A Visual Web Query System for NeuronBank Ontology

Weiling Li
Georgia State University
Atlanta, GA
wli16@student.gsu.edu

Rajshekhar Sunderraman
Georgia State University
Atlanta, GA
raj@cs.gsu.edu

Paul Katz
Georgia State University
Atlanta, GA
pkatz@gsu.edu

ABSTRACT
Ontologies have recently been heavily developed in the life sciences. Scientists retrieve life science ontologies and references to them (e.g. annotations) for aiding their research. As users usually have little or no knowledge about the query languages such as Algernon or SPARQL to access these ontologies, designing flexible and powerful Web query interfaces will make this information more accessible. In this paper we describe a Web query system for NeuronBank[1], a web-based tool for cataloging, searching, and analyzing neuronal circuitry within and across species. Information from a single species is represented in an individual branch of NeuronBank. We aim to assist users in searching within a branch or perform queries across branches to look for similarities in neuronal circuits across species. The query interface ideas are general enough that they can be adapted to work with other ontologies.

Author Keywords
Ontologies, Visual Query Interfaces, Form-based Querying

INTRODUCTION
During the last decades, many ontology-based knowledge base systems have been developed in order to meet the challenges of complex and quickly evolving life science data. Briefly, an ontology is a specification of a conceptualization ([9]). The design purpose of an ontology is to allow researchers to share and reuse knowledge bases. To easily access the information located in these ontologies, several query languages and systems have been proposed, such as SPARQL ([18]), Algernon ([1]), COQL ([6]), and CIRI ([5]). The Algernon rule-based inference system, which is implemented in Java, performs forward and backward rule-based processing of frame-based knowledge bases, and efficiently stores and retrieves information in ontologies and knowledge bases.

In this paper, we introduce an ontology-based Web query interface that is built on the Algernon system. The visual query interface enables end-users to query the knowledgebase without knowing any query language. It is the major web-based query component of the NeuronBank system ([15]), an online reference source and informatics tool for exploring vast knowledge of identified neurons and the circuits they form. The query system is developed using the JavaServer Faces (JSF) technology. Through the readily usable query interface that dynamically retrieves the ontology, the user can specify their query criteria, and create a form-based query. The form based design relieves end users from the low level knowledgebase details and the terse and error prone text-based query expressions. Moreover, it is unrealistic to expect naive users who have no desire to work in computer science to become proficient in a query language. The web-based design makes the query system accessible from any computer on the internet.

The form-based query is translated into a textual Algernon query that is then executed against the ontology-based knowledgebase. The query result includes not only the main results but also all intermediate results. With the intermediate results, users can easily trace the inference chain performed on the data in the knowledgebase. This query system also supports user-defined operators that are derived from the functions and relationships defined in the ontology.

The remainder of this paper is organized as follows: The second section sketches the architecture of the NeuronBank system. The third section presents details of the NeuronBank visual query system. The fourth section discusses related work and the advantages of our system compared to other visual query systems. Conclusion and future work are presented in the final section.

BASIC ARCHITECUTRE OF NEURONBANK
Figure 1 (obtained from [12]) shows the overall distributed architecture of the NeuronBank system. To manage information related to different species with different data models and user interfaces, knowledge about each species is designed as a branch. Each branch of NeuronBank hosts a knowledge-base, called *BranckKB*, that contains knowledge about the neurons and synapses of a particular species. The knowledge base is stored within the Protégé Ontology Editor and Knowledge-Base System ([8]). In addition to hosting the knowledge base, the BranchKB also provides a Web service interface to the rest of the system, thereby enabling sharing of the knowledge in a distributed manner. Various

[1] http://www.neuronbank.org

Workshop on Visual Interfaces to the Social and Semantic Web (VISSW2011), Co-located with ACM IUI 2011, Feb 13, 2011, Palo Alto, US.

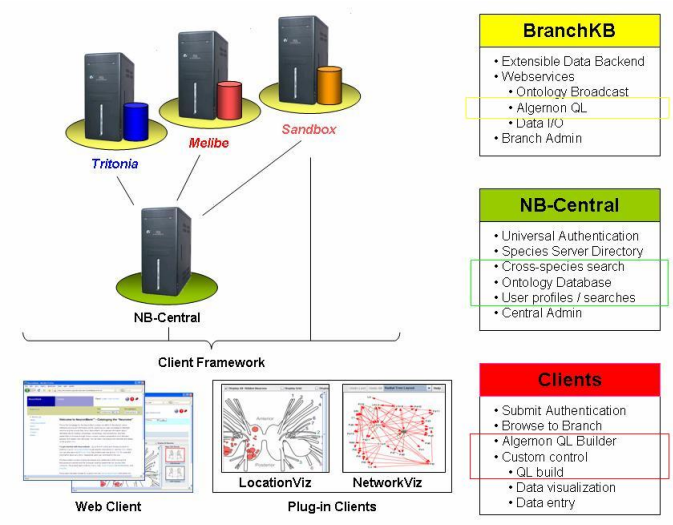

Figure 1. Architeture of NeuronBank

administrative functions such as creating a branch, importing and exporting the branch ontology, and updating branch parameters are also supported by the BranchKBs.

