Automatic query generation in guided systems
Natural language generation from graphically built query

Sara Paiva
Viana do Castelo Polytechnic Institute
Viana do Castelo, Portugal
sara.paiva@estg.ipvc.pt

Manuel Ramos-Cabrер
Vigo University
Vigo, Spain
mramos@det.uvigo.es

Alberto Gil-Solla
Vigo University
Vigo, Spain
agil@det.uvigo.es

Abstract—Guided-based systems are an alternative to natural language systems regarding the query construction method. We present a new solution that bridges the gap between these two types of systems by providing a hybrid interface which combines a graphical query interface for query construction with the correspondent natural language expression automatically generated by the system.

Keywords: Natural language expressions, automatic generation, guided-based systems, query construction

I. INTRODUCTION

Semantic search has emerged as a way to increase the accuracy of the searching activity. The incorporation of semantics – meaning of the language – empowers search engines with a better understanding of the search expression which leads to more accurate results. Semantic search optimum vision is to allow the use of natural language (NL) input. However, this approach is still not able to produce accurate results as the query is not fully interpreted and understood by the system [1].

The guided construction method arises as an alternative for NL input. We refer to these systems as guided-based systems. They guide the user in the query construction what makes them less flexible than NL systems but they can produce totally accurate results what makes it promising alternative. Despite this, there is still room for improvement in these systems.

Guided-based systems use several approaches for building the query such as auto complete functionalities, menu-based or graphical interfaces. Our study is focused on the last option as we believe this query construction approach to be an advantage for the end user as it is simpler and intuitive. In this paper we introduce our solution which has a hybrid interface where the user can build the query graphically and, simultaneously, see it expressed in natural language. This user interface proposal is based on conducted surveys [3] where users manifest their preference for a combination of two functionalities to construct the query: use of natural language and graphical construction.

II. GUIDED-BASED SYSTEMS INTERFACES

As mentioned above, guided-systems use several approaches for building the query. Ginseng [2] uses an auto complete approach as the system predicts the possible completions of what the user entered in a free form entry field.

Semantic Crystal [3], TAMBIS [4] and LingoLogic [5] all provide a graphical query construction method. The first two systems present much of the ontology’s formalisms (classes and properties are presented to the user with the same designation from the ontology which in the great majority is closest to a code than to a real word concept) which represents a disadvantage to the end user, not necessary familiar with this notation. LingoLogic’s interface is menu-based which makes it different from the two previous systems. The formalism issue raised above is well addressed in LingoLogic as the options shown to the user are real-world concepts. However, we find the menu-based approach less attractive and easier for the end user.

Regarding the usability of the interfaces, Semantic Crystal presents the query as a graph which makes it hard to read and difficult to understand. TAMBIS has also an unfriendly query presentation as it relies on a presentation structure that resembles a tree, with a hierarchical form. As mentioned above, LingoLogic presents the user with the possible options in a menu. As the query is built, the options shown are already presented in order to make the query look like natural language.

We see our solution as “the best of both worlds” as it allows the graphical creation of the query in order to facilitate this process to the user and, simultaneously, generate the correspondent natural language expression so the query is easily understood.

III. OUR SOLUTION

Our solution provides a hybrid interface as it offers a graphical interface for the query construction and a real-time natural language expression generated automatically by the system. This user interface proposal is based on conducted surveys [3] where users manifest their preference for a combination of two functionalities to construct the query: use of natural language and graphical construction.
A. Solution overview

The solution has three main components:
- A semantic validation rules engine (SVRE)
- A natural language generation engine (NLGE)
- A hybrid interface

Other components that complement our solution include:
- A search expression representation component (SERC) that is used as an input to the NLGE and the SVRE
- An XML file with the rules to apply depending on the data type, conjunction and involved operators
- An XML file for keeping personalized synonyms for ontology concepts
- A self-developed API for reading RDF ontologies (.NET platform, C# language)
- An XML configuration file that keeps information about the system’s supported data types and supported comparison operators for each data type

B. Ontology construction guidelines

The developed solution relies on an ontology with some construction rules in order to allow a simpler and friendlier user experience, the execution of validation rules, the generation of natural language expressions and also to make the distinction between search and auxiliary classes. Thus, for the first purpose, classes and properties must have two annotations: a short designation (anot_short_design), a full description (anot_full_descr). They are used to present the user with the possible options during the query construction showing him friendly names and not codes. Along with the friendly name of the property/class, a more detailed description is given to clarify the concept. For the second and third purposes, the datatype properties defined in the ontology must have a supported data type associated. Also for the third purpose, classes and properties must have an annotation (anot_exp_for_query) indicating the expression to use in the build of the query. Properties that require an additional expression after the restriction value (e.g. expression euros in the restriction “books which price is 50 euros”) should express that in the annotation anot_after_restriction_value. Finally, for the forth purpose, classes must have an annotation representing the type of class (anot_type_class).

