITELIGHT tool using a tooltip display mechanism.

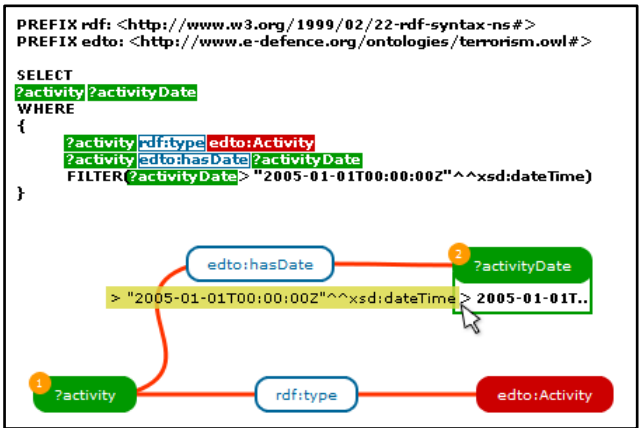

Fig. 8. vSPARQL representation of a SPARQL query that includes a filter on the ?date query variable.

3.7 SPARQL CONSTRUCT Queries

SPARQL has a number of query forms, namely SELECT, CONSTRUCT, ASK and DESCRIBE [1]. All the examples we have encountered so far are of the SELECT query form variety. CONSTRUCT queries are different from SELECT queries because they define both a set of triple patterns to match against the RDF graph, as well as a template for RDF graph construction. The RDF graph generated as a result of query execution is formed by taking the values of variable bindings associated with the triple patterns (in the WHERE clause) and substituting these into the RDF graph template (see [1] for more details).

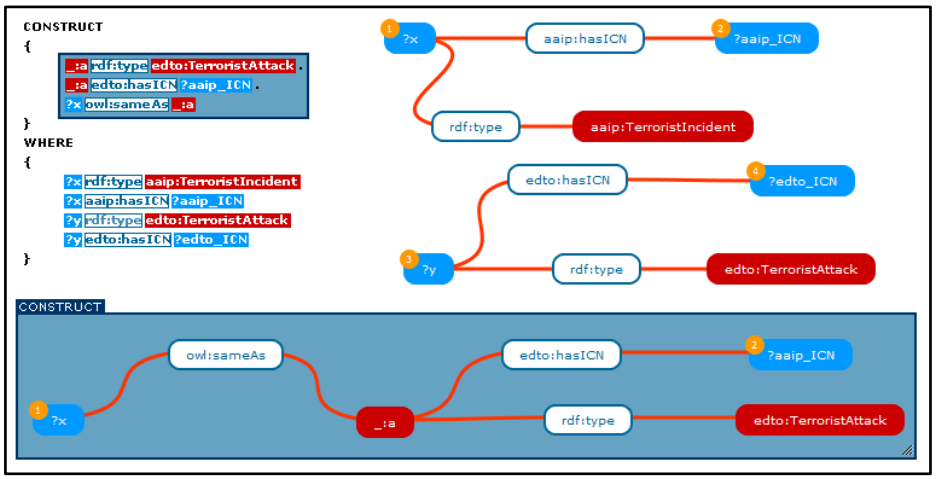

Fig. 9. vSPARQL representation of the SPARQL CONSTRUCT query form.

In vSPARQL, when a CONSTRUCT query is created, the graph pattern that comprises the graph template is highlighted using a colored box (blue in Fig. 9). This

distinguishes the graph template from graph patterns specified as part of the WHERE clause (Fig. 9).

3.8 Other SPARQL Features

There are some features of the SPARQL specification that do not easily lend themselves to a visual representation. These features are supported in the NITELIGHT tool, but they are not part of the vSPARQL specification. They include, ASK and DESCRIBE query forms, as well as DISTINCT, LIMIT and OFFSET solution modifiers.