The different branches of the system are united by a common framework, *NB-Central*: NeuronBank Central, which enables search and analysis of neural circuits across all species. This architecture allows for the sharing of knowledge from different species. NB-Central not only facilitates searching and analysis of data across all the species represented, but also provides Web services to the BranchKBs and the Clients. NB-Central works with BranchKBs to provide "cross-branch search", accepting queries from Clients and passing them on to each available BranchKB. NB-Central then collates the results and returns them back to the Clients.

Clients are individual applications, either stand alone or Web-based, that need to access the various BranchKBs. The clients can be anonymous or can register themselves with *NB-Central* and can create a form-based query, submit search criteria, and store search results, and use other services.

QUERY SYSTEM OVERVIEW

One of the important components of NeuronBank is the advanced search sub-system that is implemented as a client-server Web application. The client-side provides two interfaces: *query generation user interface* and *query result display user interface*. The server-side consists of several functional components such as ontology schema retriever, text Algernon query generator, Algernon engine, and user-defined operator (see Figure 2).

Through the query generation user interface, the system provides the end-users with a dynamically view of the ontology schema and allows them to build their form-based Algernon query expressions. The query results are displayed on the summary page of the query result display user interface. The query results include not only the final query objects,

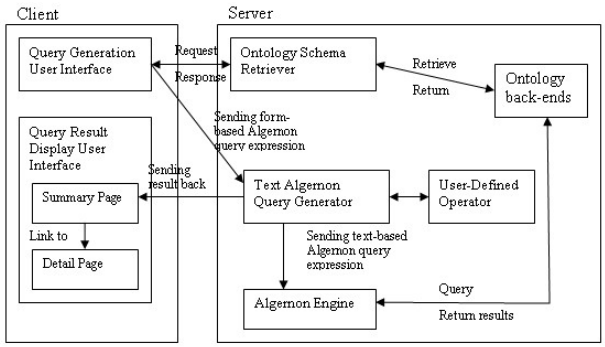

Figure 2. Query System Architecture

but also all intermediate objects specified in the form-based Algernon query expression. As a result, users can easily keep track of the chain of inference rules performed on data in the knowledge base. Within the summary page, all objects including intermediate objects are bound to hyperlinks, through which users can visit the detail pages of the corresponding objects. The detail page for a particular object shows a comprehensive view of the object including the values of all of its attributes, relationships with other objects in the system, as well as two graphical views (location and network) implemented as Java applets.

On the server side, ontology schema retriever can dynamically retrieve ontology schema elements from the ontology backend and provide them to the client applications. The text Algernon query generator takes as input the form-based Algernon query expression built by end-users and generates the text-based Algernon query statement. User-defined operators are based on the functions predefined in the ontology, and facilitate users to create a query expression close to their domain research questions. The Algernon engine executes the text Algernon query against the knowledgebase, and returns the result to be displayed by the summary and detail pages.

Each of these interfaces is described in more detail now.

Query Generation User Interface

The query generation user interface is a general purpose interface that facilitates the expression of ad-hoc queries against a given ontology. The ad-hoc queries usually represent a "path expression" in the underlying graph structure (data graph) of the objects (nodes) and relationships (edges) of the ontology. The interface allows the user to traverse a desired "path" in the underlying data graph and retrieve objects of interest along the way. Using this interface, users can formulate very complex queries without knowing much of the underlying structure of the ontology.

