

# OntoVQL: A Graphical Query Language for OWL Ontologies

Amineh Fadhil, Volker Haarslev

Concordia University, Department of Computer Science & Software Engineering,  
1455 de Maisonneuve Blvd. W., Montreal, Quebec, Canada  
{f\_amineh,haarslev}@cs.concordia.ca

**Abstract.** The database usability experience has shown that visual query languages tend to be superior to textual languages in many aspects. By applying this principle in the context of ontologies, we present OntoVQL, a graphical query language for OWL-DL ontologies. OntoVQL maps diagrammatic queries to DL based query languages such as nRQL, which is offered by the OWL-DL reasoner Racer. OntoVQL hides the complexity of the DL query language from users and allows them to query OWL ontologies with less difficulty. A visual query system equipped with this language has been implemented and is now available. This tool enables users to formulate queries incrementally by having more than one query simultaneously available for getting combined or broken down into new queries. Giving instant feedback in the form of result cardinality is another important feature of the tool that helps guiding users into building meaningful queries.

**Keywords:** ontology, owl, graphical query language.

## 1 Introduction

Ontologies occupy an increasingly important role in the domain of knowledge management and information systems. Although critical work in ontology editing and visualization has been done to assist the ontology engineer, not much has been accomplished for the domain expert or naïve user, i.e. someone who is expert in his domain of study but naïve in the sense of lacking the necessary logical background or technical skills for querying an ontology. Note that ontology querying in this article consists of ABox retrieval. Intuitively, this type of user would like to easily design meaningful queries. However, the actual state of the art translates into submitting a DL-query, written in a query language with a Lisp based syntax, to a DL-reasoner. OntoIQ [7] represents an attempt of solving this issue by providing a query interface offering the basic functionalities of nRQL [6] through some query patterns. Although the latter aids in the querying process, its predefined queries limits the querying power of the user.

The importance of an effective and easy to use query system has been recognized in the database area. In fact, “the database area has proved to be particularly fruitful for applying visual techniques specifically in accessing stored data.” [8]. Since ontologies can be considered as an extended and more powerful way of representing

knowledge than databases, it naturally follows that whatever usability experience was gained from database visual querying would be relevant in the context of ontologies. There has been an extensive work done in that field and many visual query systems (VQS) employing various visual representations and interaction strategies have been proposed in the literature. A VQS is composed of a visual query language (VQL) for pictorially expressing queries and of some functionality for facilitating the human-computer interaction [8]. A VQL is a subclass of Visual Languages that serves for data extraction from databases. Historically, database interfaces have moved from being textual (ex. SQL) to form-based, diagrammatic (ex. Gql [12]) and iconic. Many VQLs took advantage of the graph representation of databases to express queries since queries are graph patterns that are searched in the database graph [13]. There exists some work that has been done towards visually querying ontologies but these attempts are not designed for querying OWL ontologies and they are not readily available for use. However, there exists some work done for ontology based information seeking. For instance, some visual query interfaces (ex. SEWASIE [14]) were developed to allow information retrieval from heterogeneous data sources through an integrated ontology. This paper presents OntoVQL, a formal graphical query language for OWL ontologies that is an effort to apply the principles of a formal VQL. This work is an extension to our previous work, GLOO [2] which mainly consisted of a design study but due to some practical implementation/time constraint issues along with a few design imperfections, some of its properties were dropped and others modified and thus OntoVQL emerged as a new better version of GLOO.

## 2 Presenting the VQL

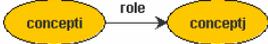
### 2.1 Visual Notations

Since VQLs are mainly based on the idea of directly manipulating database visualization as a graph by selecting the parts that are to be included in the query, then by analogy, given that the TBox of an ontology can also be visually represented by a graph, it is logical to consider that in order to formulate a query for ABox retrieval, one has to select “parts” of the TBox, i.e. concepts and roles, as components of the query. The query is thus composed of graph components and therefore can be formulated as a graph on its own. This would be one way of understanding the logistics behind viewing an ABox query as a graph. The other way is based on the nature of ABox querying itself. For instance, the starting point for ABox retrieval is the extraction of all instances of a concept. Then, one step further from getting a set of individuals based on a concept name is to learn how these individuals are related with other individuals by the mean of edges. In that sense, it is natural to construct query constraints in the form of a graph.

