I. Semantic Filtering

We are using the Life Science domain ontology GlycO to showcase our semantic visualization tool. The user can use the Semantic arc filter palette to turn on & off the super & its respective sub properties. The user can filter the arcs just by selecting the higher level arcs which in turn includes the arcs at the lower level. By selecting the super property, it turns on all its sub properties and the user can visualize the selected arcs. The user can also select only the specific sub property arcs and visualize. Additionally, the tool will have the capability of filtering the nodes connected by the property.

II. Glycan Layout

The current release version 0.5 of GlycO contains 573 classes and 113 types of named relationships. It has been semi-automatically populated with more than 480 N-Glycans. Glycan is comprised of carbohydrate residues connected by a relation. One of these concepts is the carbohydrate_residue, or basic unit of glycan structure. Carbohydrate residues are classified according to their structural features, such as absolute conformation (d or l), overall configuration (e.g., gluco or manno), anomeric configuration (α or β), ring form (f or p), and number of main-chain carbons (e.g., hexosyl or pentosyl).

Our approach can be used to visualize any ontology where the generation of images is based on the information in the ontology. The glycan images are automatically generated using the link information (connection that exists between the residues) in the GlycO ontology. Depending on the type of residue, the specialized custom layout/Glycan Layout can display carbohydrate residues in different shapes such as squares, circles, or triangles and generate custom glycan images at runtime. The shapes and colors of residue instances can be changed just by making changes in the layout configuration settings.

More details on how the glycan images are generated are explained below. The Glycan Layout uses configuration or properties file that has all the information needed to generate custom glycan images. The properties file that generates glycan images contain the following information:

- Relationship used for clustering: It clusters on relationship has_carbohydrate_residue which is provided in configuration settings of the layout.
- Associate a shape and color with given residue types or Class: The instances of a given class would have a particular shape and color specified in the configuration file. This is useful in customization of shapes and color depending on the user preferences. In glycan imaging we use the following classes galactosaminyl_residue, glucosamine_residue, mannosyl_residue, glucosaminyl_residue, galactosyl_residue, fucosyl_residue, N-acetyl-galactosaminyl_residue, N-acetyl-neuraminosyl_residue, and N-glycosyl-neuraminosyl_residue. All the above mentioned classes are of type residue. Next we associate shape and color with each residue. For example all instances of the residue type mannosyl_residue will be represented as the shape:circle and color:green. Similarly, all instances of glucosaminyl_residue will be represented as Blue Square, all glucosamine_residue instances will be visualized as Blue Square, all galactosyl_residue instances will be visualized as Yellow Circles, all galactosaminyl_residue instances as Yellow Square, all fucosyl_residue instances as Red Triangles, all N-acetyl-galactosaminyl_residue instances as Yellow Squares, all N-acetyl-neuraminosyl_residue instances as Pink Diamonds, and all N-glycosyl-neuraminosyl_residue instances as Blue Diamonds.
• Link relationships: Links are relationships that connect the residues. Glycan images are generated using the following relationships namely is_linked_to and is_linked_via. Each glycan instance is comprised of a set of residue instances. Each of the residue instances has the relationships is_linked_to and is_linked_via. The relationship is_linked_via has values 2, 3, 4, and 6. The carbohydrate residues that form a glycan are linked to each other through relationship is_linked_to.

• Link values: Our layout algorithm would look for the link values of each residue node of a glycan and based on the link value of the residue, the layout algorithm would position the node at a particular location. There can be four different link values associated with the residues for glycan imaging. The link value of 4 would place the node to the right, link value of 2 would place the node directly below, link value of 6 would place the node diagonally above, and link value of 3 would place the node diagonally below. The glycan image is generated in the form of a tree. The link values are represented in configuration file as follows: value 4 would represent RIGHT, 6 would represent DIAGONAL_UP, 3 would represent DIAGONAL_DOWN and 2 would mean that the node is to be placed DOWN.

III. Ontology browser

This browser is used for easy navigation of the ontology, especially useful as it can quickly retrieve sub classes, sub proprieties, and instances of a given class in the ontology.

IV. Filtering/Un-filtering of search result

Filtering not only removes the node from the shrimp view (i.e. a class-instance), it also will not be considered for the subsequent searches. With our un-filter option, the user is able to view the previously filtered node by clicking on the un-filter button.

V. Searches

Our tool provides an enhanced query interface with additional search capabilities including class search, instance search, relationship search, RDF comment search, triple search, domain-range search and semantic search.

I. Relationship search & description search

Descriptions are comments in the RDF or OWL file. When a keyword search is performed it returns the classes and individuals that contain this keyword. The user can navigate to the specific class-instance by selecting it from the search result.

Relationship search returns the classes & instance that define, restrict or instantiate the relationship that was queried for. The user can then navigate to the specific class-instance by selecting it from the search result. The Shrimp View in Jambalaya-plus displays the selected class-instance.
II. Triple Search

Triple search can be used to search either the subject or object instance given a subject instance and a property or an object instance and a property. The result of the triple search is the object/subject instances and its respective classes.

Triple search has an enhanced option to perform domain/range search. Given a domain and a property, the search can returns range class and its respective instances. Similarly while searching for a domain, given a range class and a property, it finds domain class and its respective instances. The user would be able to navigate to the respective class(instance) by selecting the class/instance from the query result.

IV. Semantic Search

Semantic search can be used to retrieve only the right instances; i.e. instances that belong to a particular class or instances that have some relationship with the direct or derived instances of another class also known as context class, as opposed to a traditional keyword search that returns all the matched instances in the dataset. Thus, semantic search can be used to disambiguate instances and reduce the number of entries in the search result.

Suppose we do a search for Madonna in a dataset that consists of musician and other popular people, we might come across Madonna the singer, or Madonna the Basketball player, or Madonna the entrepreneur.

Thus, querying for Madonna in the context of Singer would return only Madonna the Singer. Further, we have extended the semantic search to find the Madonna who has some direct relationship with instances of the context class or its sub classes. For example, when the user is searching for Singer Madonna who has sung Frozen, then semantic search on Madonna would ensure that it returns only the singer Madonna who has sung Frozen, rather than returning different Madonna that exists in the dataset. The user can specify different context classes such as Pop song, or Song (which is a more general class than Pop song) since Frozen is a Pop song.

The user would fill in the context, and the class in the text field of the search interface. In the above example, the user would fill in the context class as Pop Song or Song, and the class as Person. Internally the query would be evaluated by getting all the instances of the class Person and later by finding all instances that have some relationships with the direct or derived instances of the context class Song.

If the user is interested in finding the instances that form a specific relationship with the direct instances or derived instances of a context class, he would specifies the pattern text in the source instance text field, relationship in the relationship text field along with the context class that it relates to. An example would be as follows: To find all instances that begins with a prefix pattern ‘m’ and has relationship such as has_sung with the instances of the context class Song or its derived instances, user would enter ‘m’ in the source instance text field, has_sung in property text field, and Song in the context class text field respectively.