Query generation user interface is a form-based user interface that runs in a Web browser environment. The interface consists of multiple columns, each consisting of three parts:

1. a class list consisting of a dropdown menu of classes from

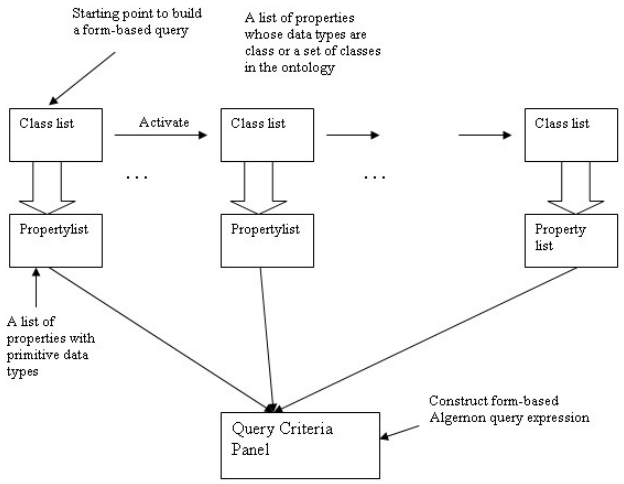

Figure 3. Query Generation User Interface

the underlying ontology,

2. a property list consisting of a listbox of primitive properties of the class chosen, and

3. a query criteria panel that allows users to express selection conditions on the chosen property.

Figure 3 illustrates these three components.

In NeuronBank, the ontology is stored within the Protege frames storage system and is interpreted as a set of "classes" with each class containing two types of "property assertions": primitive properties that have simple values and relationship properties that relate the object to other classes in the ontology. The multiple columns of the interface along with their constituent parts are displayed by the system in a piece-meal fashion responding to the user's interactions with the interface.

The first part of query generation user interface shows information including classes and relationships to the users to enable them to express their queries as follows:

1. The left-most column of the interface consists of a class list (a dropdown menu) that is the start point for building the form-based query. This class list is initialized with the names of ALL classes available in the ontology when the interface is loaded. This start dropdown menu lists all classes defined in the ontology in a hierarchical style that represents the inheritance relationships among them. An example is shown in Figure 4. In this example, class *Chemical_Synapse* is the subclass of class *Connections*, which is a root class.

2. After users select a class in the start dropdown menu, two GUI components on the page are activated:

   (a) The property list-box beneath the selected dropdown menu class is populated with the primitive properties of the selected class.

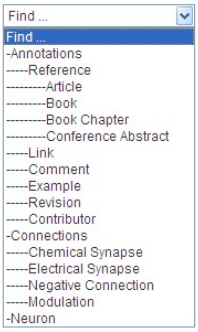

Figure 4. The Start Dropdown Menu

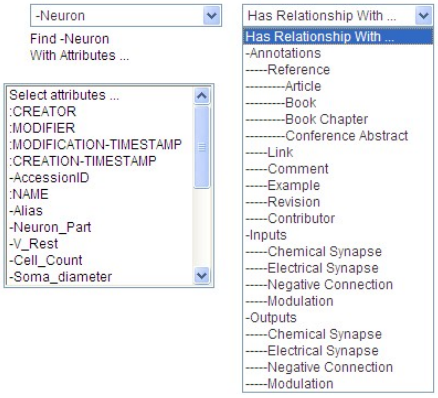

Figure 5. Relationship Properties of Selected Class in next Column

   (b) The properties of selected class that reference other classes in the ontology are imported into the dropdown menu to the right of the start dropdown menu in the next column of the interface.

   Similarly, the selection of a class in the second dropdown menu will activate and populate the third dropdown menu and the property list-box under it. All ontology schema information above is dynamically retrieved by ontology schema retriever module. For example in Figure 5, the selected class *Neuron* in the left menu has three properties, *Annotations*, *Inputs*, and *Outputs*, which are containers of compatible classes. Note that we can express the fact that *Neuron* has a relationship with not only one of these three properties, such as *Inputs*; but also its subclasses, such as *Electrical_Synapse*, and so on. This part of conceptualization defined in the ontology is dynamically retrieved and displayed in the interface for users to build their queries.

The second part of the interface is a group of property list-boxes, each of which corresponds to a class chosen by the user in the dropdown menu above. By choosing a class from the dropdown class list, its primitive properties are displayed in the property list-box under it. An example is shown in Figure 6. All primitive properties of the selected "$-Neuron$" class are listed beneath it. The property with ":" sign in the front signifies that this property is an intrinsic property, i.e. system-generated. Property with "$-$" sign in the front signi-

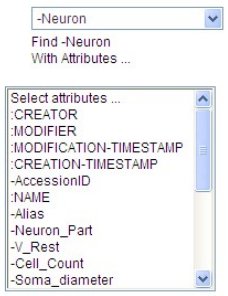

Figure 6. Primitive Properties of Selected Class - in Property List-Box

fies that this property is a user-defined property. All classes in the NeuronBank ontology have certain fixed core properties and a varibale number of user-defined properties.

