QueryMed: An Intuitive SPARQL Query Builder for Biomedical RDF Data

Oshani Seneviratne
Massachusetts Institute of Technology
Cambridge, MA
USA
oshani@csail.mit.edu

Rachel Sealfon
Massachusetts Institute of Technology
Cambridge, MA
USA
rsealfon@csail.mit.edu

ABSTRACT

We have developed an open-source SPARQL query builder and result set visualizer for biomedical data, QueryMed, that allows end users to easily construct and run translational medicine queries across multiple data sources.

QueryMed is flexible enough to allow queries relevant to a wide range of biomedical topics, runs queries across multiple SPARQL endpoints, and is designed to be accessible to users who do not know the structure of the underlying ontologies or the SPARQL query language. The system allows users to select the data sources that they wish to use, drawing on their specialized domain knowledge to decide the most appropriate data sources to query. Users can add additional data sources if they are interested in querying endpoints that are not in the default list. After retrieval of the initial result set, query results can be filtered to improve their relevance. As an advanced search feature, the system also allows the user to exploit the underlying structure of the RDF data to improve query results.

Categories and Subject Descriptors

J.3 [**Life and Medical Sciences**]: Computer Applications; H.3.3 [**Information Search and Retrieval**]: Information Systems

Keywords

Biomedical Ontologies, SPARQL, Query Federation, Query Building, Semantic Web, User Interfaces

1. INTRODUCTION

The biomedical domain is among the early successes of the semantic web, due to the rapidity with which the biomedical community has made its data available in RDF triple stores [17]. Although a plethora of useful biomedical data is currently available in RDF, there is a need for easy-to use systems that do not require the end user to have knowledge of the underlying structure of the data, and also allow users to run federated queries on multiple SPARQL endpoints.

For example, a physician may know her patient's personal information, symptoms, current medications, and genotype. She may wish to determine the patient's treatment plan and identify clinical trials for which the patient is eligible. Although the physician has a single question–"based on the

information I have about this patient, what is the best treatment plan and set of clinical trials available?"–there is no single data source that the physician can use to answer this question. The information that the physician needs must be gathered from numerous data sources such as Pubmed, DailyMed, Drugbank, LinkedCT, Diseasome, and GO [6, 1, 3, 5, 2, 4]. Her question must be broken up into discrete pieces that can be executed individually at one data source at a time.

Suppose the physician might be interested in information on coronary artery disease (which she thinks is the cause of her patient's symptoms). The information about the disease can be found in Diseasome, a database that lists diseases, disease genes, and associations between them [2]. The physician might also want to know the set of drugs in DailyMed [1] and Drugbank [3] that can be used to treat coronary artery disease. Information on clinical trials available for her patient can be found in LinkedCT [5]. Due to the large number of databases that the physician needs to query in order to find an answer to her single question, she is likely to find useful a system that can automatically run queries over multiple data sources. Also, the physician may not know SPARQL query syntax, the location of the SPARQL endpoints, or the structure of the relevant ontologies. She is likely to want an intuitive way to query and to display the query result. Developing intuitive ways to query multiple data sources and display results is both an important and a challenging problem. Our system, QueryMed, allows users with no knowledge of the SPARQL query language or the structure of the underlying ontologies to easily run queries across multiple SPARQL endpoints.

This paper is organized as follows: Section 2 provides background information on the semantic web and its relevance for the biomedical domain. Section 3 describes our system. Section 4 discusses related work and illustrates how QueryMed differs from previous systems. Finally, section 5 outlines future work and summarizes the contributions of our system.

2. BACKGROUND

The semantic web can be viewed as a global database system for the data available on the world wide web. Semantic web data is modeled by structured languages such as RDF and OWL, and can be queried using the SPARQL query language. The addition of structure to web data allows inferences to be automatically drawn by intelligent agents integrating data from multiple sources [9].