C. The hybrid interface

The interface follows a guided-based approach where the query is built graphically. At each moment, only the possible classes/properties are presented, using the designation in the anot_short_design annotation. At the same time, the full description is shown using the anot_full_descr annotation. One of the assumptions made in our solution, which is controlled by the interface, is the use of only one conjunction when more than one restriction is specified. The search expression “books whose price is higher than 30€ or whose publishing year is 1999 and whose number of pages is less than 200” demands the use of precedence which would lead to great complexity, usually inexistent in search engines queries. However, two conjunctions can appear in the query as long as one of them is used to join the several restrictions and the other to specify different values for a given property. The query: “books whose price is higher than 30€ and whose publishing year is 1999 or 2000 and whose number of pages is less than 200” is an example of a valid query using two conjunctions.

During the query construction phase, the interface communicates with the NLGE for natural language expression generation. When creating a new restriction, the interface communicates with the SVRE to validate the semantics of the new sentence.

D. The Search Expression Representation Component (SERC)

SERC is a component that represents a search expression. It is used as a mean of interaction between the user interface and the NLGE and SVRE.

SERC defines two main classes – Expression and Restriction – and two other classes as sub-classes of Restriction: FunctionalRestriction and NonFunctionalRestriction. The class diagram is illustrated in Figure 1.

The fields in class Expression indicate the search class of the expression (exp_search_cl) – the main thing the user is looking for (eg.: books, movies); the main property of the expression (exp_main_prop) – a specific property of the search class the user wants to see (eg.: title); the conjunction used between the several restrictions (when at least two exist) (exp_conjunction) and the several restrictions defined in the search expression (exp_restrictions).

The class Restriction is used for restrictions over datatype properties or over object properties. The fields in this class indicate whether the restriction is over an object property or over a datatype property (rest_type), the object property name (if it exists) (rest_obj_prop_name), the datatype property name (which always exists) (rest_dt_prop_name) and the comparison operator (rest_comp_operator).

The FunctionalRestriction class inherits the fields from the Restriction class and defines an additional string field representing the value for comparison (value).

The NonFunctionalRestriction class inherits the fields from the Restriction class and defines an additional Arraylist

![Figure 1. SERC component (class diagram)](image)
of string fields representing the list of values for comparison (values).

E. The Natural Language Generator Engine (NLGE)

The syntactic rules defined in guided-based systems aim to guide the user into building a syntactic valid query as the user types it. The syntactic rules of our solution have a different purpose which is to allow the generation of a natural language expression so it is possible to get the most from a guided-based system, making it closer to NLIs and improving the user experience.

The NLGE is accessed by the user interface in three possible stages of the query construction phase, by the following order: (a) when the search class is chosen; (b) when (and if) the main property is chosen and (c) when a restriction is added.

1) The syntax of construction

In (a) the expression only contains the search class; NLGE generates the natural language expression as follows:

\[
\text{SYNTAX}
\begin{align*}
& [\text{all available}] \\
& [< \text{anot}\_\text{exp}\_\text{for}\_\text{query}> \text{search}\_\text{class}]
\end{align*}
\]

In (b) the user chooses the main property to obtain. Considering no restrictions can exist at this point (according to the query construction process), the natural language expression is generated as follows:

\[
\text{SYNTAX}
\begin{align*}
& [< \text{anot}\_\text{exp}\_\text{for}\_\text{query}> \text{main}\_\text{property}] \\
& \text{of all available} \\
& [< \text{anot}\_\text{exp}\_\text{for}\_\text{query}> \text{search}\_\text{class}]
\end{align*}
\]

2) The rules

In (c) the search expression has one or more restrictions. We will use [BEGIN_QUERY] to refer to the beginning of the query as mentioned above (either with or without the main property).

