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 used in describing the datasets, or the SPARQL query language to query the data. 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. 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 quantity of publicly available data in the biomedical domain has dramatically increased over recent years. Publicly available biomedical resources include data on drug discovery [23, 16], clinical trials, diseases, disease genes, and phenotypes. With the linked open data movement, the semantic web community has been very proactive in converting these rich information resources to RDF [8]. In fact, the biomedical domain is among the early successes of the semantic web due to the rapidity with which the community has made its data available in RDF triple stores [25].

To allow end users to exploit the abundance of useful biomedical data that 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 that also allow users to run federated queries on multiple SPARQL endpoints. There is also a need for efficient hybrid interfaces that allow both browsing and querying [18], since many currently available systems are linked data browsers such as the Tabulator [24], which allow a user to navigate the data in an exploratory manner but lack support for filtering and querying the data in a user friendly manner.

Answering many medically and biologically relevant questions requires searching, filtering, and combining information from multiple 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.

Since the physician must search many databases in order to find an answer to her single question, she requires 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 modeling data in the biomedical domain. Section 3 describes our system. Section 4 discusses related work and illustrates how QueryMed differs from previous systems. Section 5 outlines future work giving an outlook for Querymed. Finally section 6 summarizes the contributions of our system.

WWW2010, April 26-30, 2010, Raleigh, North Carolina.
.

2. BACKGROUND

The semantic web can be viewed as a global database system for the information 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 [12].

Many major biological and biomedical data resources, including Gene Ontology, DailyMed, LinkedCT, and Diseasome, are currently available as RDF triplestores. Almost all of these data sources are interlinked with each other. 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 be a daunting task especially 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 are often distributed among many sources [20].

3. DESIGN & IMPLEMENTATION

The QueryMed system allows users to easily query multiple biomedical data sources. Queries can be run against a default list of datasources, or against a set of user-selected SPARQL endpoint. The end user can easily input additional endpoints in order to utilize resources that are not included in the default list. The system automatically translates the user input into a SPARQL query for each individual endpoint, combines the results, and returns them to the user. The user can choose to refine the query by iteratively modifying the original query terms, and by filtering the result list. The advanced query functionality of the QueryMed system allows the user to easily construct complex logical SPARQL queries that take advantage of the underlying structure of the ontologies. The simple user interface of the QueryMed system is designed to be intuitive for users with no knowledge of the SPARQL query language.

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

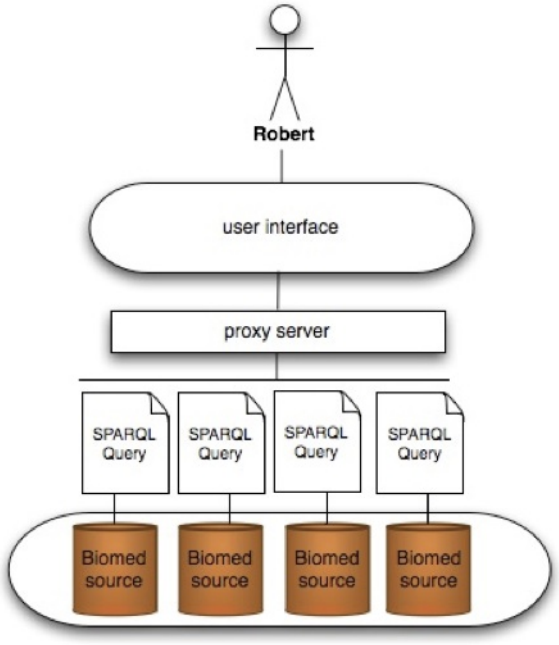

Figure 1: QueryMed Architecture Overview

3.1 QueryMed 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 takes input from the user and retrieves relevant biomedical data from remote SPARQL endpoints. After a user submits a query from the user interface, the query is translated by the proxy server into individual SPARQL queries for each remote endpoint. The query results are returned from the remote endpoints, combined by the proxy server, and presented to the user. A detailed illustration of the parts of the QueryMed system is shown in Figure 2.