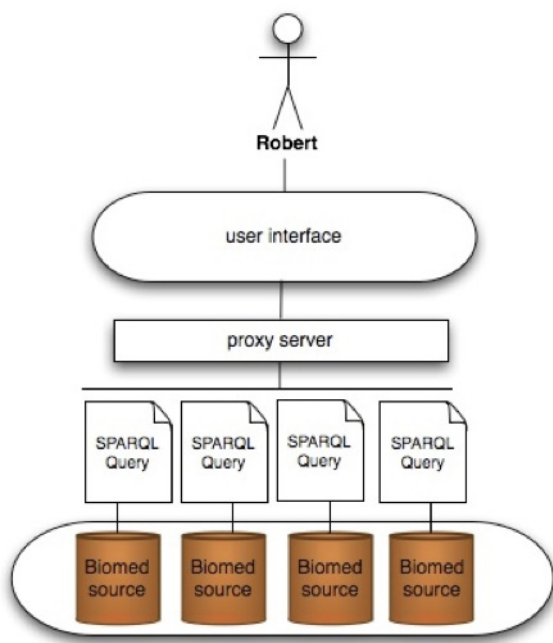Many major biological and biomedical data resources, in-

Figure 1: QueryMed Architecture Overview

Figure 2: QueryMed Architecture Details

cluding Gene Ontology, DailyMed, LinkedCT, and Diseasome, are currently available as RDF. Integrating biomedical data across multiple data sources and automatically extracting specific knowledge from web resources are crucial tasks for physicians and biologists. These semantic web resources represent valuable repositories of information that can be automatically mined for applications that require biological knowledge.

Although many valuable resources in the biomedical domain are available in RDF, there are a number of challenges that must be addressed in order to make such resources accessible to physicians, patients, and life scientists. One challenge is constructing systems that allow end users to run intuitive queries on biomedical data. Users of biomedical resources are likely to have extensive domain knowledge, but be unfamiliar with the SPARQL query language syntax and with the structure of biomedical ontologies. Almost all the SPARQL endpoints available today are offering a generic query interface that requires users to write SPARQL queries. Also, no hints about the structure of data available from the SPARQL endpoint is given as the user is constructing the SPARQL query. The user has to essentially construct the queries that are needed for her purpose by trial and error. This can especially be a daunting task for someone who is new to semantic web technologies. Therefore, it is important to design user-friendly systems that allow users to take advantage of the wealth of structured biological knowledge available on the semantic web. Another central challenge is designing systems that permit users to query multiple data sources simultaneously, since relevant biological data is often distributed among many sources [14].

3. QUERYMED

QueryMed allows users to query many biomedical data sources from a single interface. The data sources that the
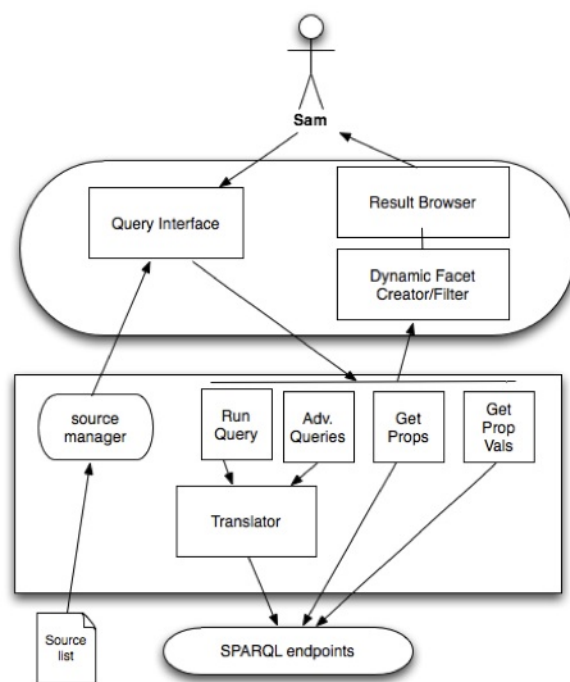
user wishes to query are not fixed, and can be extended very easily. The system was designed with non-expert users of SPARQL in mind, and presents a very intuitive user interface that anybody with basic biomedical domain knowledge can use.

In the following sections we explain the system functionality by first giving an overview of the QueryMed system architecture and the design decisions behind some of the crucial components of the system.

3.1 Architecture overview

A general overview of QueryMed architecture is shown in Figure 1. The main components of the system are the user interface and the proxy server that retrieves biomedical data from remote SPARQL endpoints. QueryMed relies on the following remote resources:

- **SPARQL Endpoints**: QueryMed runs queries on multiple remote SPARQL endpoints that expose biomedical data.

- **Sources List**: The default list of endpoints available to the user is stored in a source list on the proxy server.