4 NITELIGHT Tool

The NITELIGHT tool is a Web-based application written entirely in JavaScript. The main user interface (see Fig. 10) has five elements, each of which works together to provide a visually compelling environment for graphical query formulation. The main user interface components are briefly described in subsequent sections.

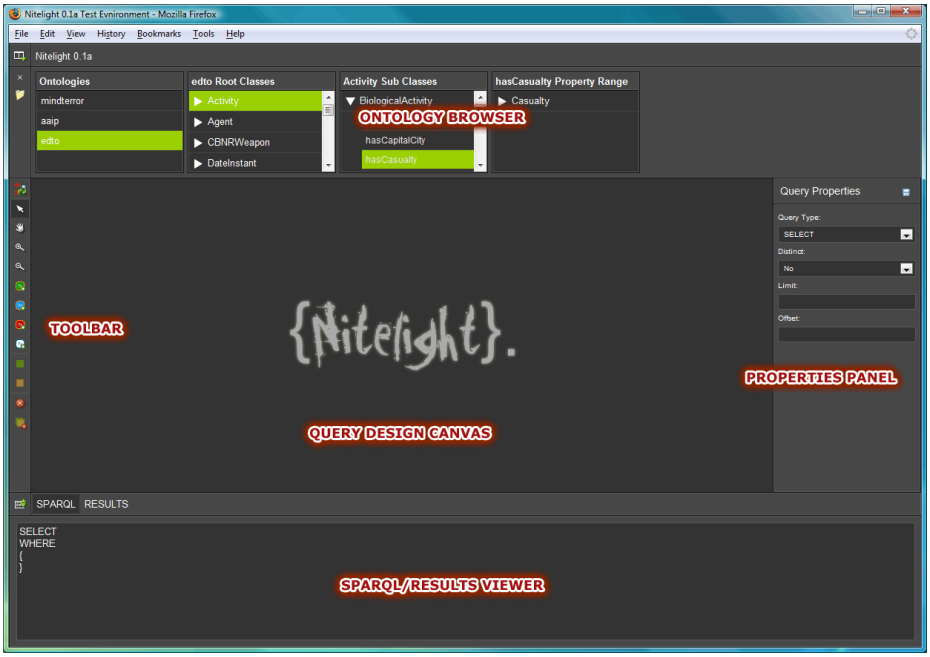

Fig. 10. The NITELIGHT tool main user interface.

4.1 Query Design Canvas

The Query Design Canvas (see Fig. 11) is the center-piece of the NITELIGHT tool. It provides a canvas for the graphical rendering of SPARQL queries using the graphical constructs of the vSPARQL language. Many of the vSPARQL constructs, once rendered on the Query Design Canvas, 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 design canvas itself can be zoomed and panned to view the entire query at different levels of visuo-spatial resolution.

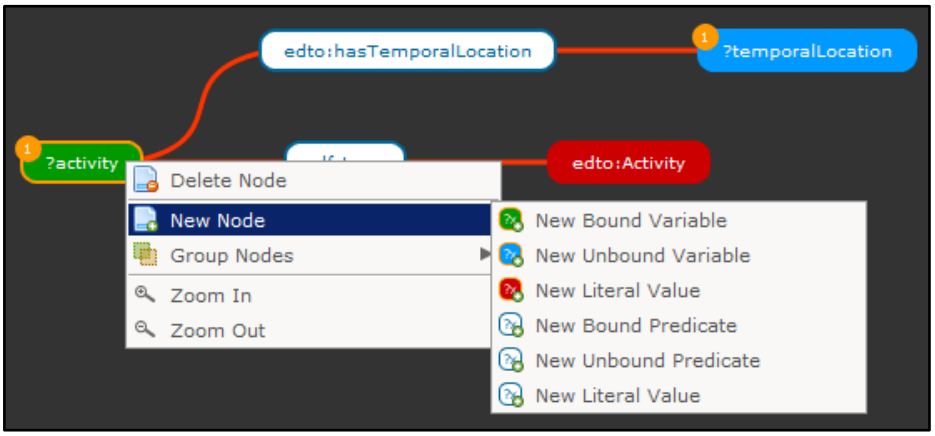

Fig. 11. NITELIGHT Query Design Canvas.

