OWL-QL – A Language for Deductive Query Answering on the Semantic Web

Richard Fikes[1], Patrick Hayes[2], and Ian Horrocks[3]

[1]Knowledge Systems Laboratory, Computer Science Department
Stanford University
Stanford, California U.S.A.
fikes@ksl.stanford.edu

[2]Institute for Human and Computer Cognition
University of West Florida
Pensacola, Florida U.S.A.
phayes@coginst.uwf.edu

[3]Information Management Group
Department of Computer Science
University of Manchester
Manchester, M13 9PL, U.K.
horrocks@cs.man.ac.uk

Abstract

This paper discusses the issues involved in designing a query language for the Semantic Web and presents the OWL Query Language (OWL-QL) as a candidate standard language and protocol for query-answering dialogues among Semantic Web computational agents using knowledge represented in the W3C's Ontology Web Language (OWL). OWL-QL is a formal language and precisely specifies the semantic relationships among a query, a query answer, and the knowledge base(s) used to produce the answer. Unlike standard database and Web query languages, OWL-QL supports query-answering dialogues in which the answering agent may use automated reasoning methods to derive answers to queries, as well as dialogues in which the knowledge to be used in answering a query may be in multiple knowledge bases on the Semantic Web, and/or where those knowledge bases are not specified by the querying agent. In this setting, the set of answers to a query may be of unpredictable size and may require an unpredictable amount of time to compute.

I. Introduction

OWL Query Language (OWL-QL) is a formal language and protocol for a querying agent and an answering agent to use in conducting a ***query-answering dialogue*** using knowledge represented in the Ontology Web Language (OWL) [MH03]. OWL-QL is an updated version of the DAML Query Language (DQL) developed by the Joint United States/European Union ad hoc Agent Markup Language Committee[1], and the authors of this paper, who are members of that committee, are the editors of both the DQL specification [FHH03a] and the OWL-QL specification [FHH03b].

OWL-QL is intended to be a candidate standard language and protocol for query-answering dialogues among Semantic Web computational agents during which answering agents (which we refer to as *servers*) may derive answers to questions posed by querying agents (which we refer to as *clients*). As such, it is designed to be suitable for a broad range of query-answering services and applications. Also, although OWL-QL is specified for use with OWL, it is designed to be prototypical and easily adaptable to other declarative formal logic representation languages, including, in particular, first-order logic languages such as KIF [G98] and the earlier W3C languages, RDF [B03], RDF-S [BG03], and DAML+OIL [HHP01].

The OWL-QL Web site (http://ksl.stanford.edu/projects/owl-ql/) provides links to the OWL-QL specification and to current OWL-QL implementations, including an OWL-QL client with a Web browser user interface suitable for use by humans for asking queries of an OWL-QL server. This paper describes OWL-QL and

[1] The committee is chaired by Michael Dean, and the members during the development of DQL were Harold Boley, Daniel Brickley, Stefan Decker, Richard Fikes, Benjamin Grosof, Frank van Harmelen, Patrick Hayes, Jeffrey Heflin, Ian Horrocks, Ora Lassila, Deborah McGuinness, Peter Patel-Schneider, and Lynn Andrea Stein.

discusses significant design issues that arise in the development of a language for deductive query answering on the Semantic Web.

II. Querying on the Semantic Web

The design of OWL-QL is predicated on a number of basic assumptions about query-answering dialogs on the Semantic Web, and on the intended role of OWL-QL.

First, the Semantic Web is expected to include many kinds of query-answering services with access to many types of information represented in many formats. Traditional database query languages like SQL [I92] and languages for retrieving information from the Web (e.g., XML query [M03] and RQL [KC03]) are not suitable for supporting such heterogeneity, ranging from simple services that provide retrieval-based functionality to complex services that provide sophisticated automated reasoning functionality and may act as intermediary agents between their clients and more specialized servers. OWL-QL supports query-answering dialogues in which the answering agent may use automated reasoning methods to derive answers to queries, as well as scenarios in which the knowledge to be used in answering a query may be in multiple knowledge bases on the Semantic Web, and/or where those knowledge bases are not specified by the client.

Second, we must expect that some servers will have only partial information about the topic, some will have performance limitations, and some will be simply unable to handle certain kinds of queries. So, it is important that the querying protocol provide some means for the transfer of partial query results and about the querying process itself. In this setting, the set of answers to a query may be of unpredictable size and may require an unpredictable amount of time to compute. OWL-QL therefore provides an adaptable query answering protocol which both allows a server to return partial sets of answers as the answers are computed and allows a client to specify the maximum number of answers that it wants the server to include in the next set of answers it sends to the client.