3.1.1 Proxy Server

The user submits a query to the SPARQL endpoints via the proxy server. The query is translated into a SPARQL query for each individual endpoint, and the results are returned to the user. The specific components implemented in the proxy server are as follows.

- **Source Manager:** The source manager reads the source list and populates the default query list on the user interface. It also keeps track of the default endpoints, the currently selected endpoints, and the endpoints that have been dynamically added.

- **Translator:** This component is responsible for translating the user query into valid SPARQL syntax.

- **Run Query:** The purpose of this service is to execute SPARQL queries. It takes input on source and keyword from the client, and returns the query results as a JSON object. The structure of the JSON object is given in figure 4.

- **Get Properties:** This service takes as input an individual source, and returns an array of all properties for that source.

- **Get Property Values:** This service takes as input an individual source and a specific property. It returns a list of the possible values that this property can take.

- **Advanced Queries:** This service takes as input list of sources, properties, query terms, and logical operators, which are passed to the proxy server as a JSON object. The structure of this JSON object is given in figure 5. After constructing and executing the SPARQL query, it returns a set of query results as a JSON object in the format given in figure 4.

3.1.2 User Interface

The query interface is implemented in JavaScript, HTML and CSS. The main components of the user interface are as follows:

- **Initial Query Interface:** The user inputs their query term into the provided text box (figure 6).

- **Add New Source:** Allows the user to add a new data source to extend the query (figure 8)

- **Dynamic Facet Creator/Filter:** This component allows the user to select a set of data sources, load the properties available at these sources, and dynamically construct a complex SPARQL query connected by logical operators to take advantage of the structure of the data at each endpoint (figure 12).

- **Result Browser:** The user can view their query results, organized by source. The results are presented in a table with pagination. The user can choose the number of results viewed at a time, search the columns based on some text value and also sort the columns (figure 13).

3.2 System Functionality and Design Decisions

3.2.1 Source list

Because the set of default endpoints is stored on the proxy server, the set of resources available by default to the user can easily be updated. Since useful biomedical resources are rapidly being developed and made available as RDF triple stores, the ease of updating the resource list ensures that the system can easily be brought up to date. In fact, the QueryMed system could easily be adapted to perform queries outside the biomedical domain by modifying the list of input data sources.

```
SELECT ?projection_1 ?projection_2 ...
       ?projection_n
WHERE {?x source:property ? projection
       FILTER regex(? projection, '" +
       input +"', 'i)
}
... //Other property filers are to follow
```

Figure 3: Minimal SPARQL Query Structure in the "Query All" Case

```
{
    "bindings":
    [
            {"source" : "source_label",
             "uri" : "http://source.com/sparql.",
             "vars" : ["variable1","variable2"],
             "count":2,
             "results" :
                     [
                        {"variable1": "Value 1",
                      "variable1": "Value 2"},
                         {"variable1": "Another
                     Value 1",
                       "variable1": "Another Value 2"},

                     ]
            }
            ,
        { //Another Source and Result Description
        ...
        },
        ...
    ]
}
```

Figure 4: JSON returned from the RunQuery service

3.2.2 Proxy Server

While an entirely client side application is possible, we chose to have a proxy server perform the SPARQL query execution and caching. This design is advantageous for several reasons: 1. Efficient cache management: The result set from running an unrestricted query can include millions of results. It may be infeasible to keep unfiltered query results in browser memory. A typical memory footprint for a browser (for e.g. Firefox) is usually between 20MB and 100MB. A poorly constructed query can result in gigabytes worth of triples returned, causing the browser to crash. The proxy server can cache results from initial query execution, so that only a filtered result set is subsequently returned to the client. 2. The proxy server enables us to avoid cross domain XML-HTTP-Request errors in accessing SPARQL endpoints in various domains.

3.2.3 Data Structures

The parameters required for constructing the SPARQL queries are sent from the user interface to the proxy server.

```
{
    "http://some-source/sparql": {
            "property1": {"value1": "AND"},
        "property2": {"value 2": "OR"},
        "property2": {"value 3": "FILTER"}
                    }
            }
    //Other sources ...
}
```

Figure 5: JSON constructed to send the structure of a SPARQL query for an advanced query with logical operators

In the general "Query All" case, the system will take the user-specified query term as text-box input, filter on all the triples available at the SPARQL endpoint to select only those that contain the keyword, and display the results. The generated SPARQL query will be of the form illustrated in figure 3. The word "input" in the figure represents the keyword specified by the user.

