Developing A Web-based User Interface for Semantic Information Retrieval

Daniel C. Berrios[1], Richard M. Keller[2]

[1]Research Institute for Advanced Computer Science, MS 269-2,
NASA Ames Research Center, Moffett Field, CA USA 94035
[2]Computational Sciences Division, MS 269-2,
NASA Ames Research Center, Moffett Field, CA USA 94035
{berrios, keller}@email.arc.nasa.gov

Abstract

While there are now a number of languages and frameworks that enable computer-based systems to search stored data semantically, the optimal design for effective user interfaces for such systems is still unclear. Such interfaces should mask unnecessary query detail from users, yet still allow them to build queries of arbitrary complexity without significant restrictions. We developed a user interface supporting semantic query generation for SemanticOrganizer, a tool used by scientists and engineers at NASA to construct semantic networks of knowledge and data. Through this interface users can select node types, node attributes and node links to build *ad-hoc* semantic queries for searching the SemanticOrganizer network.

Introduction

To imbue web documents with machine-readable semantic content, authors now have formats such as RDF for storing such content (Lacher & Decker, 2001) and tools like Annotea and the SHOE Knowledge Annotator (Heflin, Hendler, & Luke, 1999) to help create such content. Furthermore, standards for query languages to search this content are also beginning to emerge (Miller, Seaborne, & Reggiori, 2002). However, there are still very few tools to help users create semantic queries in any of these languages, and the design of such tools remains the subject of ongoing research.

We have developed a user interface for building semantic queries of arbitrary complexity for SemanticOrganizer[1] (SO), a combined knowledge and data repository that features an extensive semantic network. Through this interface a user can generate a complex query to search the SO knowledge space for sets of items consistent with the query. The queries are stored as RDF models with anonymous nodes, hidden within HTML pages of the interface, and incrementally updated as the user builds a query.

We approached the design of this interface with the twin goals of accommodating users who know nothing or very little about RDF and presenting the queries in a simple, straightforward manner. SO has a wide array of users who vary in technical savvy and who use a variety of computing platforms and software. By and large, most user interaction with SO is via HTML forms, and we have eschewed more sophisticated interfaces such as specialized java applet widgets largely because of cross-platform/browser compatibility issues. Thus, we sought to develop a semantic searching interface using only HTML technology.

Methods

Because the task of building all but the simplest query can require substantial cognitive reasoning on the part of users, we chose a successive refinement design for the query building interface (Fig. 1). Users iteratively add "terms" to a query; each term is represented as a typed, but otherwise anonymous node, similar to a resource in the RDF model. A node is added by linking it to a node already in the model through a "link" type property selected from the SO knowledge network (i.e., a property whose range must be another resource). The query can be submitted for execution any time after the first node is created and added to the model. In fact, the user can continue to refine the query and/or submit it for execution even after search results are presented.

Figures 2 through 7 show the development of a query to search for all DNA sequences from any bacterial culture of a (stromatolite) sample with certain properties. Figure 2 shows a user beginning to build a query using the interface. The interface is separated by a simple horizontal line into an upper **query building area**, in which users select and

