Incremental Query Generatio

Laura Perez-Beltrachini, Claire Gardent, Enrico Franconi

Incremental Query Generation

Laura Perez-Beltrachini
Faculty of Computer Science
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy
laura.perez@loria.fr

Claire Gardent
CNRS/LORIA
Nancy, France
claire.gardent@loria.fr

Enrico Franconi
Faculty of Computer Science
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy
franconi@inf.unibz.it

Abstract

We present a natural language generation system which supports the incremental specification of ontology-based queries in natural language. Our contribution is two fold. First, we introduce a chart based surface realisation algorithm which supports the kind of incremental processing required by ontology-based querying. Crucially, this algorithm avoids confusing the end user by preserving a consistent ordering of the query elements throughout the incremental query formulation process. Second, we show that grammar based surface realisation better supports the generation of fluent, natural sounding queries than previous template-based approaches.

1 Introduction

Previous research has shown that formal ontologies could be used as a means not only to provide a uniform and flexible approach to integrating and describing heterogeneous data sources, but also to support the final user in querying them, thus improving the usability of the integrated system. To support the wide access to these data sources, it is crucial to develop efficient and user-friendly ways to query them (Wache et al., 2001).

In this paper, we present a Natural Language (NL) interface of an ontology-based query tool, called Quelo[1], which allows the end user to formulate a query without any knowledge either of the formal languages used to specify ontologies, or of the content of the ontology being used. Following the conceptual authoring approach described in (Tennant et al., 1983; Hallett et al., 2007), this interface masks the composition of a formal query as the composition of an English text describing the equivalent information needs using natural language generation techniques. The natural language generation system that we propose for Quelo's NL interface departs from similar work (Hallett et al., 2007; Franconi et al., 2010a; Franconi et al., 2011b; Franconi et al., 2010b; Franconi et al., 2011a) in that it makes use of standard grammar based surface realisation techniques. Our contribution is two fold. First, we introduce a chart based surface realisation algorithm which supports the kind of incremental processing required by ontology driven query formulation. Crucially, this algorithm avoids confusing the end user by preserving a consistent ordering of the query elements throughout the incremental query formulation process. Second, we show that grammar based surface realisation better supports the generation of fluent, natural sounding queries than previous template-based approaches.

The paper is structured as follows. Section 2 discusses related work and situates our approach. Section 3 describes the task being addressed namely, ontology driven query formulation. It introduces the input being handled, the constraints under which generation operates and the operations the user may perform to build her query. In Section 4, we present the generation algorithm used to support the verbalisation of possible queries. Section 5 reports on an evaluation of the system with respect to fluency, clarity, coverage and incrementality. Section 6 concludes with pointers for further research.

2 Related Work

Our approach is related to two main strands of work: incremental generation and conceptual authoring.

Incremental Generation (Oh and Rudnicky, 2000) used an n-gram language model to stochas-

[1] krdbapp.inf.unibz.it:8080/quelo

Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 183–191,
Gothenburg, Sweden, April 26-30 2014.

tically generate system turns. The language model is trained on a dialog corpus manually annotated with word and utterance classes. The generation engine uses the appropriate language model for the utterance class and generates word sequences randomly according to the language model distribution. The generated word sequences are then ranked using a scoring mechanism and only the best-scored utterance is kept. The system is incremental is that each word class to be verbalised can yield a new set of utterance candidates. However it supports only addition not revisions. Moreover it requires domain specific training data and manual annotation while the approach we propose is unsupervised and generic to any ontology.

(Dethlefs et al., 2013) use Conditional Random Fields to find the best surface realisation from a semantic tree. They show that the resulting system is able to modify generation results on the fly when new or updated input is provided by the dialog manager. While their approach is fast to execute, it is limited to a restricted set of domain specific attributes; requires a training corpus of example sentences to define the space of possible surface realisations; and is based on a large set (800 rules) of domain specific rules extracted semi-automatically from the training corpus. In contrast, we use a general, small size grammar (around 50 rules) and a lexicon which is automatically derived from the input ontologies. The resulting system requires no training and thus can be applied to any ontology with any given signature of concepts and relations. Another difference between the two approaches concerns revisions: while our approach supports revisions anywhere in the input, the CRF approach proposed by (Dethlefs et al., 2013) only supports revisions occurring at the end of the generated string.

There is also much work (Schlangen and Skantze, 2009; Schlangen et al., 2009) in the domain of spoken dialog systems geared at modelling the incremental nature of dialog and in particular, at developing dialog systems where processing starts before the input is complete. In these approaches, the focus is on developing efficient architectures which support the timely interleaving of parsing and generation. Instead, our aim is to develop a principled approach to the incremental generation of a user query which supports revision and additions at arbitrary points of the query being built; generates natural sounding text; and maximally preserves the linear order of the query.