The third part of the interface is the query criteria panel. This part is a form-based interface that allows users to specify selection conditions on the primitive properties chosen by the user from the property list-box above. Users can construct Algernon query criteria to limit the objects of interest in the current column of the interface. The aim is to build the search interface dynamically to match the current data model of a selected branch. User can simply select one or more properties from the property list-boxes, and click *Add* button to add them into the form-based query. Then, they can specify the selection conditions in the query criteria section of the interface. All the conditions expressed in the query criteria section are treated as a conjunction ("and").

The query generation user interface descibed above is a general purpose interface that provides the user the capability to formulate very complex query requests. One such query request is discussed in the example below.

An example query formulation. Consider the following query request: Find all neurons which are involved in chemical synapses satisfying the following two properties:

1. the connection probability of the synapse is greater than 2, and
2. the synapse has an article annotation which was published after year 2000.

Using the query generation user interface the user has expressed the query as shown in shown in Figure 7. The user is looking for

a a neuron

b with a chemical synapse (whose parent is "$Inputs$" hidden in the figure)

c has an attribute of "-connection_probability"

d and the value of that attribute is larger than 2

e where an article (whose parent is "$Annotations$" hidden in the figure)

f has an attribute of "-Year"

g and the value of that attribute is larger than 2000

In this example the user has added two query criteria: *chemical synapse involved with the connection probability* > 2 and *article involved with year* > 2000.

h When the "*Add*" button is clicked a table-like panel is created, which allows the user to express the selection conditions in the query criteria.

Finally, this query can be submitted to

f the current branch (e.g. *Tritonia*) or

g all branches.

The table-like panel for the query criteria has the following six columns (again refer to Figure 7):

1. *Class Name* column representing the class of a query criterion. (e.g. *-Inputs.-Chemical_Synapse* means a chemical synapse input).
2. *Attribute* column representing a property of the class. (e.g. *-Connection_Probability* is a property of domain *-Inputs.-Chemical_Synapse*).
3. *Operator* column containing a list of operators that are dynamically determined and generated according to the property data type. That is, it has only compatible operators depending on the property. This column also also supports user-defined operators that are predefined by domain experts.
4. *Value* column representing a constant value from the property domain. By default the field is supplied with a string to represent input value constraint, meaning the input value must follow the constraint of the property and only the matching value can be input to this field. By using the default String, user will know the type of the property. For example, the constant value for *-Connection_Probability* only can be entered in a numeric format. The constraint for each property is defined in the ontology.
5. *Definitional* column is for the definitional function defined in the NeuronBank ontology. Properties that are tagged by the ontologist as being important in identifying the cell are called definitional.
6. *Remove* column containing a remove button for removing a row in the panel.

Users can easily edit their queries by clicking the *Add*, *Reset* or *Remove* button. By clicking *Submit* button, queries are submitted to the current branch and the *result user interface* is displayed by the system. By clicking *Submit_to_All_Branches* button, cross branch query results are shown as in Figure 8. Then clicking one of branches, the corresponding *result user interface* is displayed by the system.

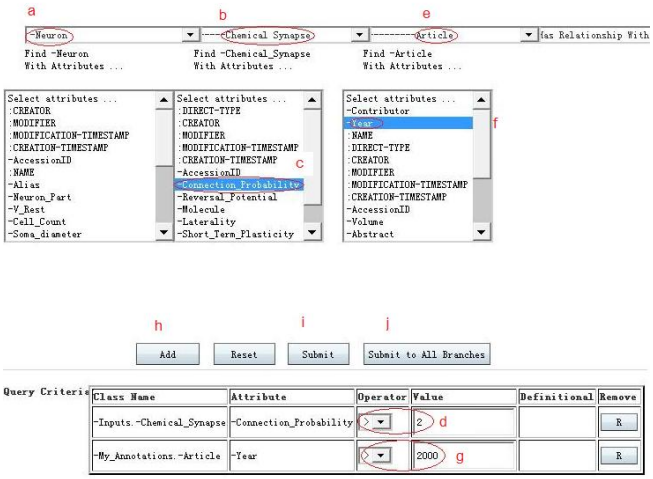

Figure 7. An Example Query Formulation

Results from

Branch	Query Status	No. of Results
Tritonia	Query Complete	3
SandBox	No results/Branch Unavailable	-
Melibe	No results/Branch Unavailable	-

Figure 8. Cross Branch Query Results

Result Display User Interface

The *result display user interface* is composed of two pages: Summary Page and Detail Page. The summary page has a query result table, whose schema matches the dropdown menus in the query formulation. That is, the first column of the table will match the selected class in the start dropdown menu, the second column matches the class selected in the second dropdown menu, and so on. The first column is the major domain and corresponds to the objects of interest in the query. Other columns are the related or intermediate domains that are used in the query criteria to retrieve the major domain result. Each row in the table consists of one instance from the major domain and a list of intermediate instances from the related domains. The list of intermediate instances is used to infer the major instance. The *Sorting* button within the table can be used to sort the result list. Every instance in a result row is bound to a link that takes the user to the detail page containing for the corresponding instance (see Figure 9).