[1] http://sciencedesk.arc.nasa.gov/

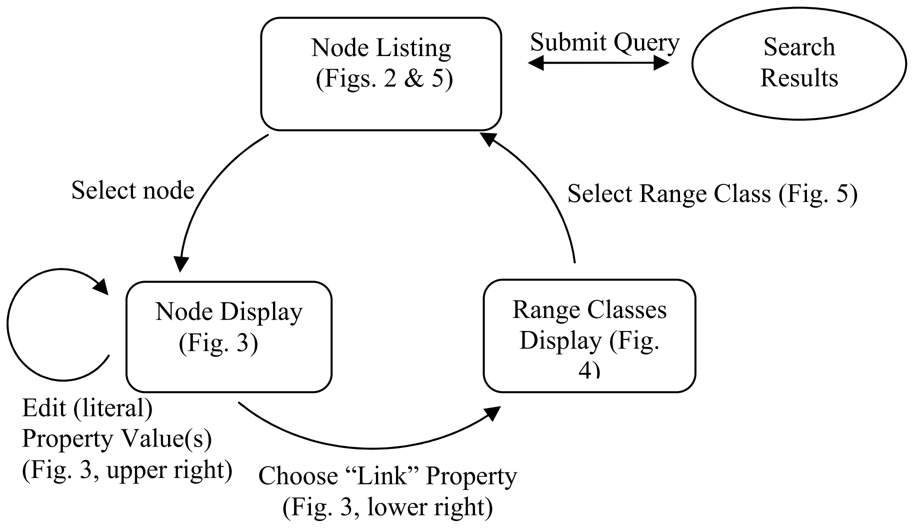

Node Listing (Figs. 2 & 5)
Submit Query
Search Results
Select node
Select Range Class (Fig. 5)
Node Display (Fig. 3)
Range Classes Display (Fig. 4)
Edit (literal) Property Value(s) (Fig. 3, upper right)
Choose "Link" Property (Fig. 3, lower right)

Figure 1. Flow of user interaction with the SemanticOrganizer query building interface. Given a listing of nodes currently in the query (top), the user can select any node for display of its properties (bottom, left), edit any of the properties, then choose a link type to relate the new node to the next new node in the query, which brings up a display of the possible range node types for that link type (bottom, right). At any time, the user may submit the query for execution, view search results (top, right), and (optionally) continue building the query.

edit terms in the query, and a lower **query execution area**, in which users can choose to submit a query, view search results, or erase the current query and begin again. Because the query (in its current state) is stored client-side (i.e., embedded within the web page) and not server-side, the user can "back up" to previous versions of the query at will using only their browser's navigation buttons and pursue different paths of query refinement.

As shown in Figure 2, the user begins to build the query by selecting the type "DNA Sequence" for a new (anonymous) node in the model, labelled "DNA Sequence 1." All nodes are typed and so labeled by order of creation. Specifying the type for a node in the query adds a statement to the RDF model that restricts the type property of the node to the appropriate class. The interface requires the user to select a type for each node before any of the node's properties can be defined. While this design choice initially followed logically from the types of queries we solicited from potential users (e.g., "Find all experiments…", "Find all samples…", etc.), it also obviates the need to develop methods for users to sort through the dozens or even hundreds of possible properties defined on all the various node types in a given domain. Instead, the interface only needs to display those properties whose domain is the type of node selected.

After the user chooses the type of node to be added to the model, the interface displays the Edit Node form (Fig. 3). This form allows the user to enter or select literal-valued properties of the node, or select from a list of properties that have other nodes as ranges to link this node to other nodes in the model. Literals can be specified by entering them directly or selecting from a list of allowed val-