Conceptual authoring Our proposal is closely related to the conceptual authoring approach described in (Hallett et al., 2007). In this approach, a text generated from a knowledge base, describes in natural language the knowledge encoded so far, and the options for extending it. Starting with an initial very general query (e.g., all things), the user can formulate a query by choosing between these options. Similarly, (Franconi et al., 2010a; Franconi et al., 2011b; Franconi et al., 2010b; Franconi et al., 2011a) describes a conceptual authoring approach to querying semantic data where in addition , logical inference is used to semantically constrain the possible completions/revisions displayed to the user.

Our approach departs from this work in that it makes use of standard grammars and algorithms. While previous work was based on procedures and templates, we rely on a Feature-Based Tree Adjoining Grammar to capture the link between text and semantics required by conceptual authoring; and we adapt a chart based algorithm to support the addition, the revision and the substitution of input material. To avoid confusing the user, we additionally introduce a scoring function which helps preserve the linear order of the NL query. The generation system we present is in fact integrated in the Quelo interface developed by (Franconi et al., 2011a) and compared with their previous template-based approach.

3 Incremental Generation of Candidate Query Extensions

The generation task we address is the following. Given a knowledge base K, some initial formal query q and a focus point p in that query, the reasoning services supported by Quelo's query logic framework (see (Guagliardo, 2009)) will compute a set of new queries $rev(q)$ formed by adding, deleting and revising the current query q at point p. The task of the generator is then to produce a natural language sentence for each new formal query $q' \in rev(q)$ which results from this revision process. In other words, each time the user refines a query q to produce a new query q', the system computes all revisions $rev(q)$ of q' that are compatible with the underlying knowledge base using a reasoner. Each of these possible revisions is then input to the generator and the resulting revised NL queries are displayed to the user. In what follows,

we assume that formal queries are represented using Description Logics (Baader, 2003).

The following examples show a possible sequence of NL queries, their corresponding DL representation and the operations provided by Quelo that can be performed on a query (bold face is used to indicate the point in the query at which the next revision takes place). For instance, the query in (1c) results from adding the concept $Young$ to the query underlying (1b) at the point highlighted by **man**.

(1) a. I am looking for **something** (initial query)
$\top$

b. I am looking for **a man** (substitute concept)
Man

c. I am looking for a young **man** (add compatible concept)
$Man \sqcap Young$

d. I am looking for a young **man who is married to a person** (add relation)
$Man \sqcap Young \sqcap \exists isMarried.(Person)$

e. I am looking for a **young** married man (substitute selection)
$MarriedMan \sqcap Young$

f. I am looking for a married man (delete concept)
$MarriedMan$

4 Generating Queries

Generation of KB queries differs from standard natural language generation algorithms in two main ways. First it should support the revisions, deletions and additions required by incremental processing. Second, to avoid confusing the user, the revisions (modifications, extensions, deletions) performed by the user should have a minimal effect on the linear order of the NL query. That is the generator is not free to produce any NL variant verbalising the query but should produce a verbalisation that is linearly as close as possible, modulo the revision applied by the user, to the query before revisions. Thus for instance, given the DL query (2) and assuming a linearisation of that formula that matches the linear order it is presented in (see Section 4.2.1 below for a definition of the linearisation of DL formulae), sentence (2b) will be preferred over (2c).

(2) a. $Car \sqcap \exists runOn.(Diesel) \sqcap \exists equippedWith.(AirCond)$

b. A car which runs on Diesel and is equipped with air conditioning

c. A car which is equipped with air conditioning and runs on Diesel

In what follows, we describe the generation algorithm used to verbalise possible extensions of user queries as proposed by the Quelo tool. We start by introducing and motivating the underlying formal language supported by Quelo and the input to the generator. We then describe the overall architecture of our generator. Finally, we present the incremental surface realisation algorithm supporting the verbalisation of the possible query extensions.

4.1 The Input Language

Following (Franconi et al., 2010a; Franconi et al., 2011b; Franconi et al., 2010b; Franconi et al., 2011a) we assume a formal language for queries that targets the querying of various knowledge and data bases independent of their specification language. To this end, it uses a minimal query language $\mathcal{L}$ that is shared by most knowledge representation languages and is supported by Description Logic (DL) reasoners namely, the language of tree shaped conjunctive DL queries. Let $\mathcal{R}$ be a set of relations and $\mathcal{C}$ be a set of concepts, then the language of tree-shaped conjunctive DL queries is defined as follows: $S ::= C \mid \exists R.(S) \mid S \sqcap S$ where $R \in \mathcal{R}$, $C \in \mathcal{C}$, $\sqcap$ denotes conjunction and $\exists$ is the existential quantifier.