When the results are returned, each result set is structured as the JSON object given in figure 4. This object identifies the endpoint where the query was executed, the URI of the endpoint, the query variables, how many results are returned and the result set.

When the user runs an advanced query, the payload to be sent to the proxy server to perform the SPARQL query construction and query execution orchestration is far more complex than in the "Query All" case. The system must specify what data sources are to be queried, what properties are queried, the values for each of the selected properties, and how these property-value pairs relate to the original user-specified query term. This information is passed as a JSON object, whose structure is given in figure 5, to the proxy server.

3.2.4   Query Interface

In order to make QueryMed easy to use, but still provide a flexible tool that is able to perform a range of queries, we provide an uncluttered interface with only a blank search box and two buttons. The "Query All" button will perform a keyword-based query over pre-selected data sources. The advanced search option "Refine Query" allows the user to view additional query options. The initial query interface is shown in figure 6.

We believe that the unstructured keyword search is likely to encourage users without previous experience running queries on semantic web data to explore the additional features of our system that allow more flexible and advanced queries. For example, physicians who are not used to running queries on semantic web data may be more willing to invest time into learning how to find better search results once they see that the system can easily provide them with some initial useful query results using the basic keyword search option. However, the execution of queries using QueryMed will be very much different from a user issuing a web search on a search engine such as Google, because it exploits the underlying structure of the data as exposed in the individual data endpoints.

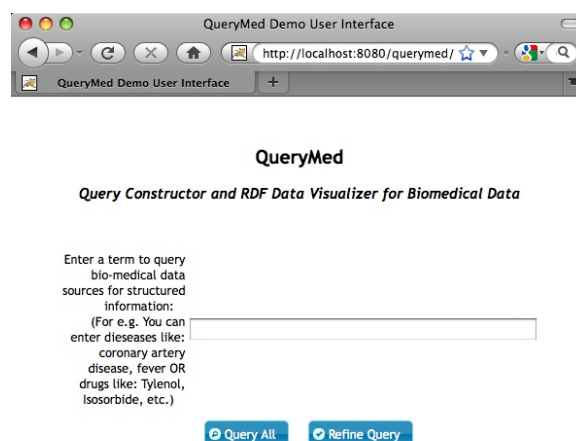The advanced query interface provides the user with the

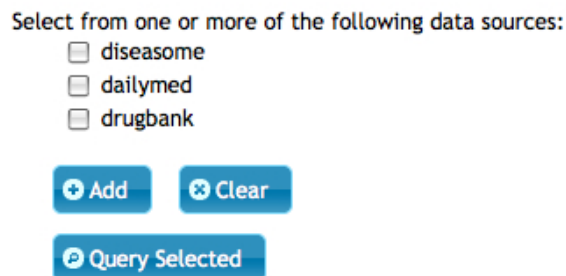**Figure 6: Initial query interface seen by the user**

Figure 7: The user has the option of selecting specific trusted or relevant data sources, or of adding additional data sources to query.

option of selecting specific data sources to query, as shown in figure 7. The user can also dynamically add additional data sources, as shown in figure 8. This feature allows the user to perform their query only over relevant or trusted data sources. The ability to dynamically add additional sources increases the flexibility of the QueryMed system, allowing users to search endpoints of interest that are not included in the default list.

When a data source is selected, the properties list is automatically populated with the properties available at the selected endpoint so that the user can run queries specific to the underlying structure of the RDF data in order to find more relevant query results. The properties list is generated by running a SPARQL query of the form given in figure 9 at the specified data source. This allows users to improve their queries using the underlying structure of the data without prior knowledge of this structure.

Once the properties are returned, those will be displayed in the user interface as shown in figure 10. Clicking an individual property link displays a description of the property, allowing the user to understand the keywords that will be most useful to search on each specific property.

The user can choose to perform exact queries, or filter re-

Figure 10: The Advanced Search feature allows the user to perform exact or pattern-matching queries connected by user-specified logical operators over specific properties in given resources, taking advantage of the structure in the RDF data.