The results within a branch are displayed in three formats: *List*, *Location*, and *Network*. Here we focus on the *List* format which presents objects instances that satisfy the query. Figure 9 shows the results table for the query of Figure 7. *-Neuron* is the major class, *-Inputs.-Chemical_Synapse* and *-My_Annotations.-Article* are the related classes. Each row in the table represents a chain of inference rules performed on the data in the knowledgebase. One instance may be retrieved from multiple inference-rule chains, such as the instance *S-Cell* (one neuron) in above table. This design provides users the maximum information related to their major query result. The detail information about each instance can

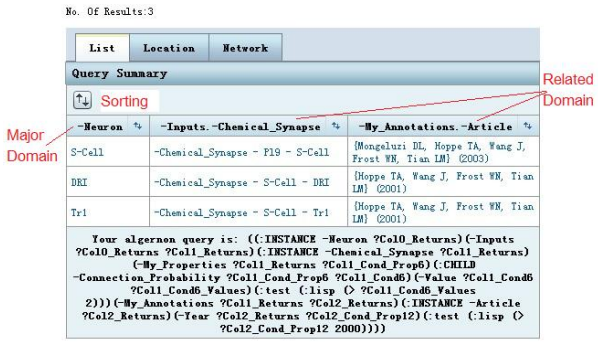

Figure 9. Summary Page

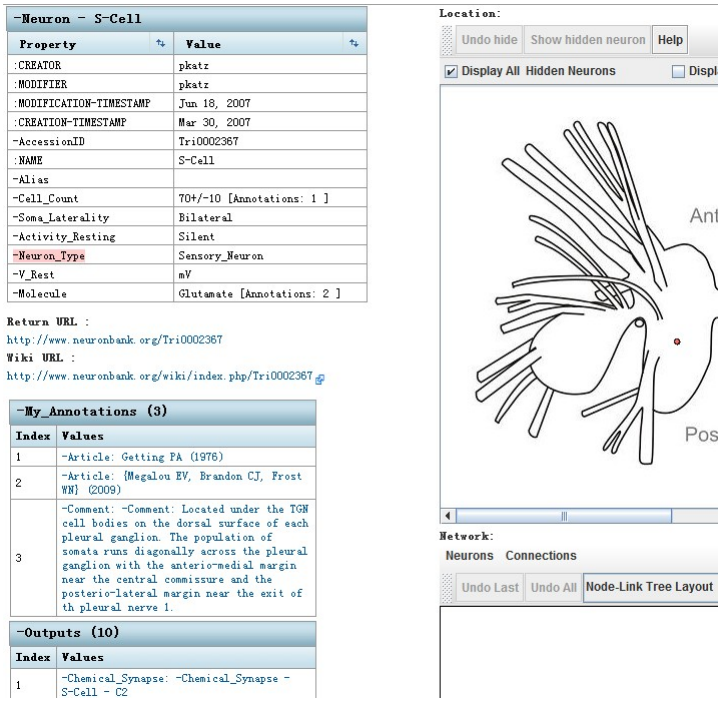

Figure 10. Detail Page

be reached by clicking the instance (i.e. a hyperlink) inside the table. A sample detail page for neuron *S-Cell* is shown in Figure 10.

Algernon Query Generation

A key aspect of the advanced query interface is the automatic generation of the Algernon query. As the query is constructed by the user using the visual interface, at each step the system is automatically creating and appending various components of the final query string. The query generation process is illustrated by looking at the example of Figure 9. Here, the user has created a query to retrieve neurons with a chemical synapse which has an attribute of *connection probability* which is *larger than 2* and which has an article annotation has an attribute of *year after 2000*. Corresponding to each of the three dropdown menus of classes in the visual query constructed by the user, there are three columns in the Algernon query expression:

- instance *Neuron*,
- *Inputs*, whose instance is *Chemical_Synapse* with the value of *Connection_Probability* property (whose parent class is *My_Properties*) *larger than 2*, and
- *My_Annotations*, whose instance is *Aritcle* with the value of *Year* property *larger than 2000*.