SyR1 – restriction over a datatype property: Restriction construction rule when using a datatype property associated with the selected search class

\[
\text{SYNTAX}
\begin{align*}
& \text{[BEGIN_QUERY] whose} \\
& [< \text{anot}\_\text{exp}\_\text{for}\_\text{query}> \text{restriction}\_\text{dt}\_\text{property}] \\
& \text{[comparator}\_\text{operator}] \\
& \text{[restriction value]}
\end{align*}
\]

SyR2 – restriction over an object property: Restriction construction rule when using a datatype property of a class referenced by an object property associated with the selected search class

\[
\text{SYNTAX}
\begin{align*}
& \text{[BEGIN_QUERY] whose} \\
& [< \text{anot}\_\text{exp}\_\text{for}\_\text{query}> \text{restriction}\_\text{obj}\_\text{property}\_\text{class}] \\
& \text{[comparator}\_\text{operator}] \\
& \text{[restriction value]}
\end{align*}
\]

SyR3 – representative word after property value: Some properties are better described if a word is placed after the property value. This word is defined in the ontology, in an annotation associated to the property.

\[
\text{SYNTAX}
\begin{align*}
& \text{(Example using a datatype property)} \\
& \text{[BEGIN_QUERY] whose} \\
& [< \text{anot}\_\text{exp}\_\text{for}\_\text{query}> \text{restriction}\_\text{property}] \\
& \text{[comparator}\_\text{operator}] \\
& \text{[restriction value]}
\end{align*}
\]

SyR4 – data type properties come first: when a restriction has restrictions over data type and object properties, the restrictions over data type properties come first in the NL expression

\[
\text{SyR5 – group together restrictions over the same property:} \text{ when there are several restrictions over the same property, the words “whose” and the name of the property should not be repeated for each restriction}
\]

\[
\text{SyR6 – use of comma with more than two restrictions:} \text{ when a search expression has more than two restrictions, the comma should be used to separate the all restrictions except the last two, where the conjunction should be used. If there are several restrictions over the same property and grouping is used, this rule is also applied to the set of grouped restrictions.}
\]

3) Application of rules

Whenever the user makes a change to the query, the interface communicates with the NLGE so the natural language expression is generated. NLGE provides an API for this purpose. The main method is called generateNLExpression and receives one argument of type Expression, defined in the SERC component. The method returns a string corresponding to the natural language expression generated by the engine. Figure 2 shows the algorithm followed in order to decide which rules to apply, and illustrates the following behaviors:

- When using one restriction only, the following rules apply: SyR1 (data type property only), SyR2 (object property only) and SyR3
- When using both data type and object properties, SyR4 is also applied.
- When using more than one restriction over the same property, the rule SyR5 is also applied.
- When using more than two restrictions in the search expression or in a group, SyR6 is also applied.
F. NLGE in action

1) The demo ontology

In order to demonstrate how NLGE generates the natural language expressions, we present in this section a simple ontology built with Protégé, only with the necessary annotations for the purpose of natural language automatic generation process demonstration. Two classes were created, one object property and five data type properties. Figure 3 and 4 provide a graphical representation of this ontology.

Figure 2. NLGE algorithm

Figure 3. Demo ontology – class “cl_publisher”

2) Applying rules for expression generation

In this section we present several examples that show the pertinence of NLGE as an enhancement for query representation for the end user.

For each case, we present the Expression object, the annotations of classes and properties necessary to build the query, and the final natural language expression generated, according to the syntax and rules (if any) applied.

Query nr. 1: search class and main property

Figure 5 shows one of the simplest cases which consists of an Expression object with the search class and the main property only. In this case, no rules apply.

Query nr. 2: one restriction (object property)

This second example is an extension of the first one (therefore share the same exp_search_cl and exp_main_prop fields) and uses an object property, which is identified in the Expression object by the rest_type field. Thus, the syntax for object properties is applied in this case.

Figure 6 illustrates the restriction using an object property and the generated natural language expression.

Figure 4. Demo ontology – class “cl_books”

Figure 5. NL generation with search class and main property

Expression object input of search expression

<table>
<thead>
<tr>
<th>exp_main_prop</th>
<th>exp_search_cl</th>
</tr>
</thead>
<tbody>
<tr>
<td>• dp_title</td>
<td>• cl_books</td>
</tr>
<tr>
<td>• anot_exp_for_query “title”</td>
<td>• anot_exp_for_query “books”</td>
</tr>
</tbody>
</table>

Applying the Syntax results in the following NL expression:

title

of all available books
The applied rules are SyR2, SyR3. The first one is applied because the restriction is over an object property. The second one is always applied, according to the algorithm presented in Figure 2.

