8-2008

TcruziKB: Enabling Complex Queries for Genomic Data Exploration

Pablo N. Mendes
Wright State University - Main Campus

Bobby McKnight

Amit P. Sheth
Wright State University - Main Campus, amit@sc.edu

Jessica C. Kissinger

TcruziKB: Enabling Complex Queries for Genomic Data Exploration

Pablo N. Mendes[1], Bobby McKnight[2], Amit P. Sheth[1], Jessica C. Kissinger[3]

[1] *Kno.e.sis Center*
Department of Computer Science and Engineering
Wright State University, Dayton, OH
{mendes.2,amit.sheth}@wright.edu

[2] *LSDIS Laboratory*
Department of Computer Science
University of Georgia, Athens, GA
mcknight@cs.uga.edu

[3] *The Department of Genetics, Center for Tropical and Emerging Global Diseases and Institute for Bioinformatics*
University of Georgia, Athens, GA
jkissing@uga.edu

Abstract

We developed a novel analytical environment to aid in the examination of the extensive amount of interconnected data available for genome projects. Our focus is to enable flexibility and abstraction from implementation details, while retaining the expressivity required for post-genomic research. To achieve this goal, we associated genomics data to ontologies and implemented a query formulation and execution environment with added visualization capabilities.

We use ontology schemas to guide the user through the process of building complex queries in a flexible Web interface. Queries are serialized in SPARQL and sent to servers via Ajax. A component for visualization of the results allows researchers to explore result sets in multiple perspectives to suit different analytical needs.

We show a use case of semantic computing with real world data. We demonstrate facilitated access to information through expressive queries in a flexible and friendly user interface. Our system scores 90.54% in a user satisfaction evaluation with 30 subjects. In comparison with traditional genome databases, preliminary evaluation indicates a reduction of the amount of user interaction required to answer the provided sample queries.

1. Introduction

The contemporary scientist, when formulating a hypothesis or looking for evidence to validate one, commonly performs intensive queries of multiple independent online databases. The technology associated with the current information sources invariably require high levels of human involvement to manually browse through an extensive mass of data, observe the evidence, analyze their interconnections and draw conclusions. This is a time consuming and difficult task due to the size, diversity and complexity of both the data and the tools.

We propose an ontology-based analytical environment to provide greater flexibility and abstraction from implementation details, while retaining the expressivity required for post-genomic research. The richer representation model of semantic Web ontologies and associated query languages unveil new ways to approach database queries and knowledge discovery. The use of ontologies in this work goes beyond the reference to a standardized vocabulary as is commonly seen in bioinformatics. We explore the representation of conceptual relationships between entities to help guide the researcher with "connecting the dots" and take hypotheses to evidence.

The use case described in this work is a real-world effort, focused on the human parasitic protist called *Trypanosoma cruzi*. Approximately 18 million people, predominantly in Latin America, are infected with the *T.cruzi* parasite[1], the cause of Chagas Disease for which there is no definitive chemotherapeutic treatment. The parasite has a complex life-cycle with four main stages occurring in two hosts. In the insect host, *T.cruzi* is found in the form of epimastigotes and metacyclic trypomastigotes. In the vertebrate host, it is found in the form of bloodstream trypomastigotes and intracellular amastigotes.

Research on *T.cruzi*, like many other organisms and diseases, has reached a critical juncture with the quantities of experimental data being generated by labs around the world outstripping the community's ability to synthesize and understand all. In an attempt to provide integrated resources for the research community, researchers have created "Omics" databases that specialize in gathering as much information as possible about one organism, or a group of related organisms. TcruziDB [1] is an integrated genome database for *T.cruzi* that incorporates and provides online access to sequence data, "Omics" data and supplementary bioinformatics analyses on the Web. We have used semantic computing technologies to create a knowledge base called TcruziKB (The *Trypanosoma cruzi* Knowledge Base) as a complement to TcruziDB's existing functionality.

In comparison with traditional genome databases, which are based on flat files or relational databases, TcruziKB offers several advantages. (i) **flexibility on the query perspective**. Common query interfaces reflect the possible uses of the system as designed by the development team and/or the community of users involved in the development stages. For instance, TcruziDB.org offers 5 perspectives, where the user can search for/among different data types, e.g. genes, ESTs, contigs, scaffolds and ORFs. In such a