A tree shaped conjunctive DL query can be represented as a tree where nodes are associated with a set of concept names (*node labels*) and edges are labelled with a relation name (*edge labels*). Figure 1 shows some example query trees.

4.2 NLG architecture

Our generator takes as input two $\mathcal{L}$ formula: the formula representing the current query q and the formula representing a possible revision r (addition/deletion/modification) of q. Given this input, the system architecture follows a traditional pipeline sequencing a document planner which (i) linearises the input query and (ii) partition the input into sentence size chunks; a surface realiser mapping each sentence size $\mathcal{L}$ formula into a sentence; and a referring expression generator verbalising NPs.

4.2.1 Document Planning

The document planning module linearises the input query and segments the resulting linearised

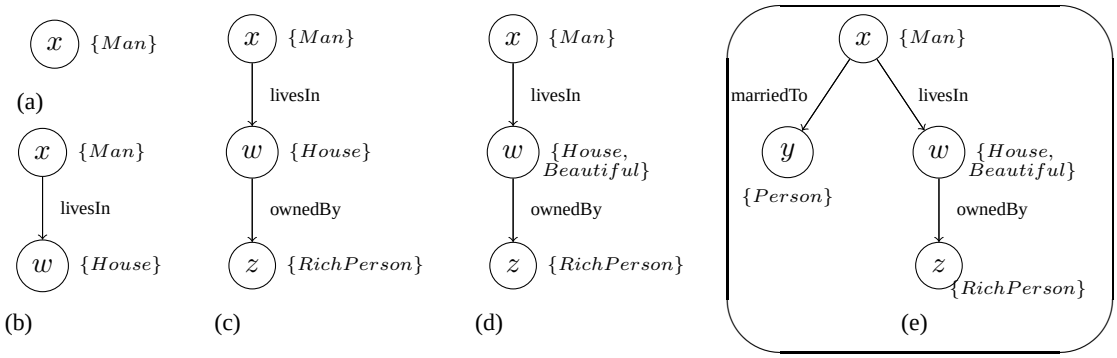


Figure 1: Example of query tree and incremental query construction.

query into sentence size chunks.

Query Linearisation Among the different strategies investigated in (Dongilli, 2008) to find a good order for the content contained in a query tree the *depth-first planning*, i.e. depth-first traversal of the query tree, was found to be the most appropriate one. Partly because it is obtained straightforward from the query tree but mostly due to the fact that it minimizes the changes in the text plan that are required by incremental query modifications. Thus, (Franconi et al., 2010a) defines a *query linearisation* as a strict total order[2] on the query tree that satisfies the following conditions:

- all labels associated with the edge's leaving node precede the edge label
- the edge label is followed by at least one label associated with the edge's arriving node
- between any two labels of a node there can only be (distinct) labels of the same node

The specific linearisation adopted in Quelo is defined by the depth-first traversal strategy of the query tree and a total order on the children which is based on the query operations. That is, the labels of a node are ordered according to the sequence applications of the `add compatible concept` operation. The children of a node are inversely ordered according to the sequence of applications of the `add relation` operation.

According to this linearisation definition, for the query tree (e) in Figure 1 the following linear order is produced:

(3) a. Man marriedTo $Person$ livesIn $House$ $Beautiful$ ownedBy $RichPeron$

[2] A strict total order can be obtained by fixing an order in the children nodes and traversing the tree according to some tree traversal strategy.

Query Segmentation Given a linearised query q, the document planner uses some heuristics based on the number and the types of relations/concepts present in q to output a sequence of sub-formulae each of which will be verbalised as a sentence.

4.2.2 Incremental Surface Realisation and Linearisation Constraints

We now describe the main module of the generator namely the surface realiser which supports both the incremental refinement of a query and a minimal modification of the linear order between increments. This surface realiser is caracterised by the following three main features.

Grammar-Based We use a symbolic, grammar-based approach rather than a statistical one for two reasons. First, there is no training corpus available that would consist of knowledge base queries and their increments. Second, the approach must be portable and should apply to any knowledge base independent of the domain it covers and independent of the presence of a training corpus. By combining a lexicon automatically extracted from the ontology with a small grammar tailored to produce natural sounding queries, we provide a generator which can effectively apply to any ontology without requiring the construction of a training corpus.

Chart-Based A chart-based architecture enhances efficiency by avoiding the recomputation of intermediate structures while allowing for a natural implementation of the revisions (addition, deletion, substitution) operations required by the incremental formulation of user queries. We show how the chart can be used to implement these operations.