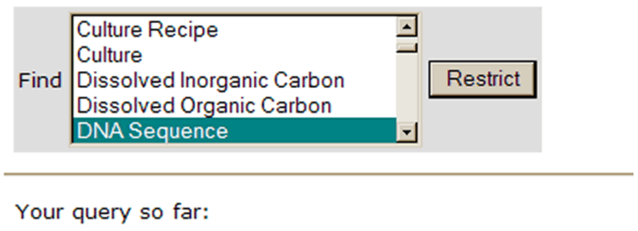
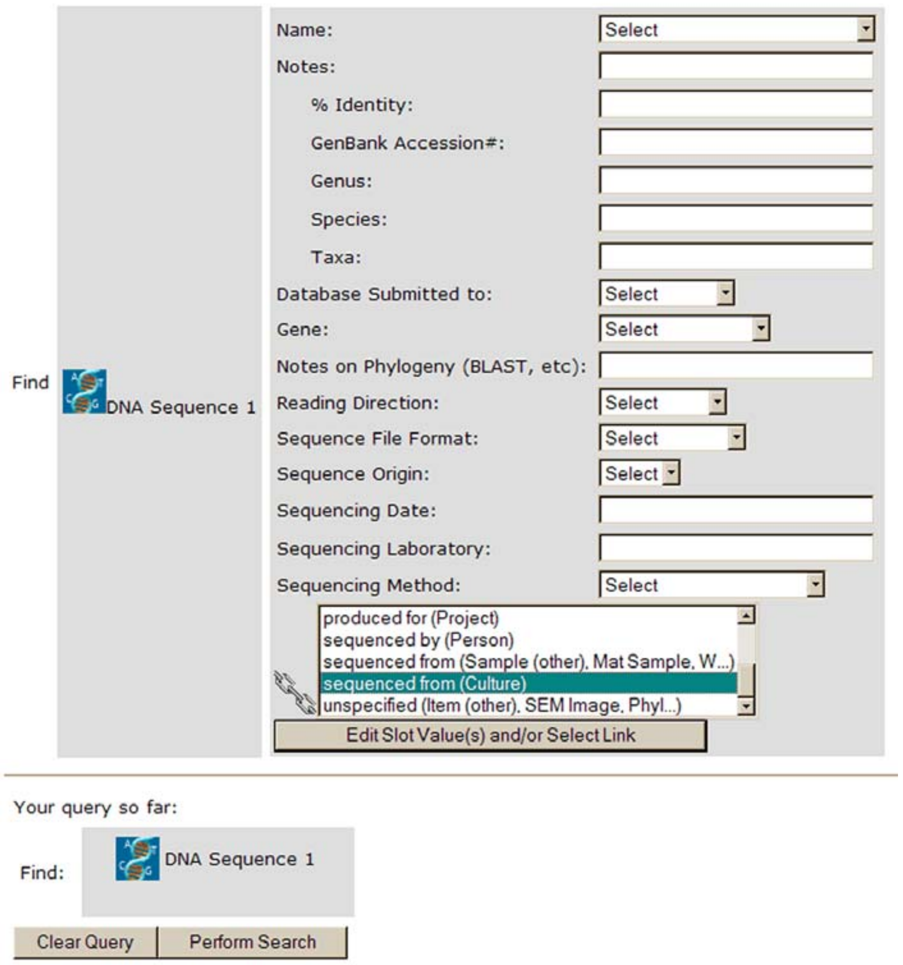


Figure 2. The **Node Type Selection** display of the query builder interface. The user must select a type for the first (and each successive) node in the query



Name:	Select
Notes:	
% Identity:	
GenBank Accession#:	
Genus:	
Species:	
Taxa:	
Database Submitted to:	Select
Gene:	Select
Notes on Phylogeny (BLAST, etc):	
Reading Direction:	Select
Sequence File Format:	Select
Sequence Origin:	Select
Sequencing Date:	
Sequencing Laboratory:	
Sequencing Method:	Select

Figure 3. The Edit Node Properties step. After selecting a type for the new node, the user is presented with a form to select/edit literal property values (upper right) and/or choose a "link" type property (lower right) to connect the new node to another node

ues (if such a list is defined for the property type) and submitting the form; the returned page displays the values along with an adjacent "scissors" icon (see Figure 7) which can be used to submit the form again, this time removing the value. Values for any number of literal properties may be submitted all at once or in any sequence as many times as desired. However, once the user selects a "link" type property and submits the form, the interface requires the user to specify a class for the range of this property (Fig. 4). After the user has selected a range type, a new node ("Culture 1") is added to the model, as well as a statement restricting its type to the type specified, and a statement linking the two nodes through the selected property. This action returns the user to the Node Listing Display, showing the two newly created anonymous nodes along with the list of all node types (Fig. 5).

At this point the user can either select one of the existing nodes in the model (to add other links and/or property values) or choose the type for a third new node in the model. He or she can continue the cycle of creating and editing existing nodes at will until satisfied with the query. This cycle could produce, for example, the complex query shown in Figure 7.

In the query execution area of the interface, we display generated queries in tabular form, which is well-supported in HTML. Each node in the query is assigned a corresponding column in the tabular display, and each row displays one or more links between nodes. While this format may not be concise, it is probably superior to merely listing the nodes and links of the model.

We could have designed the interface such that users could create any type of graph structure, including those with cycles. However, the use of HTML tables to display queries with cycles clearly and unambiguously appeared very challenging, if not impossible. Thus, we chose not to allow users to generate cyclical query structures using this initial version of the query-building interface.