[1] http://www.who.int/tdr/diseases/chagas/diseaseinfo.htm

system, the user is limited to the available data types and existing "pre-canned" queries. Complex queries are generated in the query history when the user performs Boolean operations on sets of independent queries to find their union or intersection. For example, a query for all genes could be intersected with a query for proteins expressed in blood-stage forms of the infection to find the desired subset of genes. To generate new queries however, requires, human involvement by a database designer to implement the necessary SQL statements and visualization interfaces. In contrast, TcruziKB uses ontology schemas to offer a high-level query system, where the user is guided by the system throughout the process of posing a question in a logical path such as: "?protein → expressed_in → ?any_stage → resident_in → Blood". (ii) **complex query handling**: a key component of genomics analyses is the concept of "relationship". Through the use of ontologies a user will be able to ask questions not only about entities, but also about how those entities are related. For instance, what is the relationship between a given protein and blood? Such a question could possibly reveal protein expression evidence of an interaction between a life-cycle stage of the parasite where the organism resides in the bloodstream of the host and interacts with or modifies its environment in some way. This type of query is viable through the use of ontologies to reveal semantic associations, and is very difficult to institute within the scope of a relational database. In relational databases, although relationships may be present in the form of join tables or foreign keys, they cannot be easily queried (iii) **loosely-coupled web aggregation** of distributed data sources: most genome databases aggregate data from different sources at some level, usually by downloading, parsing and storing external data in a centralized relational database. In our system we provide loosely-coupled dynamic integration with external sources on the client side. Through the use of Ajax[2] and SPARQL protocol[3], our system is able to dynamically query multiple sources and "mash up" the results on one page for user visualization.

In addition to the provision of data aggregation for query capabilities, we aim at facilitating the difficult task of making sense of the information at hand. We implemented multiple exploration interfaces to allow the user to visualize and analyze query results in different formats: (i) the **tabular explorer** lists the results in a spreadsheet format, providing prompt access to all attributes of a group of items, allowing for sorting and filtering of data in a well known and widely used interface style. (ii) the **graph explorer**, focuses on relationships, drawing each item and value as nodes, while the attributes and relationships are represented as edges. This perspective brings connectivity to the first level, allowing the researcher to unveil hidden relationships within the data. (iii) the **pie-chart explorer** offers a summary of the results at a first glance, to allow researchers to understand the general characteristics of the data set, before more specific questions are posed.

The W3C Semantic Web Health Care and Life Sciences (HCLS) Interest Group recently released a questionnaire (http://www.w3.org/2007/06/HCLSForm) to assess research areas and draft charters for Working Groups. With respect to that questionnaire, this project meets the following activities: it bridges and enhances biomedical vocabularies; works with data providers to make data available in RDF; explores/develops navigation and visualization tools; explores/develops database federation; integrates "Omics" data (e.g. genomics, proteomics, metabolomics); as well as solicit participation from organism biologists (e.g. taxonomy, ecology, biogeography).

This paper is organized as follows: in section 2 we present related work; in section 3 we describe the application domain; in section 4 we identify some challenges and present our approach for data integration; in section 5 we present the system architecture; in section 6 we present user evaluation, and in section 7 we discuss conclusions and future work.

2. Related work

We compare our work with traditional genome databases in the introduction and throughout this paper as a way to motivate and identify advancements with regard to current practices. In this section, we compare our work with other semantic Web applications focusing on queries as a way to facilitate access to information.

The tool Openlink iSPARQL [2] offers a graph-based interface for query formulation where the user can add classes to the visual graph as nodes and connect them with edges representing properties. The Semantic Discovery System (SDS) [3] also allows for complex semantic queries with a graph based approach, and offers additional methods to apply constraints. The user is presented with a graphical representation of the schema and can select properties and classes on which to focus their attention. In addition, the user can place constraints on relationships by selecting properties from a tree representation. In both cases, some familiarity with computer science is required. Other approaches use a tree representation of the data as a mechanism to construct queries. In GRQL [4] trees are used to construct complex queries of connected data. Trees can be useful because certain branches of interest can be expanded/collapsed as needed to show data of interest and hide irrelevant data. A problem arises when the data sets are very large, as is the case in the realm of bioinformatics. While a graph or tree-based construction mechanism allows for complex graph based queries similar to our system, it may seem alien to users or require extensive navigation to find concepts. This is because it forgoes the traditional search interface features such as typing keywords and selecting items from drop-down menus in lieu of its purely visual approach. TcruziKB aims at keeping common search

[2] http://www.adaptivepath.com/ideas/essays/archives/000385.php
[3] http://www.w3.org/TR/rdf-sparql-protocol

features such as keyword search and enhancing them with the addition of suggestions to help the user construct the query. In addition, we support queries involving complex types (e.g. collections), as well as user-provided instances (e.g. the amino-acid sequence "MFVAPKMAGF"). For queries that require the execution of bioinformatics tools as part of a query answer, we extend a SPARQL engine to support query-triggered web service executions, as explained in more details in section 5.