4.2 Ontology Browser

To facilitate the process of query formulation, and to provide users with a starting point for query specification, the NITELIGHT tool includes an Ontology Browser component (see Fig. 12). The first column of the Ontology Browser is a persistent list of currently loaded ontologies (the 'Ontologies' column in Fig. 12). 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 ('edto Root Classes' in Fig. 12). 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. This pattern of subclass enumeration is repeated as the user progressively selects classes from the right-most column.

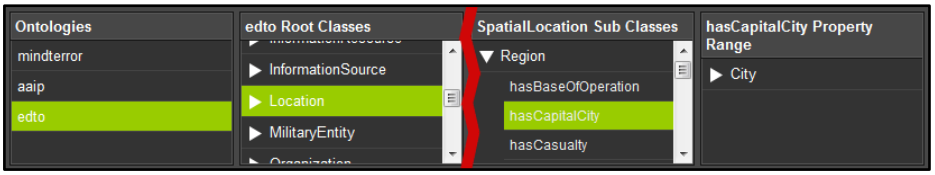

Fig. 12. The NITELIGHT Ontology Browser, showing a path through the EDTO ontology to the 'City' class.

In addition, to enabling users to navigate the class hierarchy associated with the ontology, the Ontology Browser also provides access to information about the properties associated with each class in the ontology. 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 that is already located on the canvas.

4.3 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.

4.4 Properties Inspector Panel

The Properties Inspector Panel allows the user to view and edit the properties associated with a selected vSPARQL object. Common properties include node type, node value, order value etc. However, the exact properties that are displayed in the Properties Inspector Panel ultimately depends on the type of vSPARQL object that is selected.

When no node is selected, the Properties Inspector Panel displays a general set of properties (Fig. 10) that allows the user to change the type of query (SELECT, CONSTRUCT, ASK, DESCRIBE) and to specify the value of solution sequence modifiers (e.g. DISTINCT, OFFSET, LIMIT, etc.)

4.5 SPARQL Syntax Viewer/Query Results Viewer

The SPARQL Syntax Viewer provides a text-based view of the SPARQL query being edited in the NITELIGHT tool. At the present time, the SPARQL Syntax Viewer is read-only, i.e. the user cannot edit the SPARQL syntax directly. Any changes to the SPARQL query must therefore be implemented via the NITELIGHT tool interface. 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.

The SPARQL Syntax Viewer can also be used as a Query Results Viewer to display the results of query execution (Fig. 13)[1]. The functionality of the Results Viewer is limited to displaying the raw output of the query processor (e.g. SPARQL Query Results XML Format[2]). NITELIGHT, however, is easily extensible and the outputs of query execution could be easily co-opted into applications supporting more sophisticated forms of information visualization, e.g. the display of geo-located events on a Google maps display.

---

[1] vSPARQL queries can be executed in NITELIGHT using a user-selected SPARQL endpoint.