Figure 8: The user can also dynamically add new data sources

```
SELECT DISTINCT ?property
WHERE { [] ?property [] }
ORDER BY ?property
```

Figure 9: SPARQL query to retrieve all properties of a given datasource

sults on specific keywords. If the user does not know the specific value for a property, they can instead specify a keywords using the FILTER option. They also can choose logical operators to connect the various parts of their query. When AND is used it will be appended as a basic triple pattern to the SPARQL query, i.e. as a conjunction. When OR is used the specified graph pattern is made to disjunct with the rest of the query with the SPARQL UNION operator. If no logical operator is specified, AND will be used by default. The advanced query feature is capable of dynamically constructing complex SPARQL queries, such as the query shown in figure 11.

The Dynamic Facet Creator shown in figure 12 allows the user to view properties from multiple endpoints, and to expand and contract properties for individual endpoints. This feature enables the interface to display many property lists simultaneously, grouped by source, so that only information relevant to the endpoint the user is currently examining will be visible at any time.

Query results are displayed to the user as a table, which can be filtered and searched to refine the query result, or printed as shown in in figure 13. This feature allows the user to perform additional filtering on the query results to display the most relevant results, and is particularly useful for refining queries that return a large number of results.

3.2.5 Implementation

QueryMed is implemented in Java in the backend and JavaScript in the frontend. The JQuery library [16] was used to develop an attractive user interface. In the backend,

```
SELECT distinct ?disease WHERE {
{?x <http://www.w3.org/2000/01/rdf-schema#label>
?disease
FILTER regex(?disease,
"coronary artery disease", "i").
?x <http://www4.wiwiss.fu-berlin.de/diseasome/
resource/diseasome/class>
<http://www4.wiwiss.fu-berlin.de/diseasome/
resource/diseaseClass/Cardiovascular>}
UNION { ?x <http://www4.wiwiss.fu-berlin.de/
diseasome/resource/diseasome/associatedGene>
 <http://www4.wiwiss.fu-berlin.de/diseasome/
 resource/genes/ABCA1>.}
}
```

Figure 11: A complex SPARQL query that takes advantage of the underlying data structure can be dynamically constructed using the advanced query feature of the QueryMed system.
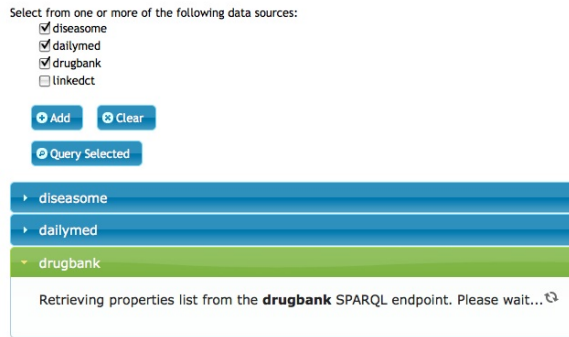

Figure 12: Dynamic Facet Creator and Filter

Figure 13: A sample result table

	Diseasome	Dailymed	Drugbank
1st Trial	3.45 s	1.77 s	9.51 s
2nd Trial	1.61 s	1.57 s	9.34 s
3rd Trial	1.71 s	1.66 s	9.06 s

Table 1: Running times to retrieve all the properties from selected endpoints

the Jena library [13] is used to run the SPARQL queries.

3.3 A Sample Use Case

A physician is interested in finding semantic web resources related to coronary artery disease. She first tries a basic search over all default resources, by entering "coronary artery disease" in the input search box. She then sees a list of disease names in the Diseasome database and drugs in DailyMed and Drugbank that relate to coronary artery disease displayed in a table. She can then filter the results using additional search terms. For example, she knows that the route of administration of the drug that she is looking for is injection, so she filters the drug query results on the route of administration field using the query term "injection." She then prints the table of results.

She is now interested in finding relevant clinical trials for her patient. However, the clinical trial database LinkedCt is not in the default set of endpoints, so she selects the "Refine Query" option to choose additional endpoints to search. She sees a list of default endpoints, and selects the "Add" option to include an additional endpoint. After entering the name and URL of the LinkedCT endpoint, she is able to search for clinical trials for which her patient may be eligible.

She is also interested in further refining her search. She uses the advanced search option to construct a boolean query that takes advantage of the underlying structure of the RDF data in the database. She searches diseasome for a list of diseases whose class is "Cardiovascular" or for which the associated gene is "ABCA1." Using the QueryMed advanced search interface, the complex SPARQL query corresponding to her question is automatically constructed, and she can view the query results conveniently displayed in a table.