Other types of query systems such as, GINSENG [5] use natural language query processing in conjunction with ontologies to allow users to formulate questions in English language. Like TcruziKB, GINSENG provides suggestions as the user types and allows for graph based queries. The drawback of this type of system is that the terms in the system ontology may not lend themselves to natural language because factors such as synonyms, tense of verbs and noun forms come into play along with the style of writing of the particular user performing the query. For example, if the user wants to search for genes that are expressed in a particular life-cycle stage they may want to ask "Which genes are expressed in the epimastigote stage?" The problem is that there may be no property labeled "expressed". Instead there may be a property labeled "life-cycle stage". This may require the user to enter in a query that doesn't fit their particular writing style, such as, "Which genes have proteins that have been observed in the life-cycle stage epimastigote?" as opposed to the question they had originally intended to ask. These factors greatly complicate the query and mapping processes for both the system and the user. It might not always be possible to correctly map a question to a formal query if users ignore suggestions in favor of their terminology. In TcruziKB we avoid the ambiguities of natural language and provide a more controlled environment that guides the user step by step with suggestions of terms that are connected in the ontology. Therefore only meaningful queries (with respect to the ontologies adopted) will be formulated.

Another type of query interface, the form based query construction method, such as used in GoGet [6] requires the user to fill out a variety of information on HTML forms. The Display of many fields at once may alienate users because this type of query system is vastly different than the more traditional web searches they are accustomed to. GoGet also lacks portability, focusing itself on one set of data.

3. Application domain and domain model

From simple char-delimited (flat) files to complex relational database schemas, gigabytes of genome annotation data are available for use in the field of Genomics. We engineered an ontology to complement and interconnect information in this domain. We use controlled vocabularies and ontologies such as the Sequence Ontology (SO) [7], Gene Ontology (GO) [8], Pfam C [9] as well as newly created relationships to connect terms from these sources. For example, every protein in our system will be an instance of the class so:SO_0000104 (polypeptide) and every Pfam domain will be an instance of so:SO_0000417 (polypeptide_domain) from the Sequence Ontology. To connect instances of these two classes we created the property "has_pfam_domain".

We identified a manageable subset from the domain of *T.cruzi* "Omics" research to test the system and its underlying concepts. Our ontology schema is able to represent genes, as well as the organisms they belong to and the proteins that they encode. Protein family information from Pfam is present, and families are grouped in clans. Enzymatic function is linked to biochemical pathways, providing information about the biological processes they are involved in. Proteomic expression is also captured, including the information about the life-cycle stage(s) in which the protein is expressed, as well as quantitative measures of that expression. This knowledge base allows users to ask questions spanning multiple data sets such as:
(1) *"Which parasite genes are expressed during the human infective life-cycle stages?"* Although this is possible on TcruziDB, it currently assumes that the user knows which life-cycle stages are human and which are insect. In TcruziKB we have added this information by defining the property "present_in" to relate a life-cycle stage and the environment it naturally occurs.
(2) *"What are the relationships between a gene X and any gene that has the Pfam domain D?"* This type of question is exploratory in nature and may allow users to find out that such genes have common features such as sequence similarity, host, life-cycle stage, EC number, etc.

(3) *"What are the common characteristics of the life-cycle stages of Trypanosoma cruzi that express tubulin genes at the protein level?"* Suppose you find tubulins in amastigotes and promastigotes. Perhaps these stages occur in the same host? Although a biologist would know this, a computer scientist would not, hence, it could be "discovered". Other "discovery" questions that are useful for biologists could be similarly formulated.

4. Data integration

We obtain data from several sources, including Pfam flat files, Interpro XML and relational data stored in the Genome Unified Schema (http://www.gusdb.org/) for TcruziDB. Several challenges for data integration arise in this scenario, including format heterogeneity and loose vs tight coupling. In this section we explain how we deal with these challenges.

We use RDF [10] as the framework for data representation. In an optimistic scenario, data sources would expose their data sets on the Web already in RDF or through a programmatic interface that is able to produce RDF output. To emulate that scenario, we downloaded non-RDF

data sources and converted them to RDF files. We used BioJava tools (http://biojava.org) to read flat file and XML formats, and Jena Semantic Web Framework[4] to output RDF. Through that process, we imported data containing information such as: 31,630 protein domain (Pfam) annotations and 8,065 ortholog groups predicted by the OrthoMCL algorithm. To acquire TcruziDB data, we automatically mapped the GUS Schema to RDF using the D2RQ mapping framework [11]. A manual step was performed to identify terms that already existed in ontologies, and thus promote reuse. This step was greatly facilitated by the fact that the GUS schema was modeled to reflect the domain of "Omics" research, much like ontologies are usually built. For an example mapping from the database to ontologies, consider the example of the table DoTS.AminoAcidSequence, which was mapped to the concept "polypeptide" (SO:0000104) in SO. This mapping will generate an instance of SO:0000104 for each row in Dots.AminoAcidSequence. The reference to an accepted standard such as SO will facilitate the access to proteins by other systems that also subscribe to SO.