Third, a Semantic Web query language needs to support queries that do not include a specification of the knowledge base(s) to be used in answering the query. That is, just as the user of a current Web browser does not specify which Web sites to consider when given a search request, we anticipate that a common use of the Semantic Web will be to send a query to a server and expect the server to select reliable knowledge sources from which to produce answers. OWL-QL supports server selection of the knowledge base(s) to be used in answering a query, and client requests that a server identify the knowledge base(s) used in answering a query.

Fourth, the set of notations and surface syntactic forms used on the Web is already large, and various communities have different preferences, none of then universal. Even within the nearest to a single established syntax, XML, there are many alternative 'styles' of notational design in use. The essential aspects of the design of OWL-QL are independent of the surface syntax of the language. So, we have stated the OWL-QL specification at an 'abstract' or structural level, allowing essentially the same language to be implemented in multiple surface syntactic forms. The specification describes the types of objects (e.g., queries and answers) that are passed between server and client during a query-answering dialogue, the necessary and optional components of each of those object types, and the expected response of a server to each type of object sent to it by a client. In addition, we have included with the abstract specification of OWL-QL a syntax specification for the language in XML Schema in order to provide an example syntax for the language. We claim that this style of 'meta-specification' of OWL-QL will be of more utility in a Semantic Web context than the more traditional approach. For the examples in this paper, we use an informal human readable surface syntax for queries and answers.

Finally, a basic premise of the Semantic Web is that the declarative languages used to represent knowledge on the Web will have a formally defined semantics and theory of logical entailment. That is the case for OWL, and for most of its predecessors, including DAML+OIL, RDF, and RDF-S. That premise also applies to query languages for the Semantic Web in that the specification of a Semantic Web query language needs to include a formal description of the semantic relationships among a query, a query answer, and the knowledge base(s) used to produce the answer. The OWL-QL specification provides those formal descriptions.

III. Queries and Answers

A. Query Patterns and Variables

An OWL knowledge base K is considered to be a collection of logical sentences K_S that represents a logical theory in which a collection of entailed sentences K_{ES} are true such that $K_S \Rightarrow K_{ES}$. It is natural, therefore, to think of a query as asking for sentences in K_{ES} that "satisfy" a given "sentence schema", and to think of using bindings to variables in that sentence schema as specifying answers to the query. This conventional picture, which we have adopted for OWL-QL, is compatible with the semantics of Semantic Web representation languages and is consistent with the Codd database model [C70] and many other logical formalisms.

An OWL-QL query-answering dialogue is initiated by a client sending a query to an OWL-QL server. An OWL-QL query is an object necessarily containing a ***query pattern*** that specifies a collection of OWL sentences in which some URIrefs are considered to be variables. For example, a client could ask "Who owns a red car?" with a query having the query pattern shown in Figure 1.

[2]Query: ("Who owns a red car?")
  Query Pattern: {(owns ?p ?c) (type ?c Car) (has-color ?c Red)}
  Must-Bind Variables List: (?p)
  May-Bind Variables List: ()
  Don't-Bind Variables List: ()
  Answer Pattern: {(owns ?p "a red car")}
  Answer KB Pattern: …

Answer: ("Joe owns a red car?")
  Answer Pattern Instance: {(owns Joe "a red car")}
  Query: …
  Server: …

Figure 1. A simple OWL-QL query and answer

A query may have zero or more answers, each of which provides bindings of URIrefs or literals to some of the variables in the query pattern such that the conjunction[3] of the answer sentences - produced by applying the bindings to the query pattern and considering the remaining variables in the query pattern to be existentially quantified - is entailed by a knowledge base (KB) called the *answer KB*. For example, the answer "Joe owns a red car." shown in Figure 1 means the answer KB entails the following sentence, expressed here in first-order logic (using KIF syntax):

(exists (?c) (and (owns Joe ?c) (type ?c Car) (has-color ?c Red)))

A formal description of the relationships between a query and its answers is given in the appendix of this paper.

Each binding in a query answer is a URIref or a literal that either explicitly occurs as a term in the answer KB or is a term in OWL. That is, OWL-QL is designed for answering queries of the form "What URIrefs and literals from the answer KB and OWL denote objects that make the query pattern true?" or, when there are no variables to be bound in the query pattern, "Is the query pattern true in the answer KB?". We will say that a variable that has a binding in a query answer is *identified* in that query answer. The use of entailment here is what most clearly distinguishes OWL-QL from SQL and other retrieval languages, since although a database may be understood to entail its table entries considered as atomic assertions, entailment in OWL also allows more complex relationships to hold which may be much more expensive to compute.