At any time during the process of building the complex query, the user may choose to completely erase the query through the “Clear Query” button or execute the query by pressing the “Perform Search” button. Choosing to erasing the query removes the RDF model embedded in the page and returns the user to the first step in the query building process (see Figure 2).

To execute the query, we viewed searching the SO knowledge space using the generated query as a constraint satisfaction problem (CSP) (as others have): the nodes in the query represent the set of variables in the CSP, the items in SO correspond to the domain of possible values for these variables, and the various properties in the RDF model that the user specifies represent the constraints. We developed procedures to solve this CSP using common programming techniques to increase efficiency, including node and arc-consistency tracking.

Figure 7 shows the search results for the query shown in Figure 7. Each node in the query corresponds to a column in the results table, and the possible sets of values for the nodes are listed as rows. Clicking on a particular value shows the item in SO.

Discussion

We present our experience developing a complex query generation interface that we hope will be effective and at the same time intelligible to naïve web users. The missions and scientific activities conducted at NASA often involve users with a wide variety of sophistication in computer science and experience with computing tools. Yet even unsophisticated users have advanced information needs that will require them to be able to specify complex queries.

There are several features that we realize users need and the current interface lacks. We will extend the functions of the interface to include selecting and searching for multiple values for literal-valued properties (using Boolean OR), specifying ranges of values for special types of literals such as dates and times, and range sets for link-type properties.

Because building some queries often requires significant time and thought, we are also developing methods for users to store, retrieve, clone, re-edit and re-execute complex queries. We are currently exploring reuse of inference rule “building blocks” for the creation of reusable complex que-

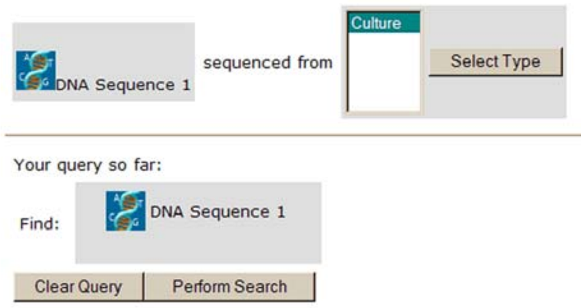


Figure 4. After selecting a "link" type property, the user is required to choose from a set of possible range classes. In this case, there is only one possible range class, "Culture", defined for the "sequenced from" property

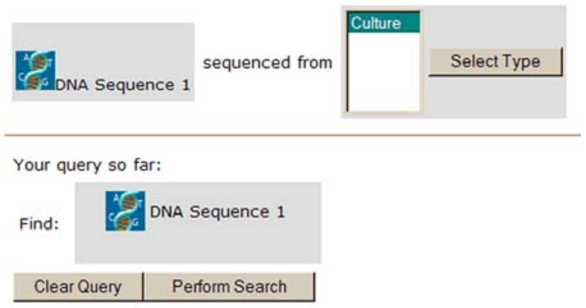
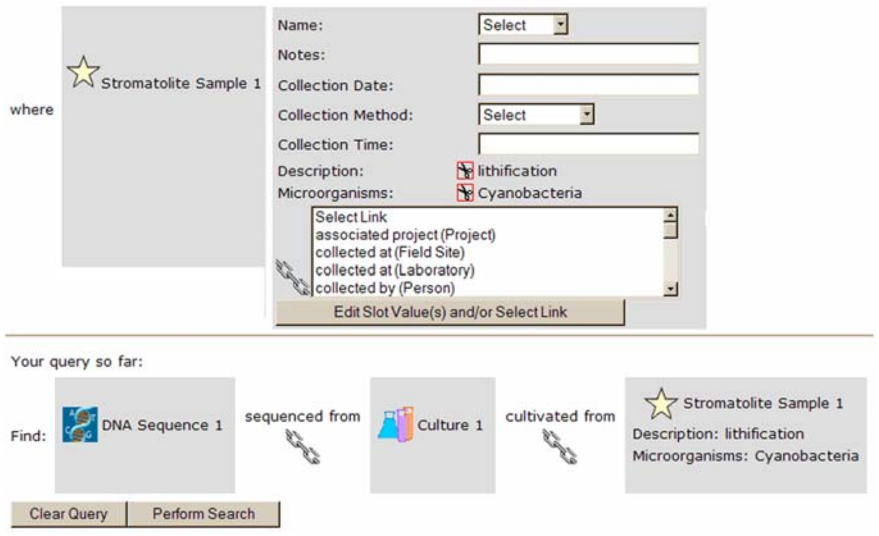