3.1.1 User Interface

The QueryMed user interface is designed to be intuitive for the end user, yet flexible enough to permit a broad range of interesting queries. The basic query interface allows the user to run simple queries, and is designed for maximal ease of use. The advanced search capabilities allow the user to easily construct complex logical queries that take advantage of the underlying structure of the biomedical ontologies. The user interface also allows the user to iteratively refine queries, and displays the query results, which have been retrieved from multiple SPARQL endpoints. The main components of the user interface are as follows:

- **Basic Query Interface:** In order to make QueryMed easy to use, but also provide a flexible system that is capable of performing a broad variety of biomedically relevant queries, we provide an uncluttered basic search interface with only a blank search box and two buttons as shown in Figure 3. The "Query All" button will perform a keyword-based query over pre-selected

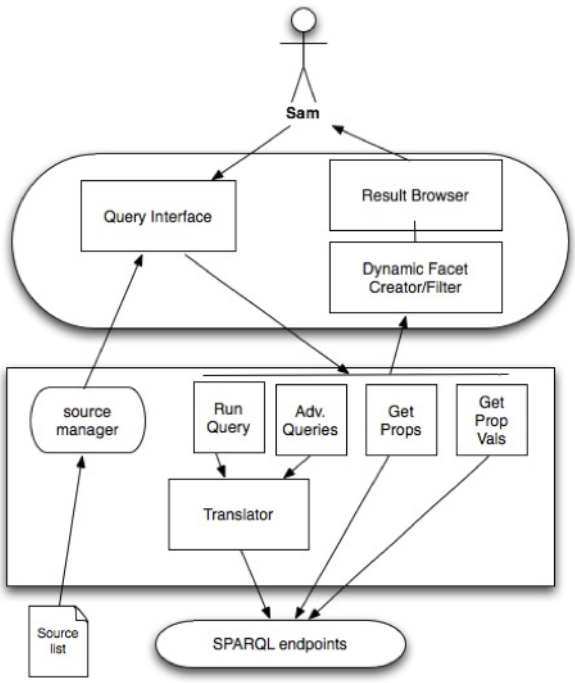

Figure 2: QueryMed Architecture Details

data sources. The advanced search option "Refine Query" allows the user to view additional query options that are handled by the Advanced Query Interface.

For example, a physician interested in finding semantic web resources related to coronary artery disease could use coronary artery disease as a search keyword in the input search box and select the "Query All" option. The simplicity of the basic query interface allows the physician to rapidly and easily browse semantic web resources of interest.

- **Result Browser:** The user can view his query results organized by source in the Result Browser. 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 as shown in Figure 9. This 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.

  For example, after searching for "coronary artery disease" in the basic query interface, the physician will see displayed in the Result Browser a list of disease names in the Diseasome database and drugs in DailyMed and Drugbank that relate to coronary artery disease. 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 interested in is injection, so she filters the drug query results on the route of administration field using the query term injection. The additional filtering capabilities of the Result Browser allows her to either browse a large number

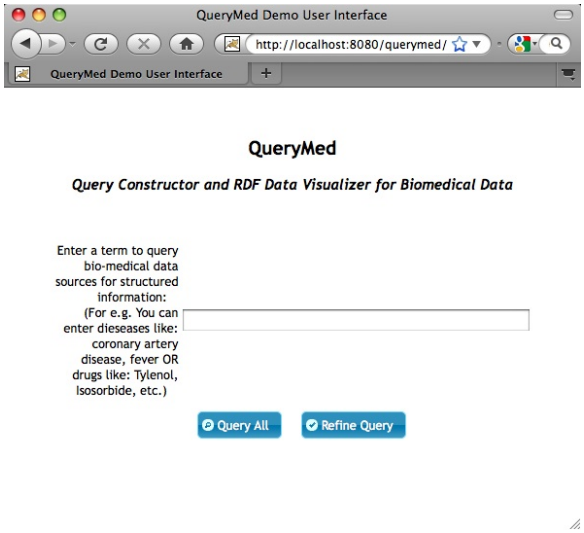

Figure 3: Initial Basic Query Interface

of query results, or refine her search to view only the most relevant query results.