We now describe how a client specifies which syntactic elements of a query pattern are to be considered as variables and what bindings are expected and required in a query answer. OWL has no suitable notion of a variable, so an OWL-QL query pattern is simply an OWL knowledge base, and a query specifies which URI references in its query pattern are to be considered to be variables. Data base query languages typically designate

---

[2] We show a query pattern as a set of triples of the form (<property> <subject> <object>), where any item in the triple can be a variable. We show variables as names beginning with the character "?".

[3] We use "conjunction" informally in this introductory section since OWL does not have a logical connective for conjoining sentences or for conjoining knowledge bases. We consider a conjunction of sentences to be a sentence that is true if and only if all of its conjuncts are true. We consider a conjunction of knowledge bases to be a knowledge base consisting of all the sentences in all the conjunct knowledge bases.

a subset of the variables in a query as being the variables for which bindings are to be included in a query answer. In typical knowledge representation languages (including OWL), a knowledge base may entail the existence of a query answer but not entail a binding for every variable in the query. For example, a knowledge base that says every person has exactly one father (i.e., that every object of type "Person" has exactly one value of the property "hasFather") and that Joe is a person (i.e., that "Joe" is type "Person"), entails that Joe has a father but may not entail a value of property "hasFather" for Joe. (I.e., the knowledge base may not identify the father.)

OWL-QL supports existentially quantified answers by enabling the client to designate some of the query variables for which answers will be accepted with or without bindings. That is, each variable that occurs in a OWL-QL query is considered to be a *must-bind variable*, a *may-bind variable*, or a *don't-bind variable*. Answers are required to provide bindings for all the must-bind variables, may provide bindings for any of the may-bind variables, and are not to provide bindings for any of the don't-bind variables. These designations are made by inclusion of a ***must-bind variables list***, a ***may-bind variables list***, and a ***don't-bind variable list*** in an OWL-QL query. These lists contain URI references that occur in the query, and no URI reference can be an item of more than one of these lists.

The following example illustrates the effects of having must-bind, may-bind, and don't-bind variables. Consider an answer KB containing sentences saying that every person has exactly one father, each of a large number of C_is is a person, and F_k is a father of C_k for each C_k in a small subset of the C_is. Then consider a query with the query pattern "{(hasFather ?p ?f)}", meaning "?p has father ?f", and the following cases:

- If ?f is a don't-bind variable, then the complete set of query answers contains C_i answers (i.e., one for each known person), and each query answer identifies a person but does not identify the person's father.
- If ?f is a must-bind variable, then the complete set of query answers contains only C_k answers (i.e., one for each known father), and each query answer identifies both a person and the person's father.
- If ?f is a may-bind variable, then the complete set of non-redundant query answers contains C_i answers (i.e., one for each known person), and each query answer identifies a person and identifies the person's father in the cases where the father is known.

Specifying a query pattern and the variables lists does not indicate how the answers – the bindings to the pattern variables – are to be returned from the server to the client. OWL-QL allows a client to specify the format in which answer bindings are returned by (optionally) including an ***answer pattern*** in a query that can be any list expression containing all of the query's must-bind and may-bind variables. If no answer pattern is specified, a two item list whose first item is the query's must-bind variables list and whose second item is the query's may-bind variables list is used as the answer pattern. Each query answer contains an instantiation of the answer pattern in which each variable having a binding in the answer is replaced by its binding.

B. Including Assumptions in a Query

Since OWL does not have an "implies" logical connective, "if-then" queries such as "If Joe is a person, then does Joe have a father?" cannot be stated using only a query pattern. OWL-QL facilitates the representation of "if-then" queries by enabling a query to optionally include a ***query premise*** that is an OWL KB or a KB reference. When a premise is included in a query, it is considered to be included in the answer KB. Omitting the query premise is equivalent to providing an empty query premise. Figure 2 provides an example of a query that includes a premise.

```
Query: "If C1 is a Seafood Course and W1 is a drink of C1, then what color is W1?"
  Premise: {(type C1 Seafood-Course) (has-drink W1 C1)}
  Query Pattern: {(has-color W1 ?x)}
  Must-Bind Variables List: (?x)
  …
```

Figure 2. Example if-then query.