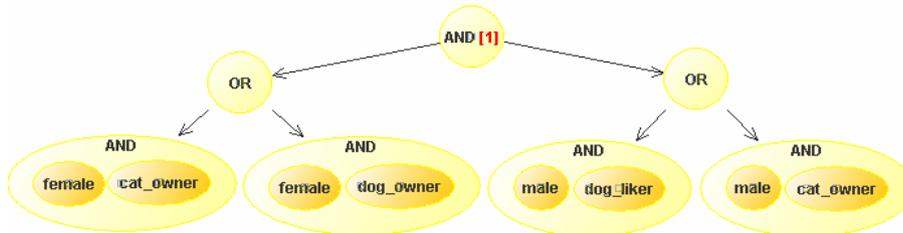
The visual notations of the basic elements of OntoVQL illustrated in Table 1 partially follow the conventions described in [1]. Concepts are represented by their name inside of a filled oval whereas individuals are represented by their name inside of a filled rectangle. The oval with ‘?’ is called an “unknown concept” and is equivalent

to owl:Thing? from the OWL language since it represents the instances that are not restricted to belong to a specific concept and can be of any type. Roles are most conveniently represented by the role name and an arrow going from one entity to another. Note that what is at stake in this representation is only the shape whereas the color is of no importance.

**Table 1.** Visualization of the basic elements of OntoVQL.

<concept>	
<unknown concept>	
<individual>	
<role>	

The operators used for assembling building blocks into a query are intersection and union. There are two kinds of intersection and union operators: AND/OR groups and AND/OR nodes. A group is represented by a circle labeled with the AND/OR keyword and encloses the intersected or unified concepts. In Figure 1, the union operator represented by the OR node is connected with directed edges to AND groups in order to symbolize the union of intersected concepts. Similarly, intersection can be applied on OR groups by connecting them to an AND node. The alternation between union and intersection operators takes the form of a tree where the root is a AND/OR node and the leaves are AND/OR groups.



**Fig. 1.** Alternation of AND/OR operators.

## 2.2 Visual Syntax

In linguistics and computer science, a generative grammar is a formal grammar that provides a precise description of a formal language. The grammar is composed by a set of rules dictating the syntax of the language, i.e., expressing how strings in a language are generated. We have adapted this principle in the context of generating a visual language instead of a textual one in order to generate OntoVQL by a formal grammar or more precisely by an adjusted version of a formal grammar. As a result,

since OntoVQL can be generated by a formal grammar and is also semantically and syntactically unambiguous, it can be viewed as a formal visual query language.

What is of primary importance that has a direct impact on the semantics of OntoVQL resides in its “connectivity syntax”, i.e., the way its components are linked together. This visual syntax can be expressed in terms of rules adopting the same form as a production rule in a formal grammar. Thus, similar to what a typical formal grammar is composed of, our grammar has a finite set of non-terminal symbols: {<Query>, <ROLE>, <AND GROUP>, <OR GROUP>, <AND NODE>, <OR NODE>, <entity>}, a finite set of terminal symbols: {<concept>, <individual>, <unknown concept>, <role>}, each of which has its visual equivalence as shown in Table 1, and a finite set of production rules that are listed below.

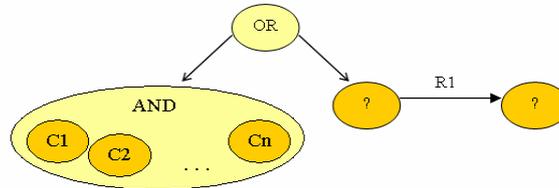
The rules dictate how the visual entities are allowed to be connected together by directed connection edges and roles. The first rule in the grammar below indicates that a query can simply be a concept or consists of more complex components. Among these components, the role is described in the second rule as a binary component where the domain and range are entities that can be expanded to any of the elements listed in the third rule. It is important to mention for the second rule that if a query has the shape of tree as in Figure 1, no role can link any of the query elements under a distinct “branch” of the tree. For example, none of the AND groups in Figure 1 can be linked by a role. The fourth and fifth rules illustrate the AND/OR group containing one or more concepts that are either intersected or unified. Note that an XML like syntax is used for representing the notion of a group by having a start and end tag enclosing one or more concept element. From rule 6, an AND node can be connected to an OR group and/or to an OR node and/or to an unknown concept that is linked to a role. A directed connection edge whose source is the AND/OR node and target is the AND/OR group or the unknown concept links the domain and range entities together.