The final Algernon query expression is as follows:

```
(
(:INSTANCE -Neuron ?Col0_Returns)

(-Inputs ?Col0_Returns ?Col1_Returns)
(:INSTANCE -Chemical_Synapse ?Col1_Returns)
(-My_Properties ?Col1_Returns ?Col1_Cond_Prop6)
(:CHILD -Connection_Probability
    ?Col1_Cond_Prop6 ?Col1_Cond6)
(-Value ?Col1_Cond6 ?Col1_Cond6_Values)
(:test (:lisp (> ?Col1_Cond6_Values 2)))

(-My_Annotations ?Col1_Returns ?Col2_Returns)
(:INSTANCE -Article ?Col2_Returns)
(-Year ?Col2_Returns ?Col2_Cond_Prop12)
(:test (:lisp (> ?Col2_Cond_Prop12 2000)))
)
```

The following describes in detail how the above textual Algernon query expression is generated by the system:

1. The user starts by creating a query for a neuron (See Figure 7 (a)). The system generates the following Algernon query:

   ```
   (
   (:INSTANCE -Neuron ?Col0_Returns)
   )
   ```

   Note: The system automatically generates a column name `?Col0_Returns` to refer to the instances descibed here. Similar column names are generated at each step.

2. Then, the user chooses a chemical synapse (whose parent is *Inputs* hidden in the Figure 7 (b)), which has a relationship with the neuron. This action by the user on the query interface prompts the system to add two sentences (each sentence in Algernon is enclosed within a set of parentheses) to the Algernon query that connects the inputs of the neuron to the chemical synapse instance using appropriately "joined" variables. The updated Algernon query is:

   ```
   (
   (:INSTANCE -Neuron ?Col0_Returns)

   (-Inputs ?Col0_Returns ?Col1_Returns)
   (:INSTANCE -Chemical_Synapse ?Col1_Returns)

   )
   ```

3. After choosing the Chemical Synapse relationship class, the user chooses the -*Connection_Probability* property of the Chemical Synapse, whose parent class happens to be -*My_Properties* (see Figure 7 (c)). When the user clicks the "Add" button, the query criterion row is displayed by the system into which the user enters the selection condition that the value of that chosen property is *larger than 2* (Figure 7 (d)). Corresponding to all these user interactions, the system adds four more sentences to the Algernon query. The updated Algernon query is:

   ```
   (
   (:INSTANCE -Neuron ?Col0_Returns)
   (-Inputs ?Col0_Returns ?Col1_Returns)
   (:INSTANCE -Chemical_Synapse ?Col1_Returns)

   (-My_Properties ?Col1_Returns ?Col1_Cond_Prop6)
   (:CHILD -Connection_Probability
       ?Col1_Cond_Prop6 ?Col1_Cond6)
   (-Value ?Col1_Cond6 ?Col1_Cond6_Values)
   (:test (:lisp (> ?Col1_Cond6_Values 2)))

   )
   ```

4. Next, the user chooses the article sub-class from the third pulldown menu of classes which is a relationship property of the chemical synapse. The parent class for the article sub-class is *My_Annotations*, which is hidden in the Figure 7 (e)). This action by the user on the query interface prompts the system to add two more sentences to the Algernon query. The updated Algernon query is:

   ```
   (
   (:INSTANCE -Neuron ?Col0_Returns)

   (-Inputs ?Col0_Returns ?Col1_Returns)
   (:INSTANCE -Chemical_Synapse ?Col1_Returns)

   (-My_Properties ?Col1_Returns ?Col1_Cond_Prop6)
   (:CHILD -Connection_Probability
       ?Col1_Cond_Prop6 ?Col1_Cond6)
   (-Value ?Col1_Cond6 ?Col1_Cond6_Values)
   (:test (:lisp (> ?Col1_Cond6_Values 2)))

   (-My_Annotations ?Col1_Returns ?Col2_Returns)
   (:INSTANCE -Article ?Col2_Returns)

   )
   ```

5. Lastly, the user chooses the primitive property -*Year* of the article class (Figure 7 (f)). Upon clicking the "Add" button, the system introduces a second row in the query criteria panel. The user enters the value of the year property as *larger than 2000* (Figure 7 (g)). The final Algernon query is generated with the addition of two more sentences to the query:

   ```
   (
   (:INSTANCE -Neuron ?Col0_Returns)

   (-Inputs ?Col0_Returns ?Col1_Returns)
   (:INSTANCE -Chemical_Synapse ?Col1_Returns)

   (-My_Properties ?Col1_Returns ?Col1_Cond_Prop6)
   (:CHILD -Connection_Probability
       ?Col1_Cond_Prop6 ?Col1_Cond6)
   (-Value ?Col1_Cond6 ?Col1_Cond6_Values)
   (:test (:lisp (> ?Col1_Cond6_Values 2)))

   (-My_Annotations ?Col1_Returns ?Col2_Returns)
   (:INSTANCE -Article ?Col2_Returns)

   (-Year ?Col2_Returns ?Col2_Cond_Prop12)
   (:test (:lisp (> ?Col2_Cond_Prop12 2000)))
   )
   ```