C. Specifying Answer KBs

The set of OWL sentences that are used by the server in answering a query is referred to as the *answer KB*. This may be one or more actual knowledge bases, or a virtual entity representing the total information available to the server at the time of answering. An OWL-QL query contains an ***answer KB pattern*** that is a KB, a list of KB

references, or a variable. If a query's answer KB pattern is a KB or a reference to a KB, then the conjunction of the answer sentences specified by each query answer must be entailed by that KB. If a query's answer KB pattern is a list of KBs and/or KB references, then the conjunction of the answer sentences specified by each query answer must be entailed by the conjunction of the KBs in or referenced in that list. If a query's answer KB pattern is a variable, then the server is free to select or to generate an answer KB from which to answer the query, but if the variable is a must-bind variable, then the answer must provide a binding to the variable that is a reference to a resource representing the answer KB. In many cases, that URIref will be a URL that can be used to access the KB or to communicate with the server about the KB, but the URIref is not required to be a URL.

IV. Query Answering Dialogues

A query may have any number of answers, including none. In general, we cannot expect that a server will produce all the answers at once, or that the client is willing to wait for an exhaustive search to be completed by the server. We also cannot expect that all servers will guarantee to provide all answers to a query, or to not provide any redundant answers. OWL-QL attempts to provide a basic tool kit to enable clients and servers to interact under these conditions.

Answers are delivered by the server in *bundles*, and the client can specify the maximum number of answers in each bundle. Each request from a client to a server for answers to a query can include an ***answer bundle size bound***, and the server is required to respond by delivering an ***answer bundle*** containing at most the number of query answers given by the answer bundle size bound. The collection of all answers sent to the client by the server in a query-answering dialogue is called the *response collection* of that dialogue.

An answer bundle must also contain either a ***process handle*** or one or more character strings called ***termination token****s*. The presence of a termination token in an answer bundle indicates that the server will not deliver any more answers to the query, and the presence of a server continuation in an answer bundle represents a commitment by the server to deliver another answer bundle if more answers to the query are requested by a client.

A client requests additional answers to a query by sending the server a ***server continuation*** containing the process handle provided by the server in the previously produced answer bundle and an answer bundle size bound for the next answer bundle to be produced by the server. Upon receiving a server continuation from a client, the server is expected to respond similarly by sending to that client another answer bundle. A client terminates a query-answering dialogue by sending the server a ***server termination*** containing the process handle provided by the server in the previously produced answer bundle. The overall structure of the dialogue is illustrated in Figure 3.

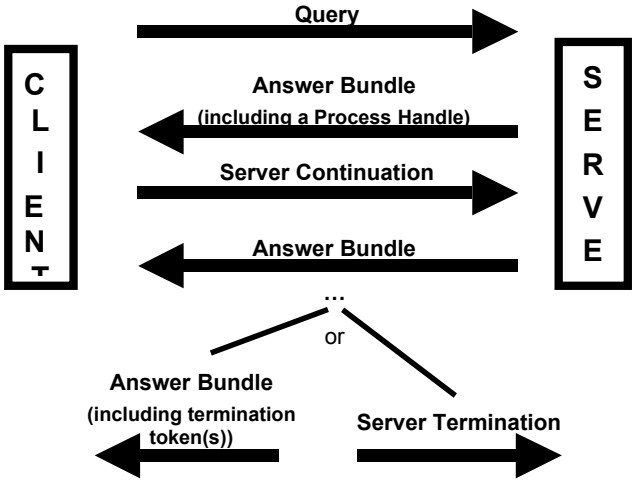

Figure 3. OWL-QL Query-Answering

Note that more than one client can participate in a given query-answering dialogue with a server in that the client that sends a server continuation to the server need not be the same client that sent the original query or earlier server continuations during the dialogue.

The OWL-QL specification does not restrict the nature or content of process handles. Different servers may use process handles in different ways. Some database servers may generate a complete table of answers, store it in association with a record of the query, and then use as a process handle an index or hash code keyed to the query record. Other servers may take advantage of the protocol to store enough information in a process handle to enable them to reconstruct the state of a search process and continue the search. Still others may simply store the answers already produced in a record of the query, use the query record as a process handle, and restart the query answering process from the beginning each time additional answers are requested. Note that the inclusion of a process handle in an answer bundle is not a commitment to provide more answers. If, for example, a server is unable to reconstruct the state of a query process when asked for more answers, it can always respond with an answer bundle containing a termination token and no answers.

OWL-QL specifies the following three termination tokens:

- "**End**" simply indicates that the server is unable to deliver any more answers; it is conventionally used to terminate the process of responding to a query. One possible response to any query is a single answer bundle containing "End", indicating that the server will not provide any answers to the query.
- "**None**" expresses an assertion by the server that no other answers are possible. This assertion should be used with care, particularly when in response to a query containing a don't-bind variable, where it has the semantic force of a negated existential, i.e. a universal negation.
- "**Rejected**" can be used by a server to indicate that the query is outside its scope for some reason, e.g., by being posed in a subset of the language which it is unable to process, or by being in some way ill-formed. This is a crude device for expressing what could be a complex topic, but servers may also define their own termination tokens to be used in conjunction with the OWL-QL tokens, which can be used to express more nuanced forms of rejection.