Grammar generating OntoVQL:

- (1) <Query> -> <concept> |  
                   <ROLE>  
                   <AND GROUP> |  
                   <AND NODE> |  
                   <OR GROUP> |  
                   <OR NODE>
- (2) <ROLE> -> <entity> <role> <entity>
- (3) <entity> -> <concept> |  
                   <unknown concept>  
                   <individual> |  
                   <AND GROUP> |  
                   <OR GROUP> |  
                   <AND NODE> |  
                   <OR NODE>
- (4) <AND GROUP> -> <and group> <concept>\* </and group>
- (5) <OR GROUP> -> <or group> <concept>\* </or group>
- (6) <AND NODE> -> <and node>(<OR GROUP>\* | <OR NODE>\* |  
   (<unknown concept><role><entity>)\*)
- (7) <OR NODE> -> <or node>(<AND GROUP>\* | <AND NODE>\* |  
   (<unknown concept><role><entity>)\*)

As an illustrative example of how the above grammar describes the visual syntax of OntoVQL, consider the query in Figure 2. The query is composed of an OR node

that links an AND group with an unknown concept related to another unknown concept by a role R1. Note that the query's visual syntax corresponds to the 7<sup>th</sup> rule.



**Fig. 2.** An example query generated by the 7<sup>th</sup> rule of the grammar.

### 2.3 Mapping the VQL to the nRQL Language

The expressive power of OntoVQL can be informally described as being able to formulate queries on DL Abox elements (concepts and role assertions) and make use of conventional operators (union, intersection) for building up more complex, refined queries. Our proposed VQL is designed independently of any OWL query language and offers basic functionalities for querying OWL-DL ontologies. This means that there is not a one-to-one mapping between the visual components of OntoVQL and the elements of an OWL query language. Note that a number of these elements have no visual counterpart and therefore, OntoVQL does not match the full expressive power of OWL query languages. For instance, datatypes and negation are features that are not available in OntoVQL but are provided by the textual query language we are mapping to, nRQL. Even though OntoVQL is less expressive, for the simple queries mostly asked by the naïve user, the OntoVQL version of the query exhibits a lower complexity by mainly getting rid of the textual syntax and hiding the query variables of the OWL query. In that sense, we claim that OntoVQL is less complex than an OWL query language.

**Table 2.** Example of a graphical query and its equivalence in nRQL.

<p>Query 100 (2)</p>
<pre>(retrieve ( \$?x2 \$?x3 \$?x1 )   (and ( \$?x2  http://cohse.semanticweb.org/ontologies/people#animal      ( \$?x1  http://cohse.semanticweb.org/ontologies/people#animal        (or ( \$?x3  http://cohse.semanticweb.org/ontologies/people#woman          ( \$?x3  http://cohse.semanticweb.org/ontologies/people#man )       ( \$?x3 \$?x1  http://cohse.semanticweb.org/ontologies/people#has_pet          ( \$?x3 \$?x2  http://cohse.semanticweb.org/ontologies/people#likes )))     )   ) )</pre>

It is possible to translate the visually expressed query semantics into a written version using the query language of your choice. In our case, the translation is done using nRQL. nRQL, its syntax is implemented by an optimized OWL-DL query processor known to be highly effective and efficient. A nRQL query is composed of a query head and of a query body. The query body consists of the query expression. The query head corresponds to the projected variables, that is, variables that are mentioned in the body and will be bound to the result. A tag number in brackets visually represents this variable projection. For instance, from the above query, there are three variables in the head that correspond to the three tag numbers. This “tagging” is not inherently part of OntoVQL as such but is provided for the purpose of integrating the projection feature into the VQS. Note that the “tagging” process is not up to the user and is automatically taken care of by the system.