The subset of the TcruziDB dataset used in this project was extracted from Release 5.1 (Nov 07, 2006) and includes: 19,613 automated gene predictions (protein coding); 139,147 protein expression results from metacyclic trypomastigotes (CL strain) and amastigotes, trypomastigotes and epimastigotes (Brazil strain) of *T. cruzi*. The data set already contained links to the sequence ontology, gene ontology and enzyme commission numbers at the row level. That is, a given record in a table has some relationship with a given ontology term. However, the type of relationship is not explicit. As part of the relational to RDF mapping, we use existing and newly created relationships to capture the semantics of the association between records in the database, thus providing a richer description of the dataset. We then expose all imported sources on the Web through programmatically accessible interfaces to emulate a possible future case where data providers start to adopt RDF as data exchange format. From Web-accessible data sources, we can perform loose-coupled integration of data, in contrast with the tight-coupled approach commonly observed in genome databases. The term coupling in this case refers to the ability of one to attach or detach data sources to the system. Current genome databases adopt a "warehousing" approach, where data is downloaded from several external sources, transformed and stored in a local centralized source. In our system, the user is able to plug-and-play data sources at query time. This functionality is enabled by the SPARQL Protocol for communication with external sources and RDF representation for merging results.

[4] http://jena.sourceforge.net/

5. System architecture and implementation

TcruziKB is composed by a Web interface on the client side and communicates through the SPARQL Protocol with the server side (SPARQL Endpoint) over the Web. In order to use this Web interface, the user just needs a common Web browser with Javascript support (tested on Firefox 2.0.0.14). Queries are built through a friendly interface in the query formulation module, translated to SPARQL in the SPARQL-JS module and sent to the server side via asynchronous background calls (Ajax) for execution. Once returned to the client side, the controller module notifies the visualization modules subscribed, providing the results to be displayed to the user. We also extend a SPARQL Endpoint implementation to support Web services executions as a way to obtain parts of query results. Each component is explained in more detail in the following sections.

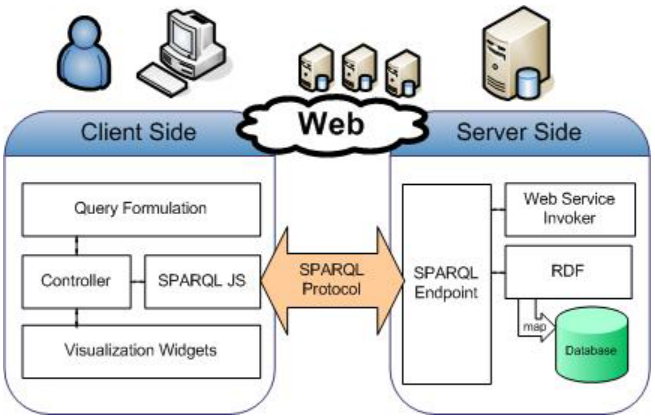

Figure 1 – System architecture

5.1. Visual query formulation

The key enabler of the visual query builder is the ontology schema. We read from the schema the types of data and possible interconnections to guide the user in creating a query. Therefore, the user interface enables the user to perform queries involving any terms in the ontology schema (and thus all parts of a database schema that are mapped to the ontology). Please refer to Figure 2 for an illustrative user interaction between Joe (fictional) and our system.

After starting the Web browser and pointing it to TcruziKB's URL, the user interaction begins with a search for a term to start the query construction. In Step 1, Joe types in "gene." Based on the ontology schema definitions, the system will suggest terms such as "gene", "gene_identifier", and "gene_prediction_task" as possible candidates. Upon selection of the first term "gene", Joe can indicate that he is interested in "any gene" by setting the placeholder "?gene" or he may choose to constrain the search to a specific set of genes. In Step 2, the system will suggest possible relationships associated with "gene".