The use of "None" would be appropriate in a case where a server has access to a collection of data which is known to be complete or exhaustive in some way, such as a database of employees of a company. Suppose a query asks for all employees with a salary over $200K, and the returned answer bundle is empty, terminated with "None". This would be sufficient grounds for the client to conclude that the company has no employees with that salary. Notice that the termination token "End" would *not* provide this kind of a guarantee, given the monotonic semantics of OWL. To treat an "End" token as though it meant "None" would be to make a 'closed-world assumption', which is not valid. The distinction between these tokens was motivated in part by the widely noted utility of closed-world reasoning. Making the distinction explicit in the exchange protocol provides a way to express closure without forcing clients to draw invalid conclusions in cases where a closed-world assumption is inappropriate.

These conventions, taken together, allow a simple expression of a 'yes/no' query. Such queries can be expressed by a query pattern with no variables; an answer bundle containing one answer indicates that the pattern is entailed by the KB; an answer bundle containing no answers and the termination token "None" indicates that the query is known to not be entailed by the answer KB; and any other answer bundle containing no answers indicates that entailment of the query cannot be determined by the server.

OWL-QL does not specify a complete inter-agent protocol (e.g., with provisions for time-outs, error handling, resource budgets, etc.). OWL-QL servers are required to support the specified core protocol elements and are not constrained by the OWL-QL specification as to how additional protocol functionality is provided. Queries, answer bundles, server continuations, and server terminations are all designed to support additional protocol functionality in that they are objects consisting of property value pairs and can include values of additional properties as specified and supported by a given server.

A. Duplicate and Redundant Answers

While there are no global requirements on the response collection of a query answering dialogue other than that all its members are correct answers, clients will typically find it useful to know whether a given server ensures that its response collections contain no duplicate or redundant answers. Redundant answers can be a particularly vexing problem for queries that contain a variable that is a value of a maxCardinality restriction in a query pattern since a server could potentially produce an unlimited number of answers with less and less specific bindings for

such a variable. For example, if a property for an individual has a maxCardinality restriction value of 3, then it also has a maxCardinality restriction value of 4 and 5 and 6, etc.

For some servers, assuring that no duplicate or redundant answers are produced would be very expensive, and imposing such a requirement as part of an intended standard would impose a high initial implementation cost for simple servers. On the other hand, a server that is able to deliver non-repeating or non-redundant responses may wish to advertise this useful quality. OWL-QL specifies a set of conformance levels which a server can use to do that advertising.

A server which always produces a response collection that contains no duplicate answers can be called ***non-repeating***, where two answers are considered to be duplicates if they have the same set of bindings. A server which always produces a response collection that contains no redundant answers can be called ***terse***, where an answer is considered to be redundant if it *subsumes* (i.e., duplicates or is less specific than) some other answer in the response set. An answer is considered to be less specific if it binds fewer may-bind variables or has less-specific bindings for variables that occur only as values of minCardinality or maxCardinality restrictions. Formally:

An answer A1 subsumes an answer A2 if and only if
for every variable V that has a binding in A1's binding set,
V has a binding in A2's binding set and
the binding of V in A1's binding set subsumes the binding of V in A2's binding set.

For every V that occurs in a query Q,
binding B1 of a variable V subsumes a binding B2 of V if and only if
B1 is identical to B2 or
V occurs in Q only as a value in a minCardinality restriction in the query pattern of Q and
B1 is less than B2 or
V occurs in Q only as a value in a maxCardinality restriction in the query pattern of Q and
B1 is greater than B2.

Guaranteeing terseness is a quite harsh requirement on a server that is incrementally deriving answers and returning bundles of answers as they are produced. The difficulty is that if such a server derives and returns an answer A_1 with an unbound may-bind variable (i.e., A_1 does not provide a binding for that variable), then it cannot later return any answer A_2 that it derives containing the same bindings as those in A_1 with the addition of a binding for the unbound may-bind variable because A_1 would subsume any such A_2. Similarly, if such a server derives and returns an answer A_1 with a binding B for a variable V that occurs in the query only as a value in a minCardinality (maxCardinality) restriction in the query pattern, then it cannot later return any answer A_2 that it derives containing the same bindings as those in A_1 with the addition of a binding for V that is less than (greater than) B because A_1 would subsume any such A_2.