Although no AND operator is present in the visual query, all the elements in the corresponding nRQL query are intersected. This is an implicit conjunction in contrast to the explicit one that is visualized by an AND node/group. Therefore, outgoing roles from an entity as well as domain and range specification for a role requires the use of conjunction in nRQL.

In order to be able to combine or break down queries, it must be possible to visualize more than one query simultaneously, each query having its own scope of variables. When mapping to nRQL, each concept must be mapped to a unique variable. In the case of AND/OR groups, the concepts inside the group must be mapped to the same variable. Therefore, in the nRQL translation, the concepts inside the OR group are both mapped to x3 whereas the animal concepts are mapped to different variables x1 and x2.

The reason why concepts involved in an AND/OR operation must be mapped to the same variable is because this has a direct impact on the result. For example if concepts C3 and C2 were intersected, then the nRQL query body will include (and (x1 C3) (x1 C2)) which results into binding x1 to those individuals who are C3 and C2. If distinct variables were used instead, then the answer would be those individuals who are C3, plus those individuals who are C2 but not already mentioned for C3. Hence, the semantics changes into adding up C2 and C3 individuals instead of intersecting them. This is why variable mapping is crucial when translating the visual query into nRQL.

## 2.5 Presenting the visual query system

Having an actual implementation of OntoVQL helps to evaluate its usefulness. Figure 2 shows a screenshot of the system’s main window that is split into two parts, the information tabs and the query tabs. The information tabs allow viewing the list of concepts, roles and individuals whereas the query tabs provide the query formulation pane, the query translation as well as the results.

Before starting to use the system for querying, the user must load some OWL ontology. Once the OWL file is loaded, the user can drag and drop concepts, roles or individuals into the query pane. Each dropped element becomes a distinct query and these simple queries can then be combined by using roles, connection edges, AND/OR groups and nodes. Only permitting those connections that follow the

grammar rules enforces the connectivity syntax. Thus, the user can only formulate legal queries. Deleting connection edges and roles allows breaking down complex queries into simple ones. The undo/redo pattern is implemented to facilitate query formulation. It is also possible to save queries into an XML file and load them later for further use.

Each query in the system is identified by a query number that uniquely distinguishes it from the others. One important aspect of the tool is the query preview that indicates the number of tuples returned in the result. The number next to the query identifier indicates the result cardinality. Whenever a new query is created, it is instantly evaluated and its preview set to the obtained number of tuples. Giving instant feedback plays a significant role in guiding the user into formulating meaningful queries. For instance, it would be pointless to combine concepts in an AND group when their preview is zero.

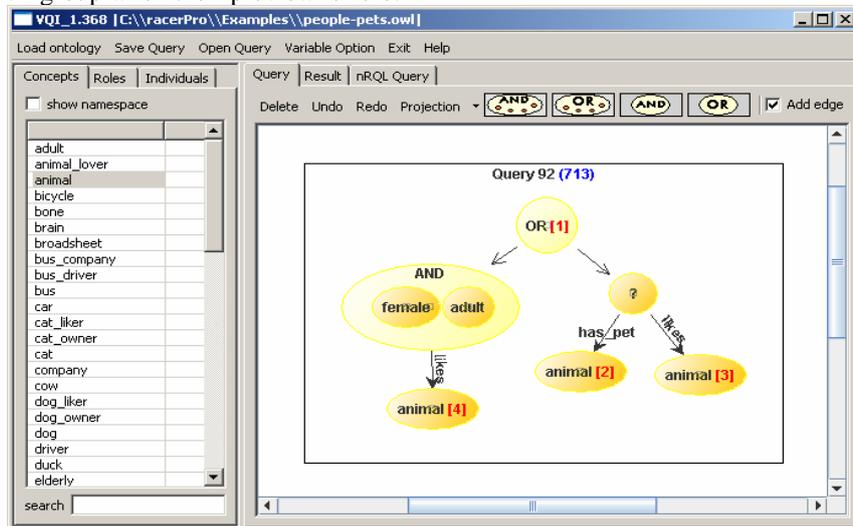


Fig. 2. Screenshot of the actual implementation of OntoVQL.