A video illustrating this sample use case can be found at: http://www.youtube.com/watch?v=JCr7vryS9Vg

3.4 Performance

The slowest step in running queries using the QueryMed system is populating the property values for each selected endpoint. We compared the times required to load properties from several endpoints, by running the query illustrated in figure 9. Timing data are shown in table 1.

For all selected endpoints, the property values take longer to load on the initial query than in subsequent iterations, probably due to browser caching. The running time to acquire the list of property values for the Drugbank endpoint is much larger than the time for the other two endpoints, probably due to the greater size of the Drugbank database. (As of 12/10/09, Diseasome contains 91,182 triples, DailyMed contains 164,276 triples, and DrugBank contains 765,936 triples).

3.5 QueryMed Resources

The source code for the Querymed system is available at the QueryMed Google Code project:

http://code.google.com/p/querymed/

4. RELATED WORK

A number of existing tools aim to provide a user-friendly interface for browsing semantic web data, or to allow users to perform federated queries. Several of these are described below.

The SMART query tool is a web-based application designed to allow biologists to run SPARQL queries over multiple endpoints. Queries to the SMART system are constructed using a descriptive logic query written in the natural-language like Manchester OWL syntax [8]. However, it cannot be expected that the end users know how to use the Manchester OWL syntax to write queries, whereas in QueryMed, users can construct queries intuitively by giving keywords for user selected properties.

GoWeb is another system designed for answering queries on biomedical data. It allows users to run traditional keyword-based web search with ontology search features [10]. After a keyword search, documents can be filtered based on the biomedical annotations they contain. However, in GoWeb the exact sources queried are not transparent and cannot be selected or modified by the user as in QueryMed.

Twinkle offers a stand-alone graphical user interface to load and edit SPARQL queries that can be used to query online SPARQL endpoints [11]. Our system differs from the Twinkle system in several aspects. First of all, in Twinkle, the user is expected to know what is already available at the SPARQL endpoints to write the query. But in QueryMed, we only ask for specific keywords of interest, and give the option of restricting the query should the user wish to run a more precise query. Second, although Twinkle was designed to be a more general purpose system, it only supports a small number of specific SPARQL endpoints, while QueryMed allows the user to dynamically add SPARQL endpoints.

Most SPARQL query engines are designed to run queries against individual endpoints. But it is often useful to draw on multiple web resources in answering a query. There are a number of systems, including the DARQ [15] and CALO query manager [7] systems, designed to allow the user to run integrated queries against multiple SPARQL endpoints.

Table 2 compares selected features of the QueryMed system with other related systems. The QueryMed system was unique among the systems that we found in that it allows endpoints to be dynamically added by the user. Other features of the QueryMed system that distinguished it from similar systems included the ability both to perform keyword queries and to construct more advanced queries taking advantage of the structure of the semantic web data. This feature increases the ease of use of our system relative to other similar systems. Furthermore, the Javascript based user interface of the QueryMed system, implemented using the JQuery library, makes our user interface particularly attractive, easy to interact with, and capable of handling a variety of user input events. Another unique feature of our system is the property-based advanced query interface. This enables users to take advantage of the structure of the underlying ontologies used to represent the data without prior knowledge of the ontology structures.

5. FUTURE WORK AND CONCLUSIONS

The bottleneck in the running time of our system is run-

ning queries that must retrieve many triples from slow remote SPARQL endpoints. By caching relevant data locally, the running time could be significantly improved. We observed that the QueryMed system performs significantly faster the second time that a query is run on a given resource possibly due to browser caching. By managing our own cache to contain the data most likely to be needed on the proxy server, we could reduce running time still further. Another approach that could reduce running time still further, especially for users with a slow network connection, would be to create a complementary standalone application that gives the user the option at startup time of loading all the required RDF data. Since data is stored locally after the initial startup, using the system in subsequent queries will be rapid after an initial loading phase. This approach might be too memory-intensive if the user wishes to run queries over many large triple stores, but might work best in the situation where there is a small or moderate amount of data in the repositories of interest to the user.