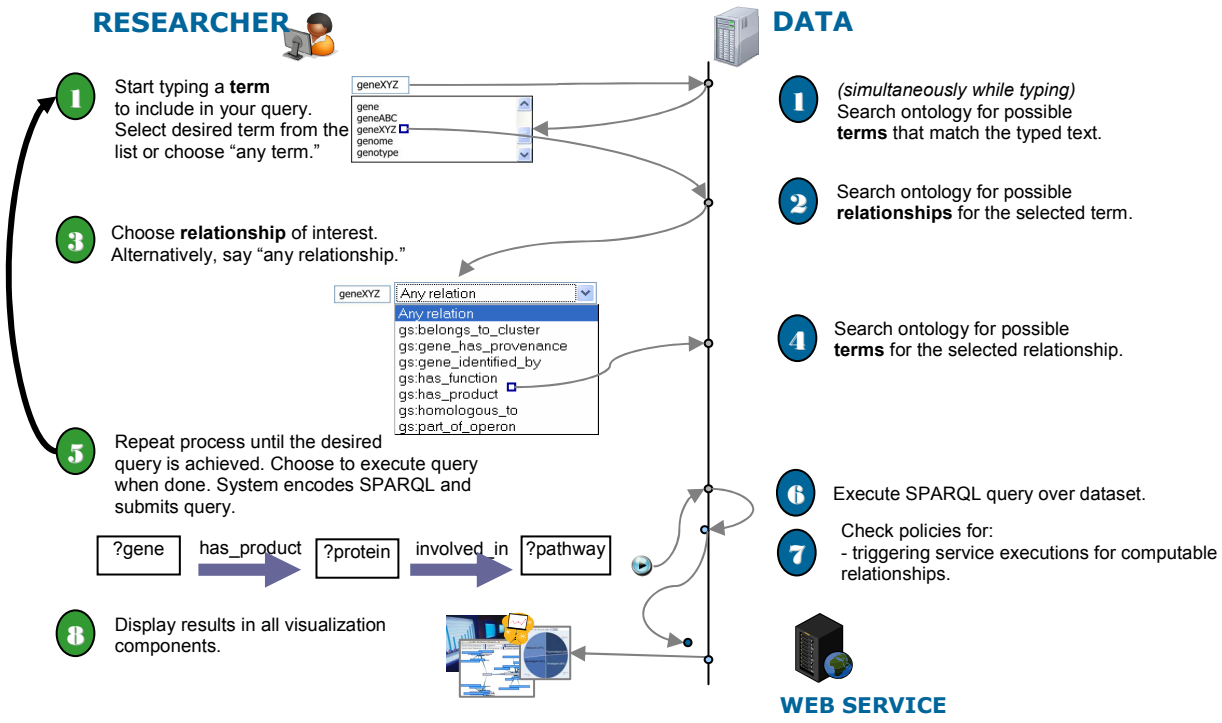

Figure 2 – Example user interaction through the query formulation interface

Examples of relationships could be "has_product", "has_function" and "homologous_to" or others, which will be displayed through the user interface through a drop-down box. In Step 3, Joe will select "has_product", and then the system will suggest terms possibly associated with "gene" through relationship "has_product" in Step 4. Here Joe will select "protein" from the suggested list. This process continues until Joe has finalized the desired query pattern.

The queries composed by the visual query builder are directed graph patterns to be searched in a knowledge base. A variable in a graph pattern is represented by a question mark prefixing a variable name – e.g. ?protein. Variables represent gaps to be filled by classes, instances or properties. This is a major difference with respect to standard genome databases, where no query can be asked about "relationships" or "classes", since those are not explicitly represented in the database. An example of a query involving a relationship would be *"what are all the relationships between gene A and gene B."* An example of a query involving a class would be "what are the types of proteins expressed in a given life cycle stage."

We also support queries involving collections: *"Find all enzymes with orthologs in genomes G1, G2, …, Gn."* This query requires the formulation of a collection (or set) of genomes as part of the query pattern. Patterns containing collections are represented internally as follows "enzyme → common_to → (genome1, genome2, genome3)." A utility tool to test the membership in a given collection is provided at the query engine.

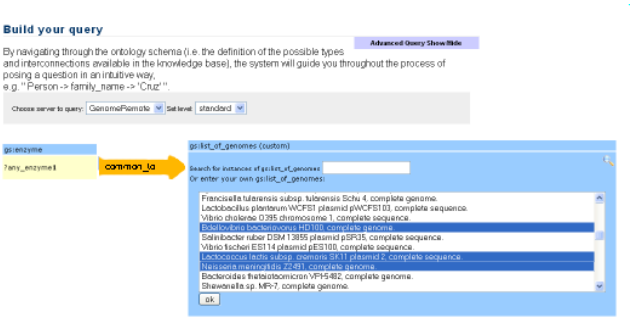

Figure 3 – Support for structured input in the query formulation interface.