Figure 5. After selecting a "link" type property, the user is required to choose from a set of possible range classes. In this case, there is only one possible range class, "Culture", defined for the "sequenced from" property



where

Stromatolite Sample 1

Name: Select

Notes:

Collection Date:

Collection Method: Select

Collection Time:

Description: lithification

Microorganisms: Cyanobacteria

Select Link
associated project (Project)
collected at (Field Site)
collected at (Laboratory)
collected by (Person)

Edit Slot Value(s) and/or Select Link

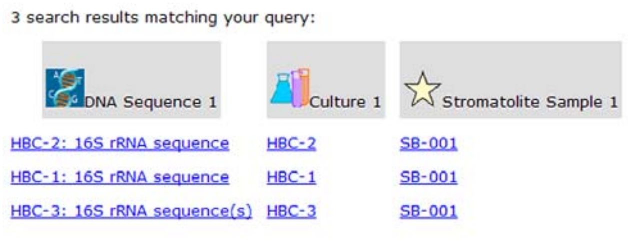
Your query so far:

Find: DNA Sequence 1 — sequenced from — Culture 1 — cultivated from — Stromatolite Sample 1
Description: lithification
Microorganisms: Cyanobacteria

Clear Query | Perform Search

Figure 7. The final complex query to search DNA sequences from any culture cultivated from a sample of Cyanobacteria showing lithification

ries. These building blocks include knowledge represenations for anonymous nodes and their properties (i.e., represented as triples) that can be combined through a subset of first order logic operators to form rule predicate sets. For example, a user can create a rule that looks for any field trips undertaken by a workgroup, and sets the "participated in" property for each member of the workgroup to those field trips, if the workgroup participated in the field trip as a group. The predicates for this rule stipulate that there exist a node, *x*, of type "person" that has the property "member-of" with a value of node *y*, and that *y* is of type "workgroup" and has the property "participated in" with a value of node *z* and also that *z* is of type "field trip." The firing of the rule sets the property "participated in" of *x* (the person) to the value *z*. These same predicate sets capture the logical predicates embodied in the complex query "find me all nodes of type person such that these persons are members of a workgroup that participated in some field trip." (However, there is no equivalent to the rule conclusion – only that the values for variables in each of the predicates be returned) By implementing the execution of such complex queries using persistent inference rule predicates, users will be able to store, retrieve, and share common queries. Furthermore the result sets retrieved by the queries could also be persistently stored and then continuously updated using a standard inference engine.



3 search results matching your query:

DNA Sequence 1	Culture 1	Stromatolite Sample 1
HBC-2: 16S rRNA sequence	HBC-2	SB-001
HBC-1: 16S rRNA sequence	HBC-1	SB-001
HBC-3: 16S rRNA sequence(s)	HBC-3	SB-001

Figure 6. Example search results. Each row represents a set of possible values for each node in the model

Acknowledgements

We wish to thank Brad Bebout and Lee Prufert-Bebout for the kind use of their data in our examples. We would also like to thanks Mr. Shawn Wolfe.

References

Heflin, J., Hendler, J., & Luke, S. (1999). *SHOE: A Knowledge Representation Language for Internet Applications.* (Technical Report CS-TR-4078 (UMIACS TR-99-71)): Dept. of Computer Science, University of Maryland at College Park.
Lacher, M. S., & Decker, S. (2001). RDF, topic maps, and the semantic Web. *Markup Languages: Theory & Practice, 3*(3), 313-331.
Miller, L., Seaborne, A., & Reggiori, A. (2002). *Three implementations of SquishQL, a simple RDF query language.* Paper presented at the ISWC 2002. First International Semantic Web Conference Proceedings, Sardinia, Italy.