[2] http://www.w3.org/TR/rdf-sparql-XMLres/

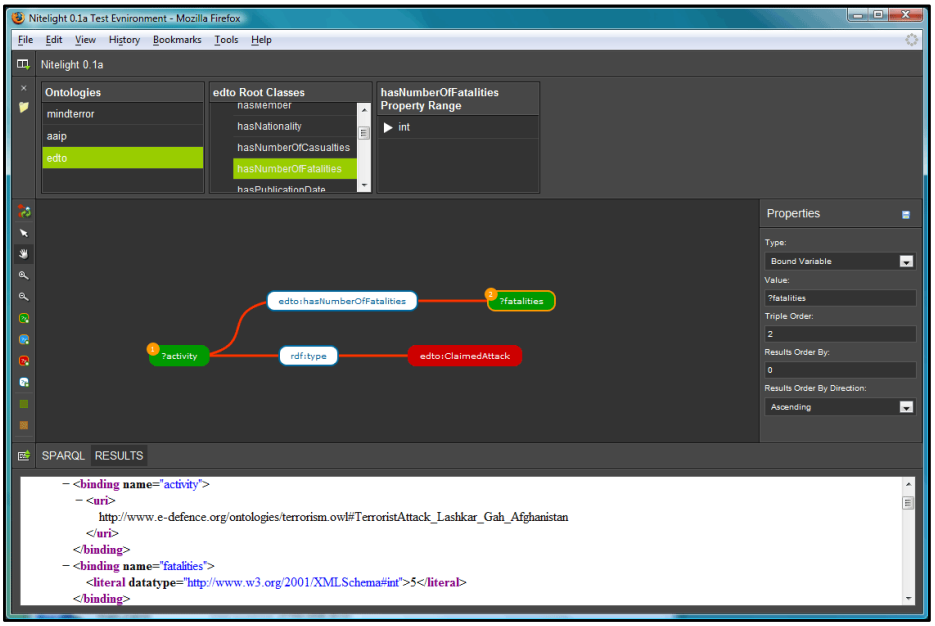

Fig. 13. Query results displayed in the NITELIGHT Query Results Viewer.

5 Additional Application Areas

Thus far, we have described the functionality of the NITELIGHT tool with respect to one particular application area, namely the use of the tool to design semantic queries that retrieve information from a back-end repository. This is clearly the most common use of query languages, semantic or otherwise. There are, however, a number of additional application areas that we are exploring as part of our current and future work. In subsequent sections we focus on just two of these application areas: the use of NITELIGHT to create, edit and visualize domain-specific rules, and the use of the NITELIGHT tool to facilitate the development of information integration and interoperability solutions.

5.1 Rule Creation

The fact that SPARQL supports a number of query forms (i.e. SELECT, CONSTRUCT, ASK, DESCRIBE) means that the functional application of NITELIGHT is not necessarily limited to information retrieval; it can also be used to create ‘queries’ that contingently modify and extend (perhaps multiple) knowledge bases according to the presence of information detected from one or more information sources (sources that may, of course, also subtend multiple nodes of a distributed information network). This use of the tool (to create SPARQL queries that contingently assert new information) is consistent with its use as a rule editor. In such cases, we argue, the tool is being used to capture (and represent) knowledge-rich contingencies that could be modeled using either rules or SPARQL CONSTRUCT

queries. Consider, for example, one inference that can be made in the context of the EDTO ontology (see Fig. 14). In this case, we are using the Semantic Web Rules Language (SWRL) [4, 5] to represent the implied involvement of a terrorist organization in a particular terrorist attack based on the organizational affiliation of individuals responsible for the attack. This contingency can also be expressed in SPARQL using the CONSTRUCT query form (see Fig. 15) and, as such, it can be created, edited and visualized using the NITELIGHT tool.

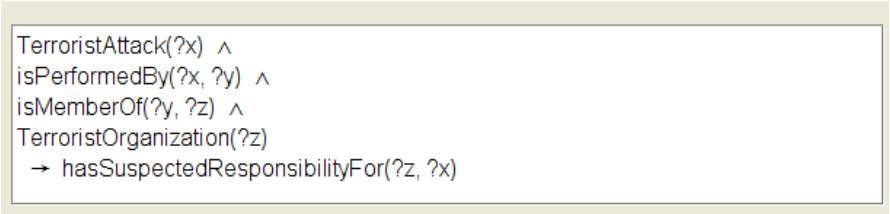

Fig. 14. A SWRL rule representing a knowledge-rich contingency in the domain of terrorist incidents. The rule states that "If a terrorist attack is perpetrated by an individual who is a member of a known terrorist organization, then that organization is, in all likelihood, involved in the terrorist attack."

```
CONSTRUCT
{
      ?z edto:hasSuspectedResponsibilityFor ?x
}
WHERE
{
      ?x rdf:type edto:TerroristAttack .
      ?x edto:isPerformedBy ?y .
      ?y edto:isMemberOf ?z .
      ?z rdf:type edto:TerroristOrganization
}
```

Fig. 15. SPARQL CONSTRUCT query representing the relationship between the perpetrators of an attack, their membership of a terrorist organization and the (inferred) involvement of the organization in the attack.