An important feature of the query builder is its ability to guide the user through a directed graph pattern from any standpoint, in any direction desired. For example, a user is able to start in a "Protein" and find any "(?protein, has_expression, ?proteinExpression)", as well as start in "ProteinExpression" and find any "(?proteinExpression, is_expression_of, ?protein)." We anticipate that not all data sources will explicitly state the inverse of every property, so the query builder implements the ability to navigate any relationship in both directions. As a matter of fact, we anticipate that some data sources will not present an ontology schema of any kind. In that case, the visual query

builder would lack the information to predict the next possible elements to be added to the query. However, for cases where the schema is not present, but the metadata is – i.e. there are no domain and range descriptions, but the type of the instances is known – we can infer domain and range by inspecting all property instances and the types of their subjects and objects.

After the user has built the desired graph pattern to be searched, the visual query builder proactively enhances the query by adding triples to retrieve optional extra information about the results. The enhancements retrieved include annotation properties such as label (rdfs:label), type (rdf:type) and a web page link for each resource (rdfs:seeAlso). As an example, for the class so:SO_0000104, the property rdfs:label would contain "polypeptide" and rdf:type would contain so:SO_0000001 (region). These additions are valuable in analytical interfaces to facilitate the understanding of the information presented.

5.2. Query execution

We support queries to multiple SPARQL endpoints by storing a list of servers and performing calls to all of them each time a query is executed. The results are asynchronously received back from the SPARQL endpoints and aggregated in a result set for presentation to the user. The addition and configuration of new SPARQL endpoints is supported through our user interface. As a consequence, researchers using our system can "plug-and-play" new data sources without any development intervention.

In Genomics, the need to combine results from database searches as well as tools executions to answer a given query is commonly observed. Consider, for example, the case where the user wants to retrieve all genes that encode proteins similar to a sequence provided by the user, e.g. "MRGVSAAEIGKFR". Protein similarity cannot be computed with simple string comparison, but in fact requires more sophisticated algorithms such as the one implemented by Blast (http://www.ncbi.nlm.nih.gov/blast/).

In order to support queries that require combinations of data lookups and web-service executions, we extend ARQ[5] the SPARQL interpretation and execution module from Jena Semantic Web Framework. ARQ offers property functions as a way to extend the basic pattern matching performed by a common SPARQL query engine. A typical SPARQL query engine builds a candidate solution set by matching the query pattern to the RDF triples in the target database. ARQ's property functions are executed when the SPARQL query engine is about to test the presence of a given relationship, in case this relationship is registered with the engine as a property function. We introduce special property functions that are able to execute Web services to compute parts of query results. Once a service is executed and the results are computed, the query engine then binds the results to the query solution and the query execution resumes. The results obtained are also saved to the knowledge base so the same process does not have to be performed again in the future.

[5] http://jena.sourceforge.net/ARQ/

5.3. Exploration of results

We provide a component for visualization of query results to allow for researchers to perform data exploration in different presentation formats, including tables, graphs or summarization charts to suit different analytical needs. We package each component as a Javascript module (widget) that can be easily plugged into other web-based SPARQL query systems.

5.3.1. Tabular Explorer

The tabular explorer displays a result set in the form of a spreadsheet (rows and columns), with each row representing a solution to the query. The columns represent variables that can assume as values a set of instances, properties or classes that match the query executed. The interface allows the user to further filter or reorder the results in the table, providing extra exploration functionality.

5.3.2. Pie-chart Explorer

To allow for an overview of a result set, we created the pie-chart explorer. It displays a summary for each result set. Consider a query for all instances of the class ProteinExpression with respect to *T. cruzi*. This class has a property "life_cycle_stage" to indicate in what stage of *T.cruzi* a given protein was expressed. An intuitive way to summarize the results is to analyze the proportion of instances that were expressed in each stage (Figure 4).

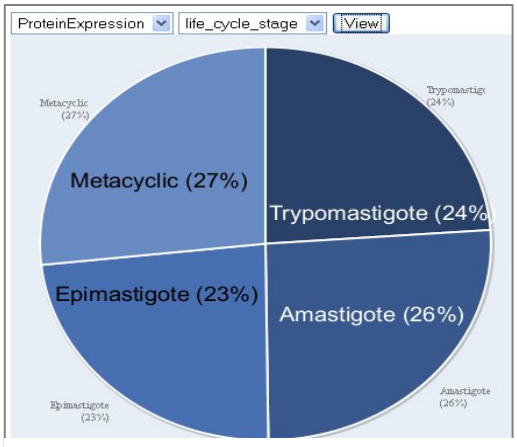

Figure 4 - The pie-chart explorer showing the percentage of expression results for *Tcruzi* by life cycle stage).