Our system currently allows only a restricted set of SPARQL queries. Supporting additional types of queries and including query optimization functionality could increase both the flexibility and speed of the QueryMed system. By drawing on the expressivity of the SPARQL language and the information contained in the ontologies used to represent the data, it would be possible to extend our system into an intelligent reasoning system on biomedical data which allows physicians to enter sophisticated, complex queries and find relevant biomedical results.

We also believe that it would be useful to allow exploration of the relationships among multiple data sources containing similar resources. One challenge in integrating biomedical data across multiple data sources is that individual data items (i.e, a specific protein or a specific drug) may be represented by distinct URIs at different endpoints. The diversity of representations of identical data items across different biomedical data sources makes it difficult to automatically combine these items. However, there is some cross-referencing between the biomedical data sources that we examined (for example, DailyMed drugs sometimes refer to diseases in the Diseasome SPARQL endpoint by their URI). It might be useful to provide a representation of the relationships among search results from different sources with the query results. Additionally, the system could automatically detect similar data items even if they are identified by distinct URIs, and group similar items from distinct data sources together in the query results by natural language processing. Another useful feature might be to allow the user to view the relationships among distinct URIs (as in the relfinder system [12] , which displays possible paths through the RDF graph between distinct resources).

Another feature that we have begun to implement in our system is an autocompletion feature to automatically retrieve the distinct values for any given property. This feature will allow the user to automatically see all valid choices for each property, and will make the system easier to use and reduce empty query result sets.

Since a major goal of QueryMed is ease of use by physicians, life scientists, and patients, we also plan to perform a user study to understand how effectively users without knowledge of SPARQL can interact with our system. We plan to use data from this study to further refine our system to improve its usability.

	Query Multiple Sources?	Dynamic Addition of Sources	Allows Keyword Queries	Open Source	GUI
QueryMed	Yes	Yes	Yes	Yes	Yes
SMART	Yes	No	No	Yes	Yes
DARQ	Yes	N/A	No	Yes	No
GoWeb	Yes	No	Yes	No	Yes
BioGateway	Yes	No	No	No	Yes
Twinkle	Yes	No	No	Yes	Yes

Table 2: Comparison of selected features of the QueryMed system with other related systems.

The main contributions of our system are: dynamic construction of complex SPARQL queries based on intuitive user input; dynamic addition of user-specified endpoints; and ability to run queries over multiple endpoints. Because of the unique features of our system, we believe it will be of use to the biomedical community.

7. REFERENCES

[1] Dailymed, http://dailymed.nlm.nih.gov/dailymed/.

[2] Diseasome, http://www4.wiwiss.fu-berlin.de/diseasome/.

[3] Drugbank, http://www.drugbank.ca.

[4] Gene ontology, http://www.geneontology.org.

[5] Linkedct, http://linkedct.org/sparql.

[6] Pubmed, http://pubmed.bio2rdf.org/sparql/.

[7] J. L. Ambite, V. K. Chaudhri, R. Fikes, J. Jenkins, S. Mishra, M. Muslea, T. E. Uribe, and G. Yang. Design and implementation of the calo query manager. In *AAAI*, 2006.

[8] A. D. L. Battista, N. Villanueva-Rosales, M. Palenychka, and M. Dumontier. Smart: A web-based, ontology-driven, semantic web query answering application. In *Semantic Web Challenge*, 2007.

[9] T. Berners-Lee. Relational databases on the semantic web, design issues. 1998.

[10] H. Dietze and M. Schroeder. Goweb: a semantic search engine for the life science web. *BMC Bioinformatics*, 10 Suppl 10, 2009.

[11] L. Dodds. Twinkle: A sparql query tool, http://www.ldodds.com/projects/twinkle/.

[12] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. Relfinder: Revealing relationships in rdf knowledge bases. In T.-S. Chua, Y. Kompatsiaris, B. Mrialdo, W. Haas, G. Thallinger, and W. Bailer, editors, *SAMT*, volume 5887 of *Lecture Notes in Computer Science*, pages 182–187. Springer, 2009.

[13] B. McBride. Jena - a semantic web framework.

[14] C. Pasquier. Biological data integration using semantic web technologies. *Biochimie*, 90:584–594, 2008.

[15] B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *ESWC*, pages 524–538, 2008.

[16] J. Resig. Jquery –javascript library.

[17] Y. Yip. Accelerating knowledge discovery through community data sharing and integration. *Yearb. Med Inform.*, 2009.