There are several important points to note in the Algernon query generation process:

1. The system systematically introduces new variables as the user continues to interact with the interface and formating the query. The variables are repeated in appropriate places in the new sentences to ensure equality of values in these positions. For example, the variable `?Col0_Returns` appearing in the first line is repeated in the second line

to indicate that the neuron that the variable represents is the one whose inputs are being considered in the second sentence of the Algernon query. This is similar to the join-conditions that are prevalent in relational databases.

2. The query generation user interface allows for the user to return back to a previous step in the formulation process and possibly add new conditions in earlier parts of the query. The interface also allows for parts of the query being formulated to be deleted. The system carefully handles all these cases by keeping track of query strings for various screens and editing these query strings appropriately.

3. The query generation user interface allows the users to choose a sub-class at any level while making progress in the query formulation. Appropriate Algernon queries are generated in such instances.

The final Algernon query, when executed against the Protege KB, generates a list of 3-tuples (Neuron, Chemical Synapse, Article) with the associated relationships between these objects as specified in the query. The results are displayed in a tabular manner.

RELATED WORK AND COMPARISON

Visual querying began with QBE (Query-By-Example) [22] in 1975. Since then, several visual query languages have been developed, such as QBT (Query-By-Templates) [19], Kaleidoquery [14], XQBE (XQuery-By-Example) [3], QBB (Query-By-Browsing) [17]. A visual query builder called Visual XQuery Builder has been distributed by IBM in DB2 Developer Workbench [21]. There have been a number of tools to assist users in building queries for semantic data. Many of them provide intuitive Web-based approaches. Some examples are NITELIGHT [20], MashQL [10], and GLOO [7]. Konduit [2] also allows for the integration of desktop data. These help users create declarative queries with pre-embedded declarative queries instead of using forms. Form-based query interfaces are also widely used to access databases and Jayapandian [11] gave an automated creation method.

Visual Query Systems (VQSs) are defined as database query systems that use a visual representation to depict the domain of interest and express related requests [4]. They are especially designed for end-users who have limited computer expertise and cannot work explicitly with the internal structure of the accessed database. The technology for the construction of VQS is generally well researched and the choice of approach is primarily one of matching other language concepts to provide a clean, conceptually consistent interface [13].

The primary differences between our query system and other visual query systems are as follows:

1. *Web-based:* Ontology-based Web query system was designed to be consistent with an existing ontology query language Algernon which uses forward and backward chaining rules to perform inference on data in knowledge bases. It extends the functionality of the Algernon system by providing a Web-browser user interface which allows user to make the query system accessible from any computer on the internet.

2. *On demand ontology schema retrieval:* Unlike VQS, user interfaces for most query systems are not designed for non-programmers. Such as, Query-By-Example (QBE) [22] which requires the users to remember too much information about database organization (schema). Before users can formulate queries, they must remember or look up the names of classes and properties in the database, which requires database knowledge. Moreover, multiple types of relationships between each class (Inheritance, relate to) are not easy to be understood to end-user. Starting from the first selection of end-users, our query system can dynamically retrieve the part of ontology schema in users' interests, which facilitates them to construct a query expression with minimal database knowledge.

3. *Intermediate Results:* Most visual query systems require a complex and difficult query expression in order to get the intermediate results. With the intermediate results, users can trace the inference chain performed on the data in the knowledgebase. The inference chain information can be easily found in the result table on the summary page. The major/final result is in the first column of the table, and other columns contain intermediate results. A row in the result table represents an inference chain, from which the major result is returned. If the major result can be resulted from multiple chains, there will be multiple rows in the table to reflect this fact.

4. *Supporting user-defined operators:* Our query system supports user-defined operators that are derived from the functions in the ontology. With this support, users can easily build a query according to their domain knowledge. The user-defined operators provide more functionalities and flexibilities to users, and make the query system readily usable.

CONCLUSION AND FUTURE WORK

In this paper, we have presented the Web query sub-system of NeuronBank. The query system provides a visual interface that allows users to express arbitrary queries using an easy to use interface. Classes are dynamically looked up by the system and presented in the visual interface. Primitive properties of classes can be queried by the users as well as relationships with other classes. The user can follow a chain of relationships to formulate complex queries.