- **Advanced Query Interface:** The Advanced Query Interface allows the user to add new data sources, and to construct a complex SPARQL query that takes advantage of the structure of the underlying biomedical ontologies. Thus, the end user can construct a targeted query without previous knowledge of the SPARQL language or the structure of the relevant biomedical ontologies.

  When the "Refine Query" in the Basic Query Interface is invoked, the user is provided with the default list of data sources in the system as shown in Figure 4. The user may select from this list relevant trusted data sources that they wish to query. The user can also dynamically add additional data sources as shown in Figure 5, allowing users to query endpoints of interest that are not included in the default list.

  A physician who is interested in using the QueryMed system to find relevant clinical trials for her patient can use the Advanced Query Interface to add additional relevant datasources . For example, she might want to use the clinical trial database LinkedCt, which is not in the default set of endpoints. So, she selects the Refine Query option to select the endpoints to search, and then the Add option to include an additional endpoint. After entering the name and URL of the LinkedCT endpoint, she can able to search for clinical trials for which her patient may be eligible.

  When a data source is selected, the properties list is automatically populated with all distinct properties available at the selected endpoint. If the user wants to restrict her query based on one or more properties available, she can specify values for the desired properties. Because the QueryMed system automatically displays a list of properties to the user, users can improve their queries using the underlying structure of the data to obtain more relevant query results.

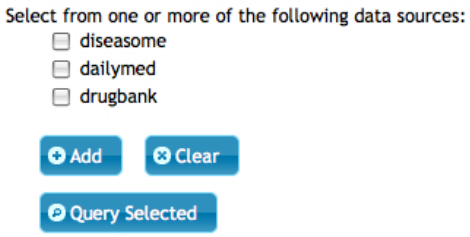

Figure 4: The user has the option of selecting specific trusted or relevant data sources, or of adding additional data sources to query.

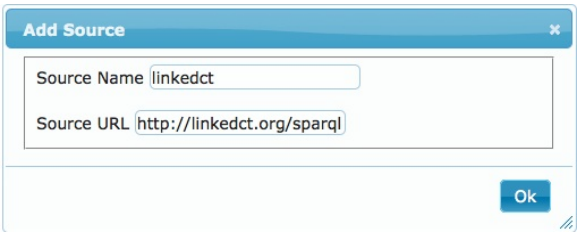

Figure 5: Add New Data Source

Once the properties are returned, they will be displayed in the user interface as shown in Figure 6. Clicking an individual property link displays a description of the property. This allows 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 results on specific keywords. If the user does not know the specific value for a property, she can instead specify a keywords using the FILTER option. She 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 7.

For example, the physician might want to further refine her search for cardiovascular diseases. 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 (Figure 7), and she can view the query results conveniently displayed in the Results Browser.