**Query nr. 3: one restriction (datatype property)**

This example is similar to the previous one, but in this case a datatype property is used. Figure 7 illustrates the restriction using a datatype property and the generated natural language expression. The applied rules are SyR1, SyR3. The first one is applied because the restriction is over a datatype property. The second one is always applied, according to the algorithm presented in Figure 2.

**Query 4: several restrictions**

In this last example, we present a query with several restrictions, over datatype and object properties.

The exp_search_cl and exp_main_prop fields remain the same. In addition, because the query has several restriction, we have to consider a conjunction that will be, for this example, the conjunction and.

As the natural language generation process of both types of restrictions has already been explained, for this example we will introduce each restriction already generated in a natural language expression, as our main goal with this example is to show how the defined rules contribute to an easier to read natural language expression.

**Restrictions in NL**

- Number of pages is higher than 200
- Name of the publisher is Editora Sextante
- Number of pages is less than 300
- Location of the publisher is Lisbon
- Price is higher than 5€

**NL expression generated without rules**

Title of all available books whose number of pages is higher than 200 and whose name of the publisher is Editora Sextante and whose number of pages is less than 300 and whose location of the publisher is Lisbon and whose price is higher than 5€

**Identified problems**

The generated NL expression without rules presents the following limitations: (1) restrictions over the same property are not grouped together; (2) expression restrictions are all concatenated using the conjunction instead of commas; (3) excessive repetition of the word “whose” and of the property name. All these problems combined result in a difficult to read and confuse expression to the end user.

**The algorithm step-by-step**

1. **Processing search class and main property:**
   - Title of all available books

2. **Processing restriction 1 (applying SyR1, SyR3)**
   - Title of all available books whose **number of pages** is higher than **200**

3. **Processing restriction 2 (applying SyR2, SyR3, SyR4)**
   - Title of all available books whose **number of pages** is higher than **200** and whose **name of the publisher** is **Editora Sextante**

4. **Processing restriction 3 (applying SyR1, SyR3, SyR4, SyR5)**
   - Title of all available books whose **number of pages** is higher than **200** and is less than **300** and whose name of the publisher is Editora Sextante

**Notes:** restrictions over **number of pages** are aggregated and repeated words are avoided
IV. CONCLUSIONS

In this paper, we presented a new approach to guided-based search systems with a new functionality that bridges the gap between these systems and NLIs. Some guided-based systems rely on a graphical interface for the query construction which is, in our opinion, an advantage for the end user.

Our solution improves this behavior by generating, simultaneously to the graphical construction, a natural language expression. This way, the user has two distinct and complementary representations of the query. For this purpose, we introduced the natural language generator engine (NLGE), a component of our solution which generates a natural language expression corresponding to the query graphically built. We presented the syntax of the expression construction with the search class and the main property and also with datatype and object properties.

Finally, we showed the NLGE in action, by providing four examples of queries and showing how the NLGE generates the natural language expression, pointing out the advantages that the NLGE brings to the process.

V. FUTURE WORK

As a future work, we intend to improve the definition of rules used by the NLGE, by grouping them in three types: 1) rules that define syntaxes, 2) syntax rules used by the algorithm and 3) syntax rules guaranteed by the interface itself, at the query construction moment. For this improvement, some rules are already defined (and presented in this paper) and some will need to be created.

In 1) syntax must be defined for the following cases: when the search class is chosen; the main property is chosen; a datatype property is chosen in a functional/non functional property and an object property is chosen in a functional/non functional property.

The goal of rules defined in 2) is to define the algorithm itself, improving it. These rules intend to assure: datatype properties are presented first; restrictions over the same property/operator are grouped; the comma is used when more than two restrictions exist and the set of values specified by operators contains and does not contain are aggregated.

In 3) we intend to include a rule to guarantee only one conjunction is chosen to unify all restrictions and another one that applies to non functional properties and guarantees that the conjunction used to specify the different values of the set is the and conjunction.

Beside the reorganization and definition of some new rules, we intend to develop a validation prototype for this solution. It would also be relevant to consider expressions with more restrictions over non functional properties as the specification of several values in each restriction can introduce some complexity in the generation of the final NL expression.

REFERENCES