In general terms, the pie-chart explorer is built as follows. For each instance returned in a query, the system obtains the ontology class. For each class-property pair, the

system retrieves the possible values of the property. For instance, for ProteinExpression and life_cycle_stage, the possible values are metacyclic, amastigote, epimastigote and trypomastigote. For each of those class-property pairs, the system presents a chart showing the proportion of instances that assume each of the values in the result set. For instance, for a query for all protein expression results, the system would present one pie chart for each property of the class ProteinExpression (e.g. life_cycle_stage, ortholog_group, etc.), reflecting the distribution of values for those properties (e.g. 23% have the value "amastigote" for the property "life_cycle_stage").

5.3.3. Graph Explorer

The nature of ontologies lends itself naturally to a directed graph representation. The query results can be organized on a graph with classes/instances corresponding to nodes and properties corresponding to edges. By spotting clusters or paths between classes, researchers can gain additional insight on the data. The graph explorer allows the extension of the results with additional classes and properties through mouse commands. For a visual example refer to Figure 5.

We recognize that graphs can get easily encumbered with too much information. The graph explorer offers the option to arbitrarily collapse or expand edges of a node according to user commands. In addition, techniques to automatically rank important relationships [12] can be applied and displayed in this component.

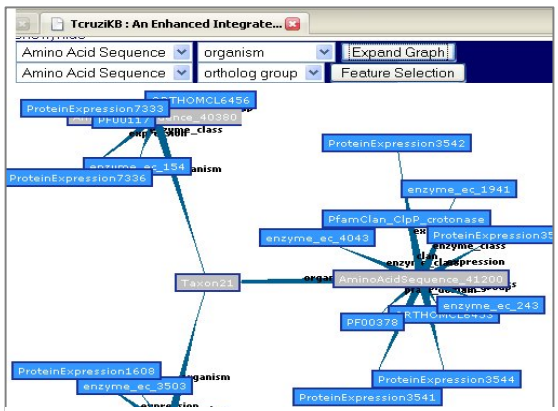

Figure 5 – The graph explorer

6. Evaluation

We evaluated TcruziKB both subjectively and objectively with regard to its parent system, TcruziDB. A panel of 30 university members including professors, graduate students, and undergraduates were asked to perform searches using TcruziKB and TcruziDB and record their experience. For the subjective evaluation, the System Usability Scale (SUS) [13] was used as the benchmark. SUS is a 10 question form that rates the usability of the system from 0, very user unfriendly, to 100, highly user friendly. The average SUS scores for each system can then be used to compare the usability of the systems.

After using the system for several minutes to answer sample queries, the panel members scored their results on the SUS forms. The SUS averages show that the usability of TcruziKB is very similar to TcruziDB. TcruziKB received an average SUS score of 90.54 while TcruziDB scored 88.11. This implies that even though TcruziKB incorporates many advanced features it does not sacrifice usability, in fact it became slightly more usable than its parent system. The scores are broken down by area expertise in Table 1.

Background Area	#users	TcruziKB Score	TcruziDB Score
Overall	30	90.54	88.11
Computer Science	20	95	84.77
Biology	10	91.43	90.03
CS and Biology	5	100	97.61

Table 1: SUS Scores for each system broken down by area of expertise.

For the objective evaluation, five members of the panel were asked to record the time taken and the number of computer interactions (the number of mouse clicks and keystrokes) needed to perform a query on each system. For benchmarking, the following queries, of variable complexity, were given to the panel:

"Which genes have protein expression during a parasite life-cycle stage that is in the human host?" This query can be executed as a single path query on TcruziKB of the form: "Gene → codes for → Protein → expressed in → Life Cycle Stage → resident in → Human Body." On TcruziDB, the query is less straightforward to execute because it requires the user have knowledge of the parasite, specifically which life cycle stages are relevant to the search.

"What are the relationships between gene Tc00.1047053409117.20 and any gene that has protein family PF03645?" This query will require the user to search for the gene by ID, search for PFAM by ID (or name) and then intersect the results through the query history. In TcruziKB the user obtains the same results without browsing pages or issuing separate queries.

"Which genes encode proteins that are expressed in both the epimastigote and trypomastigote life-cycle stages?" With TcruziDB the user will need to view all genes that are in proteins expressed in the epimastigote stage and then the trypomastigote stage and then "Combine the Results." This three step process on TcruziDB can be done from a single query in our system. "Gene → codes for → Protein → expressed in → epimastigote AND Gene → codes for → Protein → expressed in → trypomastigote."