- **Dynamic Facet Creator/Filter:** The Dynamic Facet Creator/Filter allows the user to select a set of data

```
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 7: 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.

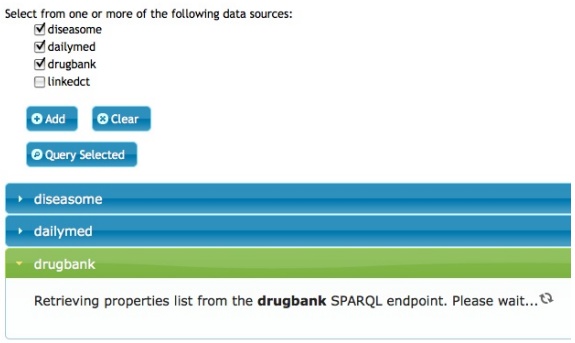

Figure 8: Dynamic Facet Creator and Filter

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 8). The interface allows displaying of many property lists from different sources simultaneously (grouped by source). Therefore, only information relevant to the endpoint the user is currently examining will be visible at any given time.

3.1.2 Proxy Server

The proxy server acts as an intermediary between the user interface and the remote SPARQL endpoints. Its functionality is twofold:

1. Execute the SPARQL queries at the relevant remote SPARQL endpoints and consolidate the results to be presented in the user interface.

2. Cache the results of the current query so that refinements of the query will have reduced network and query execution latency.

The specific components in the Proxy Server are as follows.

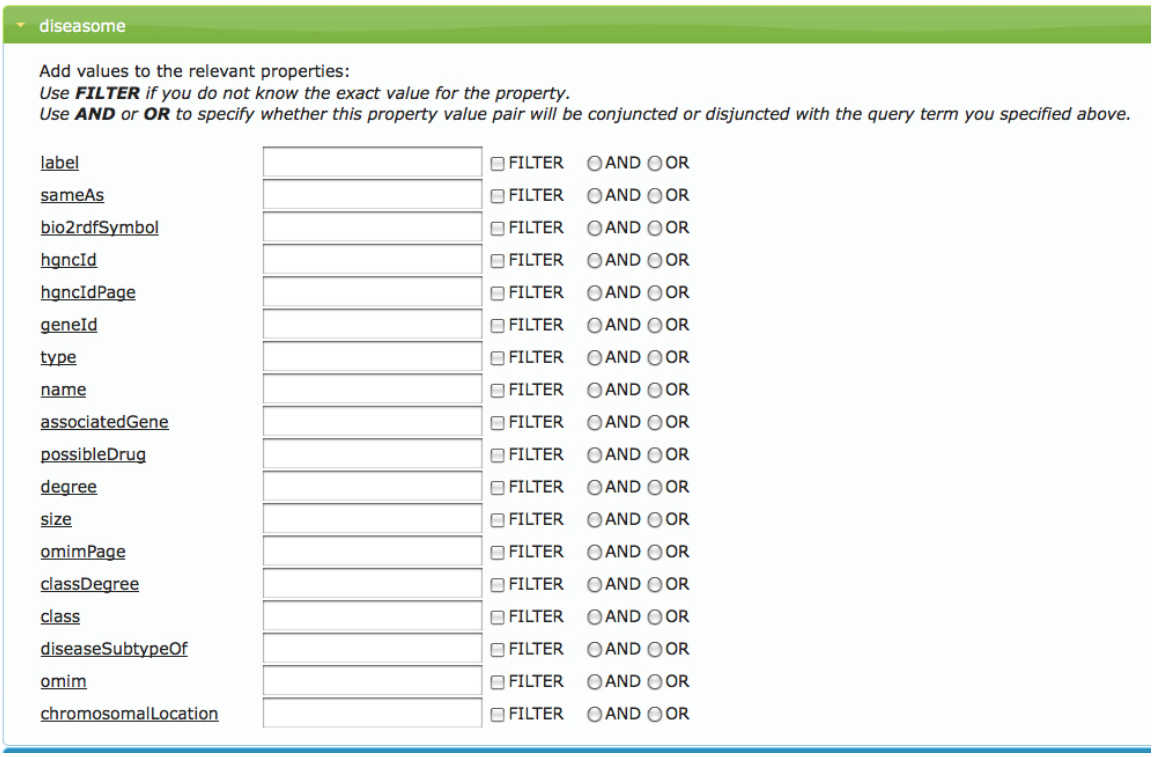

Figure 6: The Advanced Search feature allows the user to perform exact or pattern-matching queries connected by user-specified logical operators over specific properties in given resources, taking advantage of the structure in the RDF data.

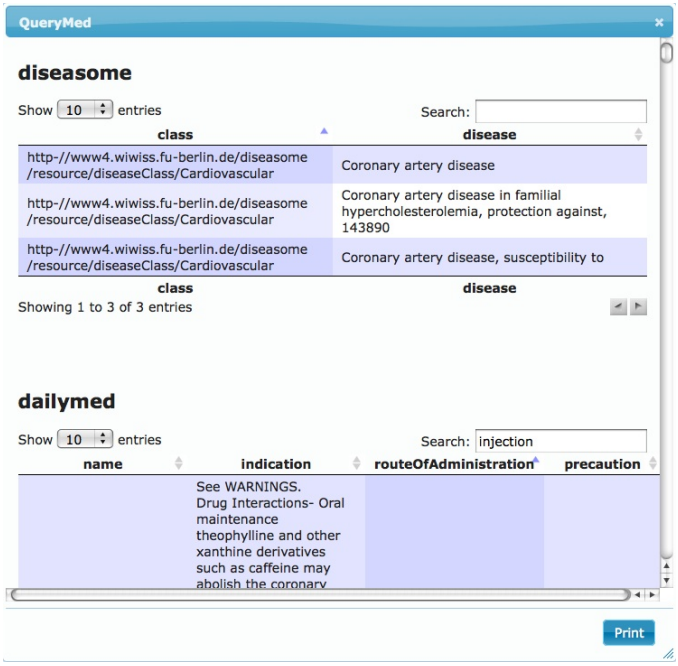

Figure 9: A sample result table

- **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:** The Translator is responsible for translating the user query into valid SPARQL syntax. The Translator obtains the parameters to construct the query from the input the user specifies in the Query Interface, and dynamically constructs a SPARQL query based on the user input. The Translator relies on two services:

  1. **Run Query:** The purpose of this service is to execute SPARQL queries generated by the Translator, and return the query results as a JSON object.
  2. **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.

- **Additional Services:** In order to make the user interface more user friendly, the following two services perform some "behind the scenes" work.

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

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

Figure 10: Minimal SPARQL Query Structure in the "Query All" Case

2. **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.

3.2 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.

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. 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 10. The word "input" in the figure represents the keyword specified by the user.

When the results are returned, each result set is structured as a JSON object. 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 proxy server takes as input the data sources to be queried, the properties to be queried, the values for each of the selected properties, and the relationship between the property-value pairs and the original user-specified query term. This information is passed to the proxy server as a JSON object.

3.2.4 Query Interface

The basic query interface is designed to be as simple as possible, so that users who have little previous experience running queries on SPARQL endpoints can easily and rapidly find query results. The advanced query functionality of our system allows users the flexibility to construct complex SPARQL queries. The QueryMed system allows the user to construct queries that take advantage of the underlying structure of the data exposed by the individual data endpoints, without requiring previous knowledge of this structure.

3.2.5 Implementation