A much more reasonable requirement is for a server to guarantee that it will not return any answer that subsumes any previous answer it has produced in a given query answering dialogue; that is, it will not gratuitously return answers to a client that are duplicates of or are less specific than answers it has already returned. Such a server can advertise itself as being ***serially terse***. Note that a terse server is necessarily a serially terse server and that a serially terse server is necessarily a non-repeating server. We expect that most applications will require the OWL-QL servers they use to be serially terse.

Note that although additional criteria for answers being redundant would be useful for clients, care must be taken to consider the computational burden on a server satisfying such criteria would impose. For example, consider a variable V that occurs only as the value of an allValuesFrom restriction in a query pattern. If V has a binding to class C in a query answer, then answers which differ only in that they have a binding of V to a superclass of C would also be correct. However, those answers would be redundant and very unlikely to be useful to a client. If the definition of redundant answers were to be extended to include such variables values, a serially terse server could not return an answer containing a binding for such a variable until it determined that the subclassOf relationship is false (not just that it is unknown) between that binding and all the other bindings that it has produced for that variable in answers that differ only in their binding of that variable.

V. Discussion

A. Utilizing the Expressive Power of OWL

Query languages for description logics and other logic-based knowledge representation formalisms often include explicit "structural queries", such as queries asking about the subsumers, subclasses, and instances of classes [BM96] [BBH91] [BHP99]. In OWL-QL, these kinds of questions can be formulated using the standard query mechanism, taking advantage of the expressive power of the OWL language itself. For example, answers to the query using the query pattern {(subclassOf ?x Person)}, where ?x is a must-bind variable, will be the derivable subclasses of class `Person`. Similarly, answers to the query using the query pattern {(type ?x Person)}, where ?x is again a must-bind variable, will be the derivable instances of class `Person`. This ability is limited to concepts which can be expressed using OWL: for example, there is no way in OWL to express the concept of a most general subclass or a most specific type. The OWL-QL query pattern language was not extended beyond the expressive capabilities of the content language used in the knowledge bases being queried (i.e., OWL) so as not to impose greater computational burdens on a server than are defined by the specification of the language it uses.

Some SQL-style queries can be expressed using a similar technique. For example, a simple relational table might be encoded in OWL as a collection of assertions using rdf:value with the following format:

(rdf:value ex:Joe _:x)
(rdf:type _:x ex:employeeInfo)
(ex:surname _:x "Jones")
(ex:SSnumber _:x "234-55-6789")
(ex:age _:x xsd:number^^"43")
(ex:location _:x ex:marketing)

where the value of rdf:type is the 'table entry' and the table name is its type. The SQL command '*select SSnumber from employeeInfo*' then translates into the OWL-QL query pattern

```
(rdf:type ?x ex:emplyeeInfo)
(ex:SSnumber ?x ?y)
```

with ?y being a must-bind variable and ?x being a don't bind variable. More complex SQL conditions can be expressed by more complex OWL query patterns – for example, a conditional selection can be expressed as an OWL restriction on a class such as ex:employeeInfo - but these will provide answers only if the server is able to perform the appropriate reasoning. In general, in contrast to the presumptions of the SQL querying model, the OWL-QL assumption is that nontrivial inferences about the data are performed by the server rather than by the client.

B. Iterative Optimization

Clients can use OWL-QL to obtain answers to queries involving concepts not expressible in OWL such as "most general subclass" or "most specific type", and indeed to optimize any variable with respect to any given transitive property, by using an iterative optimization technique as follows. To optimize the value of a must-bind variable V in a query Q with respect to a transitive property P and a server S, send Q to S asking for at most one answer. If S provides an answer to Q with a binding of B_i for V, then send S a query Q' consisting of Q with the additional premise "(P B_i V)" and ask for at most one answer. If S does not provide an answer to Q', then B_i is the optimal binding that S can provide for V. If S provides an answer to Q' with a binding of B_j for V, then send S a new query Q' consisting of Q with the additional premise "(P B_j V)". Continue this iterative querying until S does not provide an answer. The last binding produced for V is the optimal binding that S can provide for V. For example, a client could use iterative optimization to find the most general subclass of C by asking for at most one answer to a query with query pattern {(subclassOf ?x C)} and must-bind variable ?x, and then successively asking for at most one answer to the same query with the addition of premise {(subclassOf C_i ?x)}, where C_i is the most recently returned binding for ?x.