While both TcruziKB and TcruziDB were similar in terms of usability TcruziKB had a significant edge in time taken and the number interactions needed to obtain the query results. 117.33s TcruziKB and 211.33 s TcruziDB,. This is because backtracking is needed to construct complex queries on TcruziDB. The user actually conducts several queries to the system then must combine the results of each. In TcruziKB complex queries can be constructed all at once, eliminating the backtracking and thus reducing the time needed. The need for backtracking in TcruziDB is also reflected in the number of interactions needed requiring an average of 53.33 interactions as opposed to an average of 21.33 interactions in TcruziKB. This implies that time needed to do the backtracking is not only unnecessary computer processing time but is also time spent by the user doing unnecessary work.

7. Conclusion

We present an application of ontology-based information aggregation, querying and exploration in the context of *Trypanosoma cruzi* Genomics. The system is available at: http://knoesis.wright.edu/tools/tcruzikb

We constructed a query builder capable of composing complex queries through the navigation of ontology schemas. This approach enables complex queries that were only possible in traditional genome databases through multiple executions of simple queries and subsequent combination of results. User-provided addition and querying of new data sources is supported in a plug-and-play fashion.

Complex queries that require Web services executions to obtain parts of query results are supported through an extension to a SPARQL Endpoint implementation. As part of such queries, services are invoked and the results obtained are merged to the result set and returned to the user for presentation. The presentation and exploration of results in multiple interfaces is also investigated, helping to highlight for the researcher a manageable subset of interest from an extensive mass of information.

We expect the above mentioned contributions to compose a valuable toolkit for data sharing and analysis on the Web that can be reused and extended for any domain for which ontologies exist. We are extending both the query formulation software (Cuebee[6]) and the components for results exploration (Exparql[7]) for application to other domains of knowledge besides Tcruzi “Omics”.

As future work for TcruziKB, we envision the expansion of the dataset, support for path queries [14], and subgraph discovery queries [15].

[6] http://knoesis.wright.edu/library/tools/cuebee/
[7] http://knoesis.wright.edu/library/tools/exparql/

8. Acknowledgements

Pablo Mendes was supported in part through an American Heart Association award 0330338N to J.C.K. This research was also supported in part by the NIH-NHLBI funded "Semantics and Services enabled Problem Solving Environment for T.cruzi" (1R01HL087795-01A1) project. Thanks to Maciej Janik and Matthew Eavenson (Cuadro project), Sena Arpinar and Ying Xu (collaborators at IOB), members of the J.C.K. Laboratory and the TcruziDB team (for facilitating access to data and providing valuable advice).

9. References

1. Agüero, F., et al., *TcruziDB: an integrated, post-genomics community resource for Trypanosoma cruzi.* Nucleic Acids Res, 2006. **34**(Database issue).
2. *Openlink iSPARQL*. [cited; Available from: http://demo.openlinksw.com/isparql/.
3. *Semantic Discovery System*. [cited; Available from: http://www.insilicodiscovery.com/.
4. Athanasis, N., V. Christophides, and D. Kotzinos, *Generating On the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL)*, in *The Semantic Web â€“ ISWC 2004*. 2004. p. 486-501.
5. Bernstein, A., et al., *Querying Ontologies: A Controlled English Interface for End-Users*, in *4th International Semantic Web Conference (ISWC 2005)*. 2005. p. 112-126.
6. Shoop, E., et al., *Data exploration tools for the Gene Ontology database.* Bioinformatics. **20**(18): p. 3442.
7. Eilbeck, K., et al., *The Sequence Ontology: a tool for the unification of genome annotations.* Genome Biol, 2005. **6**(5).
8. Ashburner, M., et al., *Gene ontology: tool for the unification of biology. The Gene Ontology Consortium.* Nat Genet, 2000. **25**(1): p. 25-29.
9. Finn, R.D., et al., *Pfam: clans, web tools and services.* Nucleic Acids Res, 2006. **34**(Database issue).
10. Lassila, O. and R.R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. 1999 [cited; Available from: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.
11. Bizer, C. and A. Seaborne, *D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs*, in *3rd International Semantic Web Conference (ISWC2004)*. 2004: Hiroshima, Japan.
12. Aleman-Meza, B., et al., *Ranking Complex Relationships on the Semantic Web*, in *IEEE Internet Computing*. 2005. p. pp. 37-44.
13. Brooke, J., *SUS - A quick and dirty usability scale.*
14. Anyanwu, K. and A. Sheth. *ρ-Queries: enabling querying for semantic associations on the semantic web.* in *WWW '03: Proceedings of the 12th international conference on World Wide Web*. 2003: ACM.
15. Ramakrishnan, C., et al., *Discovering informative connection subgraphs in multi-relational graphs.* SIGKDD Explor. Newsl., 2005. **7**(2): p. 56-63.