QueryMed is implemented in Java in the backend and JavaScript, HTML and CSS in the frontend. In the backend, the Jena library [19] is used to run the SPARQL queries and 4-store [15] is used as a triple store to provide caching support. The JQuery library [22] was used to develop an attractive user interface.

3.3 Performance

We observed that 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 11. Timing data are shown in Table 1.

For all selected endpoints, without local data (i.e. no cache on the proxy server), the property values took longer to load. The difference in running time between *with local data* and *without local data* is approximately three orders of magnitude. The network latency and the slow performance of the remote SPARQL endpoints as compared to the local data cache can be accounted for the increase in running time for the queries in the two cases. We also noted that the running time of the subsequent iterations of the same query is significantly less than the initial query, probably due to browser caching. When comparing the query execution time amongst the data sources, we see that the Drugbank takes the longest, probably due to the greater size of the Drugbank data as compared to the other three data sources [1].

Therefore, it is fair to say that bottleneck in the running time of our system is running queries that must retrieve many triples from slow remote SPARQL endpoints. By managing our own cache to contain the data most likely to be needed on the proxy server, we were able to reduce running time further.

[1]As of February 15th 2010, Diseasome contains 91,182 triples, DailyMed contains 164,276 triples, Sider contains 192,515 triples, and DrugBank contains 765,936 triples) [2, 1, 7, 3]

	Dieseasome		Dailymed		Drugbank		Sider	
	Without Caching	With Caching	Without Caching	With Caching	Without Caching	With Caching	Without Caching	With Caching
1st Trial	3.45 s	134 ms	1.77 s	384 ms	9.51 s	60 ms	14.64 s	23 ms
2nd Trial	1.61 s	31 ms	1.57 s	32 ms	9.34 s	31 ms	2.94 s	7 ms
3rd Trial	1.71 s	7 ms	1.66 s	11 ms	9.06 s	23 ms	2.86 s	6 ms

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

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

Figure 11: SPARQL query to retrieve all Properties. This was used to measure the performance of implementing a local cache.

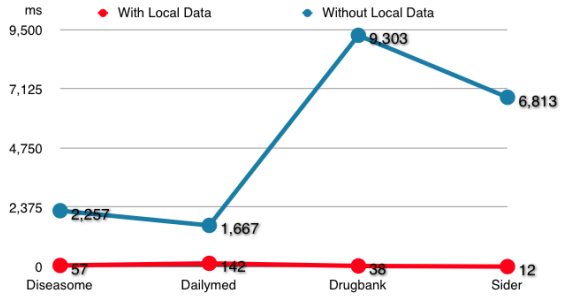

Figure 12: Comparison of Query Execution Times at Various SPARQL Endpoints With and Without Local Data (Cache)

3.4 QueryMed Resources

The source code for the Querymed system is available at the QueryMed Google Code project: http://code.google.com/p/querymed

A video illustrating a sample use case can be found at: http://dig.csail.mit.edu/2010/Papers/www-ws-colab-science/videos

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 [11]. Major differences between the QueryMed system and SMART include the ability to construct queries intuitively by specifying keywords for user selected properties, to iteratively refine query results, and to dynamically add additional endpoints in the QueryMed system.

GoWeb and BioGateway are two additional systems designed for answering queries on biomedical data [13, 10]. GoWeb allows users to a hybrid search, running keyword-based queries and then filtering based on ontological concepts. However, while GoWeb functions as a search engine that incorporates ontological background knowledge to improve search results, QueryMed is a query translation system. BioGateway provides a web interface to query a provided SPARQL endpoint that integrates multiple biomedical resources, but does not allow the user to run queries over multiple endpoints or to dynamically add endpoints.