C. Asking About the Number of Answers

Many problems involve asking "how many" queries, such as "How many cars does Joe own?". One might be tempted to ask a "how many" query by asking an OWL-QL query and counting the number of answers produced by the server. The problems with that strategy are two fold: Firstly, the server may complete the query answering dialogue without guaranteeing that it has found all the answers; and secondly, the bindings for a given variable in multiple answers may all denote the same entity (i.e., they may be equal). So, for example, a server may respond to a query having query pattern {(type ?x car) (owns Joe ?x)} with three answers that bind ?x respectively to "Car1", "Car2", and "Car3". If the server terminates the dialogue with the termination token "End" (rather than "None"), then the client doesn't know whether the answer KB entails more bindings that denote cars owned by Joe, and the client doesn't know whether Car1=Car2, Car1=Car3, and/or Car2=Car3. So, all that the client can conclude from the server's response about how many cars Joe owns is that Joe owns at least one car. In order for the client to determine how many cars are owned by Joe, it would have to ask the query of a server that advertises itself as "complete", and it would have to make (typically multiple) subsequent queries to determine which bindings denote different cars.

The primary means that OWL provides for expressing the number of entities in a domain of discourse that satisfy some set of conditions (e.g., how many cars are owned by Joe) are cardinality restrictions on the number of values of a given property for a given individual or class of individuals (e.g., "What is the value of a cardinality restriction on property "ownsCar" for "Joe"?", where "ownsCar" is a subproperty of "owns" that has a "Car" allValuesFrom restriction of "Car" for Joe). Thus, in general, the way to meaningfully ask "how many" queries using OWL-QL is to ask for the value of an appropriate cardinality restriction, rather than asking a query and counting the answers.

OWL-QL does, in fact, allow a client to ask for how many answers a server will provide for a given query by including in the query an ***answer number request*** and an accompanying query variable. If the variable is a must-bind variable, then providing the number of answers is required, if the variable is a may-bind variable, then providing the number of answers is optional. The primary motivation for including this feature in OWL-QL is that many database servers record information about the number of entries in their data tables and can rapidly respond to requests for this information. Thus, such servers can often inform a client as to how many answers they will provide to a query with no significant additional effort.

VI. Closing Comments

In this paper, we have discussed what were to us surprisingly difficult issues involved in designing a query language for the Semantic Web, and have presented a candidate standard language and protocol for query-answering dialogues among Semantic Web computational agents using knowledge represented in the W3C's Ontology Web Language (OWL). OWL-QL is a formally specified language with precisely defined semantic relationships among a query, a query answer, and the knowledge base(s) used to produce the answer. Unlike standard database and Web query languages, OWL-QL supports query-answering dialogues in which the answering agent may use automated reasoning methods to derive answers to queries, as well as dialogues in which the knowledge to be used in answering a query may be in multiple knowledge bases on the Semantic Web, and/or where those knowledge bases are not specified by the querying agent. In this setting, the set of answers to a query may be of unpredictable size and may require an unpredictable amount of time to compute.

Although OWL-QL is specified for use with OWL, it is designed to be prototypical and easily adaptable to other declarative formal logic representation languages, including, in particular, first-order logic languages such as KIF and the earlier W3C languages RDF, RDFS, and DAML+OIL. OWL-QL is, in fact, being used in multiple projects in which knowledge and query patterns can be represented in KIF as well as OWL.

Acknowledgements

This research was supported by the Defense Advanced Research Projects Agency under contracts F30602-00-0579-P00001 (Fikes, Horrocks) and F30602-00-2-0577 (Hayes). We would also like to acknowledge the valuable contributions of Yulin Li and the other members of the Joint United States / European Union ad hoc Agent Markup Language Committee, especially Peter Patel-Schneider.

References

[B03] Dan Brickley (RDF Interest Group Chair and RDF Core Working Group co-chair); Resource Description Framework (RDF); World Wide Web Consortium; August 5, 2003; http://www.w3.org/RDF/.

[BBH91] Franz Baader, Hans-Jürgen Bürkert, Jochen Heinsohn, Bernhard Hollunder, Jürgen, Müller, Bernard Nebel, Werner Nutt and Hans-Jürgen Profitlich; "Terminological Knowledge Representation: A Proposal for a Terminological Logic"; Technical Memo; DFKI; 1991.

[BG03] D. Brickley and R.V. Guha; "RDF Vocabulary Description Language 1.0: RDF Schema"; World Wide Web Consortium; October 10, 2003; http://www.w3.org/TR/rdf-schema/.

[BHP99] Sean Bechhofer, Ian Horrocks, Peter F. Patel-Schneider and Sergio Tessaris; "A Proposal for a Description Logic Interface"; Proceedings of Description Logic Conference (DL'99); pp. 33-36; August 11-14, 1999.