### 3 Conclusion

In conclusion, we proposed OntoVQL as a visual query language for OWL-DL ontologies and have implemented a tool for its practical use. OntoVQL is meant to be a better alternative over the traditional OWL query languages for the naïve user. In fact, it provides numerous advantages over a textual query language such as nRQL by eliminating its textual syntax with visual simplification, not allowing syntactic errors through the user interface (UI); simplifying the breaking down and merging of queries; and assisting users in the querying process through the system features such as providing immediate feedback with result cardinalities. The implementation of the UI has been completed and is available for download

([http://users.encs.concordia.ca/~f\\_amineh/](http://users.encs.concordia.ca/~f_amineh/)). A usability test for evaluating the tool's efficiency is underway and its results should be posted as soon as they are ready.

**Acknowledgments.** This work was supported in part by Genome Quebec, Natural Sciences and Engineering Research Council (NSERC) of Canada and by ENCS, Concordia University.

## References

- 1 Gaines, R.B.: An Interactive Visual Language for Term Subsumption Languages. In Proc. Of the 12<sup>th</sup> Int. Joint Conf. of Artificial Intelligence (IJCAI'91), pages 817-823, 1991.
- 2 A. Fadhil and V. Haarslev, GLOO: A Graphical Query Language for OWL ontologies. OWL: Experience and Directions 2006, Athens, 2006. Catarci, T., Costabile, M. F., Levaldi, S., Batini, C.: Visual Query Systems for Databases: A Survey. Journal of Visual Languages and Computing, 8(2), 215-260, 1997.
- 3 Larkin I., Simon, H.: Why a diagram is (sometimes) worth then thousand words. Cognitive Science, Vol. II 1987, pp:65-99.
- 4 Catarci, T., Dongilli, P., Mascio, T.D., Franconi, E., Santucci, G., Tessaris, S. : An Ontology Based Visual Tool for Query Formulation Support. In ECAI, pages 308-312, 2004.
- 5 Haarslev, V., Moeller, R., Wessel, M.: Querying the Semantic Web with Racer + nRQL. CEUR Workshop Proceedings of KI-2004 Workshop on Applications of Description Logics (ADL 04), Ulm, Germany, Sep 24 2004.
- 6 Christopher, J. O. B., Xiao, S., Greg, B., Volker, H.: Ontoligent Interactive Query Tool. Proceedings of the Canadian Semantic Web Working Symposium, June 6, 2006, Quebec City, Quebec, Canada, Series: Semantic Web and Beyond: Computing for Human Experience, Vol. 2, Springer Verlag, 2006, pp. 155-169.
- 7 Catarci, T., Guiseppe, S., Michele, A.: Fundamental Graphical Primitives for Visual Query Languages. Information Systems, Vol. 18 n.2, p.75-98, March 1993.
- 8 Ni, W., Ling, T. W.: Glass: A Graphical Query Language for Semi-Structured Data. In Proc. Of Eighth International Conference on Database Systems for Advanced Applications (DASFAA '03), March 2003.
- 9 Chamberlin, D., Florescu, D., Robie, J., Simeon, J., Stefanescu, M.: Xquery: A Query Language for XML. Working draft, World Wide Web Consortium, February 2001. See <http://www.w3.org/TR/xquery/>.
- 10 Baker, P.G., Brass, A., Bechhofer, S., Goble, C., Paton, N., Stevens, R.: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. 6<sup>th</sup> Int. Conf. on Intelligent Systems for Molecular Biology.
- 11 Suh, B., Bederson, B.: OZONE: A Zoomable Interface for Navigating Ontology. HCIL Tech Report #2001-04, University of Maryland, College Park, MD 20742.
- 12 Papantonakis, A., King, P.J.H.: Gql, a Declarative Graphical Query Language Based on the Functional Data Model. Proc. Of the Workshop on Advanced Visual Interfaces, Bari, Italy, 1994, pp.113-122.
- 13 Consens, M.P., Mendelzon, A.O.: GraphLog: A Visual Formalism for Real Life Recursion. In Proceedings of 9<sup>th</sup> ACM SIGA CT-SIGMOID Symposium on Principles of Database Systems, Pages 404-416, 1990.
- 14 Catarci, T., Dongilli, P., Mascio, T.D., Franconi, E., Santucci, G., Tessaris, S. : An Ontology Based Visual Tool for Query Formulation Support. In ECAI, pages 308-312, 2004.