Twinkle offers a stand-alone graphical user interface to load and edit SPARQL queries that can be used to query online SPARQL endpoints [14]. 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 [21] and CALO query manager [9] 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, and provides a hybrid interface that enables the users to both query and browse data. Other features of the QueryMed system that distinguish it from similar systems include the ability both to perform keyword queries and to construct more advanced queries taking advantage of the structure of the 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 interface 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

Our system currently allows only a restricted set of SPARQL queries. Supporting additional types of queries and includ-

	Hybrid Interface (Combines Querying & Browsing)	Provides Local Caching	Queries Multiple Sources	Dynamic Addition of Sources	Allows Keyword Queries	Open Source	GUI
QueryMed	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SMART	No	Yes	Yes	No	No	Yes	Yes
DARQ	No	No	Yes	N/A	No	Yes	No
GoWeb	Yes	No	Yes	No	Yes	No	Yes
BioGateway	Yes	Yes	Yes	No	No	Yes	Yes
Twinkle	No	No	Yes	No	No	Yes	Yes

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

ing 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.

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 a situation where there is a small or moderate amount of data in the repositories of interest to the user.

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 [17] , 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.

6. CONCLUSION

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. We also believe that systems such as such as QueryMed, which make SPARQL endpoints easily accessible to end users, will entice more people to expose their data as linked open data.

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] Sider, http://www4.wiwiss.fu-berlin.de/sider/.

[8] W3c sweo community project, linking open data.

[9] 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.

[10] E. Antezana, W. Blondé, M. Egana, A. Rutherford, R. Stevens, B. De Baets, V. Mironov, and M. Kuiper. Structuring the life science resourceome for semantic systems biology: lessons from the BioGateway Project. *PSWAT4LSBurger A, Paschke A, Romano, et al, eds*, 435, 2008.

[11] 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.

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

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

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

[15] Garlik. 4store, an efficient, scalable and stable rdf database, http://4store.org/.

[16] C. Goble and R. Stevens. State of the nation in data integration for bioinformatics. *J. of Biomedical Informatics*, 41(5):687–693, 2008.

[17] 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.

[18] A. Jentzsch, B. Andersson, O. Hassanzadeh, S. Stephens, and C. Bizer. Enabling tailored therapeutics with linked data. In *World Wide Web Conference: Linked Data On the Web Workshop*, 2009.

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

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

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

[22] J. Resig. Jquery –javascript library, http://jquery.com.

[23] M. Sharp, O. Bodenreider, and N. Wacholder. A framework for characterizing drug information sources. 2008.

[24] Tim Berners-Lee and James Hollenbach and Kanghao Lu and Joe Presbrey and Eric Prud'ommeaux and mc schraefel. Tabulator Redux: Browing and Writing Linked Data . In *Linked Data on the Web Workshop at WWW08*, 2008.

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