[BM96] Alex Borgida and Deborah McGuinness; "Asking Queries about Frames"; Proceedings of Principles of Knowledge Representation and Reasoning (KR '96); Cambridge, Massachusetts; Morgan Kaufmann; November 5-8, 1996. Also appears in Proceedings of International Workshop on Description Logics, Cambridge, Mass.; 1996. http://www.ksl.stanford.edu/people/dlm/papers/kr96-abstract.html.

[C70] Edgar Codd; "A Relational Model of Data for Large Shared Data Banks "; Communications of the ACM; Vol. 13, No. 6; June 1970; pp. 377-387; http://www.acm.org/classics/nov95/toc.html.

[FHH03a] Richard Fikes, Pat Hayes, and Ian Horrocks (editors); "DAML Query Language (DQL) Abstract Specification"; Joint United States/European Union ad hoc Agent Markup Language Committee; DARPA Agent Markup Language (DAML) Program; April 1, 2003; http://www.daml.org/2003/04/dql/.

[FHH03b] Richard Fikes, Pat Hayes, and Ian Horrocks; "OWL Query Language (OWL-QL) Abstract Specification"; DARPA Agent Markup Language (DAML) Program; October 13, 2003.

[G98] Michael Genesereth; "Knowledge Interchanged Format (KIF)"; 1998; http://logic.stanford.edu/kif/kif.html.

[HHP01] Ian Horrocks, Frank van Harmelen, and Peter Patel-Schneider; DAML+OIL; DARPA Agent Markup Language (DAML) Program; March 2001; http://www.daml.org/2001/03/daml+oil-index.html.

[I92] "Information Technology --- Database Languages --- SQL"; ISO/IEC 9075:1992; American National Standards Institute; New York, N.Y; 1992.

[KC03] G. Karvounarakis and Vassilis Christophides; "The RDF Query Language (RQL)"; Institute of Computer Science, Foundation of Research Technology; Hellas, Greece; July 18, 2003; http://139.91.183.30:9090/RDF/RQL/.

[M03] Massimo Marchiori (W3C contact for XML Query); "XML Query (XQuery)"; World Wide Web Consortium; September 23, 2003; http://www.w3.org/XML/Query.

[MH03] Deborah McGuinness and Frank van Harmelen (editors); "OWL Web Ontology Language Overview"; August 18, 2003; http://www.w3.org/TR/owl-features/.

Appendix – Formal Relationship between a Query and a Query Answer

An OWL-QL query necessarily includes a query pattern that is an OWL knowledge base, a list of must-bind variables, a list of may-bind variables, and a list of don't-bind variables. Each variable in these lists is a URIref. Also, a query optionally includes an answer pattern that is an arbitrary list structure. If there is no answer pattern included in a query, a two item list whose first item is the query's must-bind variables list and whose second item is the query's may-bind variables list is used as the answer pattern.

A query answer necessarily includes an answer pattern instance that is the query's answer pattern with each of the query's must-bind variables and zero or more of the may-bind variables (and none of any other variables that

occur in the answer pattern) replaced by a URIref or literal. The answer pattern instance specifies a binding set that satisfies the following conditions:

- Each element of the binding set is a lexical mapping that associates a URIref or literal to a query variable.
- The binding set contains a binding to each of the must-bind query variables, to zero or more of the may-bind query variables, and to none of the don't-bind query variables;
- If the binding set contains a binding to a variable that is the answer KB pattern, then the binding is to a reference to the answer KB;
- If the binding set contains a binding to a variable that accompanies an answer number request, then the binding is to a literal that is a non-negative integer;
- All bindings in the binding set to variables in the query pattern are URIrefs or literals that occur in the OWL language or in the answer KB;

In addition, a binding set in a query answer specifies a set of OWL sentences that the server claims are entailed by the answer KB. That claim is specified formally as follows. Suppose:

- Q is the query pattern for the query of which this is an answer;
- B is the subset of the binding set consisting of all the bindings to variables that occur in Q;
- B(Q) is the KB obtained by applying the bindings B to Q, i.e., by substituting the URIref or literal that is associated with v for every variable v that has a binding in B; and
- The "remaining variables" are those variables in B(Q) that are not replaced by B.

Then,

- **An interpretation I satisfies B(Q)** if there is a mapping C from the remaining variables of B(Q) to the universe of I such that I+C satisfies B(Q); that is, if the interpretation can be extended to provide interpretations of the remaining variables in some way that makes B(Q) true.
- **The answer KB entails B(Q)** just in case B(Q) is true in every interpretation that makes the answer KB true. Intuitively, this means that the remaining variables are treated as existential 'blanks' which indicate that something exists without saying what it is.

The additional condition, then, that the server is claiming is satisfied by a binding set is that **the answer KB entails B(Q)**.