5.2 Information Integration and Interoperability

One implication of the aforementioned ability to use NITELIGHT as a rule editor is that we can use the tool to represent the semantic mappings or ontology alignments between ostensibly disparate ontologies. In the case of the terrorist incident domain, for example, we see a number of differences in the way in which supposedly common domain-relevant conceptualizations are represented in ontologies (see [6] for one specific example).

One approach to representing semantic mappings between two ontologies, while simultaneously supporting bidirectional information exchange or transfer, is to use a SPARQL CONSTRUCT query [see 7]. Such a query effectively implements an information exchange or information transfer solution that is grounded in ontology alignments that may have been derived using manual and/or automatic methods. A CONSTRUCT query that represents the mapping between the EDTO ontology and a

comparison ontology (called ITO), specifically with respect to the notion of a suicide bomb attack, is represented in Fig. 16 (see [6] for more information about this specific example). Based on this example, one could clearly imagine the future use of NITELIGHT as a tool for expressing ontology alignment information and effecting information exchange/transfer solutions via the execution of SPARQL CONSTRUCT queries.

```
PREFIX edto: <http://www.e-defence.org/ontologies/terrorism.owl#>
PREFIX ito: <http://www.ito.org/terrorism.owl#>
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 xsd: <http://www.w3.org/2001/XMLSchema#>
CONSTRUCT
{
       _:t rdf:type edto:TerroristAttack .
       _:t edto:isSuicideAttack xsd:true .
       _:d ref:type edto:ExplosiveDevice .
       _:t edto:uses edto _:d
}
WHERE
{
       ?x rdf:type ito:TerroristIncident .
       ?x ito:hasType ito:Bombing .
       ?x ito:involvesWeapon ito:Explosive .
       ?x ito:hasVictim ?victim .
       ?victim ito:isFatality xsd:true .
       ?victim rdf:type ito:Terrorist .
       ?x ito:perpetratedBy ?victim
}
```

Fig. 16. SPARQL CONSTRUCT query implementing an information exchange solution for EDTO and ITO ontologies. Note that the query creates a new edto:TerroristAttack instance rather than an edto:SuicideBombAttack. This is because, in EDTO, all terrorist attacks are instantiated from the edto:TerroristAttack class. The actual task of computing the type of OWL individuals in the ontology is delegated to a subsumption reasoner.

6 Usability and User Evaluation

We have not, at the present time, evaluated the tool with respect to particular user groups (details of our proposed user evaluation studies are presented in [8]). As stated at the outset of the paper, the complexity of the representational formalisms used for graphical query construction largely precludes the use of the tool by novice users, i.e. those unfamiliar with semantic query languages. We suspect the tool may be useful to users who are in the process of acquiring SPARQL expertise, but we have not evaluated this claim in the context of controlled empirical studies. The tool has, in general, been favorably received by end users; however, controlled empirical analyses are required to fully evaluate the tool with respect to task performance and user satisfaction criteria (see [8]).