The ideas presented in this paper can easily be used to query arbitrary ontologies that are stored in Protege Frames. The system can also be easily modified to work with RDF/OWL ontologies as well. SPARQL queries will have to be generated in this case. The NeuronBank ontology is currently also available in RDF/OWL and we expect to implement the advanced search visual interface discussed in this paper in the near future. It is hoped that the ideas presented in this paper form a general template for a general-purpose visual query interface for ontologies.

ACKNOWLEDGEMENTS

This work is supported by the Brains and Behavior (B&B) program at Georgia State University.

REFERENCES

1. Algernon: http://algernon-j.sourceforge.net/
2. O. Ambrus, K. Möller, and S. Handschuh. Konduit VQB: a Visual Query Builder for SPARQL on the Social Semantic Desktop. *Workshop on Visual Interfaces to the Social and Semantic Web (VISSW), IUI*, 2010.
3. D. Braga, A. Campi, and S. Ceri. XQBE (XQuery By Example): A Visual Interface to the Standard XML Query Language. *ACM Transactions on Database Systems*, 30(2), 2005.
4. T. Catarcia, M. F. Costabileb, S. Levialdic, and C. Batinia. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*. Elsevier, 1997.
5. CIRI: http://www.seco.tkk.fi/events/2004/2004-09-02-web-intelligence/papers/airio_et_al.pdf
6. COQL: http://hcs.science.uva.nl/projects/void/coql.html
7. A. Fadhil and V. Haarslev. GLOO: A graphical query language for OWL Ontologies. http://ceur-ws.org/Vol-216/submission_16.pdf *CEUR Workshop Proceedings*, 2006.
8. J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S.W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies, Volume 58 , Issue 1, Pages: 89-123*. Academic Press Inc., Duluth, MN, USA.
9. T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies, Volume 43 , Issue 5-6 Nov./Dec. 1995, Pages: 907-928* Special issue on the role of formal ontology in the information technology. Elsevier, 1995.
10. M. Jarrar and M. D. Dikaiakos. MASHQL: a query-by-diagram topping SPARQL. *Proceeding of the 2nd international workshop on Ontologies and Information systems for the Semantic Web (ONISW)* ACM, New York, NY, 2008.
11. M. Jayapandian and H. V. Jagadish. Automated Creation of a Forms-based Database Query Interface. http://www.vldb.org/pvldb/1/1453932.pdf *Proceedings of Very Large Databases (PVLDB)*, Vol. 1, No. 1, Pages 695-709, 2008.
12. P. S. Katz, R. J. Calin-Jageman, et. al. NeuronBank: a tool for cataloging neuronal circuitry. *Frontiers in Systems Neuroscience*, 2010.
13. J. Leopold, M. Heimovics, and T. Palmer. WebFormulate: A Web-Based Visual Continual Query System. *Proceedings of the 11th international conference on World Wide Web*, 2002. http://doi.acm.org/10.1145/511446.511476.
14. N. Murray, N. Paton, and C. Goble. Kaleidoquery: A Visual Query Language for Object Databases. *Proceedings of the working conference on Advanced visual interfaces (AVI)* ACM, New York, NY, 1998.
15. NeuronBank system: http://www.neuronbank.org/index.php?section=1
16. OWL. OWL Web Ontology Language. Recommendation, W3C, February 2004. http://www.w3.org/TR/owl-features/
17. S. Polyviou, P. Evripidou, and G. Samaras. Query by Browsing: A Visual Query Language Based on the Relational Model and the Desktop User Interface Paradigm. *Proceedings of the 3rd Hellenic Symposium on Data Management, (HDMS04)*, Athens, Greece, June 2004.
18. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. Recommendation, W3C, January 2008. http://www.w3.org/TR/rdf-sparql-query/
19. A. Sengupta and A. Dillon. Query by Templates: A Generalized Approach for Visual Query Formulation for Text Dominated Databases. *Proceedings of the IEEE international forum on Research and Technology Advances in Digital Libraries (ADL)*. IEEE Press, Washington, D.C., 1997.
20. P. R. Smart, A. Russell, D. Braines, Y. Kalfoglou, J. Bao, and N. Shadbolt. A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer. *16th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2008)* 29th September-3rd October 2008, Acitrezza, Catania, Italy. Springer-Verlag, Berlin, Heidelberg, 2008.
21. Xquery in DB2 Developer Workbench. Introducing DB2 9: Application development enhancements. IBM. http://www.ibm.com/developerworks/data/library/techarticle/dm-0607ahuja/
22. M. Zloof. Query-by-Example: the Invocation and Definition of Tables and Forms. *VLDB*, 1975.