7 Related Work

There have been a number of attempts to support graphical modes of query formulation in the context of the Semantic Web. Notable examples include OntoVQL

[9], SEWASIE [10], SPARQLViz [11], and iSPARQL [12]. One tool that has similar functionality to NITELIGHT is the visual query builder associated with the iSPARQL framework [12]. 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. Despite these similarities, a number of differences exist between the iSPARQL Visual Query Builder and NITELIGHT tool. These include the following:

- **Ontology Browsing.** The iSPARQL tool relies on a Treeview component that groups ontology elements into 'Concepts' and 'Properties'. This differs from NITELIGHT, which provides access to concepts and properties using a column-based ontology browser. NITELIGHT also highlights the domain and range of properties in the ontology browser; iSPARQL simply displays the property name. In general, the NITELIGHT ontology browser provides more information about the loaded ontologies than iSPARQL[3].
- **Graphical Formalisms.** A number of differences exist in the graphical formalisms used to represent query elements. NITELIGHT, for example, uses different formalisms to represent literal and variable nodes.
- **Interactive Query Construction.** NITELIGHT updates the textual representation of a query in an interactive fashion; every change to the graphical representation of the query is associated with a corresponding change to the textual representation.
- **Look and Feel.** NITELIGHT uses Cascading Style Sheets (CSS) to style all components. As such, its look and feel can be easily modified to meet specific design requirements.

The relative significance of these differences in terms of usability criteria and user evaluation outcomes is clearly an important focus area for further research (see [8]).

7 Conclusion

This paper has presented a graphical editing environment for the construction of semantic queries based on the W3C 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 [13] 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.

[3] This is potentially significant because, 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.

Acknowledgements

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. "SPARQL Query Language for RDF", http://www.w3.org/2001/sw/DataAccess/rq23/
2. Catarci, T., Costabile, M.F., Levialdi, S., Batini, C.: Visual Query Systems for Databases: A Survey. Journal of Visual Languages and Computing 8 (1997) 215-260
3. Golbeck, J., Mannes, A., Hendler, J.: Semantic Web Technologies for Terrorist Network Analysis. In: Popp, R.L., Yen, J. (eds.): Emergent Technologies and Enabling Policies for Counter Terrorism. Wiley-IEEE Press, Hoboken, New Jersey, USA (2006)
4. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Benjamin, G., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. World Wide Web Consortium (2004)
5. Golbreich, C., Imai, A.: Combining SWRL rules and OWL ontologies with Protégé OWL Plugin, Jess, and Racer. 7th International Protégé Conference, Bethesda, Maryland, USA (2004)
6. Smart, P.R., Engelbrecht, P.C.: An Analysis of the Origins of Ontology Mismatches on the Semantic Web. 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW'08), Acitrezza, Catania, Italy (2008)
7. Braines, D., Kalfoglou, Y., Smart, P.R., Shadbolt, N.R., Bao, J.: A Data-Intensive Lightweight Semantic Wrapper Approach to Aid Information Integration. 4th International Workshop on Contexts and Ontologies (C&O) hosted by the 18th European Conference on Artificial Intelligence (ECAI'08), Patraz, Greece (2008)
8. Russell, A., Smart, P.R., Braines, D., Shadbolt, N.R.: NITELIGHT: A Graphical Tool for Semantic Query Construction. Semantic Web User Interaction Workshop (SWUI'08) hosted by the 26th CHI Conference (CHI'08), Florence, Italy (2008)
9. Fadhil, A., Haarslev, V.: OntoVQL: A Graphical Query Language for OWL Ontologies. International Workshop on Description Logics (DL-2007), Brixen-Bressanone, Italy (2007)
10. Catarci, T., Dongilli, P., Mascio, T.D., Franconi, E., Santucci, G., Tessaris, S.: An ontology based visual tool for query formulation support. 16th European Conference on Artificial Intelligence, Valencia, Spain (2004)
11. Borsje, J., Embregts, H.: Graphical Query Composition and Natural Language Processing in an RDF Visualization Interface. Erasmus School of Economics and Business Economics, Vol. Bachelor. Erasmus University, Rotterdam (2006)
12. "OpenLink iSPARQL", http://demo.openlinksw.com/isparql/
13. Hoang, H.H., Tjoa, A.M.: The virtual query language for information retrieval in the semanticLIFE framework. International Workshop on Web Information Systems Modeling, Trondheim, Norway (2006)