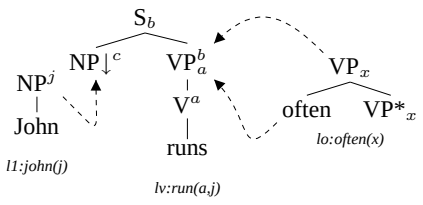
Beam search. As already mentioned, for ergonomic reasons, the linear order of the generated NL query should be minimally disturbed during query formulation. The generation system

should also be sufficiently fast to support a timely Man/Machine interaction. We use beam search and a customised scoring function both to preserve linear order and to support efficiency.

We now introduce each of these components in more details.

Feature-Based Tree Adjoining Grammar A tree adjoining grammar (TAG) is a tuple $\langle \Sigma, N, I, A, S \rangle$ with Σ a set of terminals, N a set of non-terminals, I a finite set of initial trees, A a finite set of auxiliary trees, and S a distinguished non-terminal ($S \in N$). Initial trees are trees whose leaves are labeled with substitution nodes (marked with a down-arrow) or with terminal categories[3]. Auxiliary trees are distinguished by a foot node (marked with a star) whose category must be the same as that of the root node.

Two tree-composition operations are used to combine trees: substitution and adjunction. Substitution inserts a tree onto a substitution node of some other tree while adjunction inserts an auxiliary tree into a tree. In a Feature-Based Lexicalised TAG (FB-LTAG), tree nodes are furthermore decorated with two feature structures which are unified during derivation; and each tree is anchored with a lexical item. Figure 2 shows an example toy FB-LTAG with unification semantics. The dotted arrows indicate possible tree combinations (substitution for *John*, adjunction for *often*). As the trees are combined, the semantics is the union of their semantics modulo unification. Thus given the grammar and the derivation shown, the semantics of *John often runs* is as shown namely, *named(j john), run(a,j), often(a)*.



l1:named(j john), lv:run(a,j), lv:often(a)

Figure 2: Derivation and Semantics for "John often runs"

Chart-Based Surface Realisation Given an FB-LTAG G of the type described above, sentences can be generated from semantic formulae by (i) selecting all trees in G whose semantics subsumes part of the input formula and (ii) combining these trees using the FB-LTAG combining operations namely substitution and adjunction. Thus for instance, in Figure 2, given the semantics *l1:named(j john), lv:run(a,j), lv:often(a)*, the three trees shown are selected. When combined they produce a complete phrase structure tree whose yield (*John runs often*) is the generated sentence.

Following (Gardent and Perez-Beltrachini, 2011), we implement an Earley style generation algorithm for FB-LTAG which makes use of the fact that the derivation trees of an FB-LTAG are context free and that an FB-LTAG can be converted to a a Feature-Based Regular Tree Grammar (FB-RTG) describing the derivation trees of this FB-LTAG[4].

On the one hand, this Earley algorithm enhances efficiency in that (i) it avoids recomputing intermediate structures by storing them and (ii) it packs locally equivalent structures into a single representative (the most general one). Locally equivalent structures are taken to be partial derivation trees with identical semantic coverage and similar combinatorics (same number and type of substitution and adjunction requirements).

On the other hand, it naturally supports the range of revisions required for the incremental formulation of ontology-based queries. Let C be the current chart i.e., the chart built when generating a NL query from the formal query. Then additions, revisions and deletion can be handled as follows.

- Add concept or property X: the grammar units selected by X are added to the agenda[5] and tried for combinations with the elements of C.
- Substitute selection X with Y: all chart items derived from a grammar unit selected by an element of X are removed from the chart. Conversely, all chart items derived from a grammar unit selected by an element of Y are added to the agenda. All items in the agenda are then processed until generation halts.
- Delete selection X: all chart items derived from a grammar unit selected by an element of X are removed from the chart. Intermediate structures that had previously used X are moved to the agenda and the agenda is processed until generation halts.

[3]For a more detailed introduction to TAG and FB-LTAG, see (Vijay-Shanker and Joshi, 1988).

[4]For more details on this algorithm, we refer the reader to (Gardent and Perez-Beltrachini, 2010).

[5]The agenda is a book keeping device which stores all items that needs to be processed i.e., which need to be tried for combination with elements in the chart.

Beam Search To enhance efficiency and favor those structures which best preserve the word order while covering maximal input, we base our beam search on a scoring function combining linear order and semantic coverage information. This works as follows. First, we associate each literal in the input query with its positional information e.g.,

(4) a. `man(x)[0] marriedTo(x y)[1] person(y)[2] livesIn(x w)[3] house(w)[4]`

This positional information is copied over to each FB-LTAG tree selected by a given literal and is then used to compute a *word order cost* (C_{wo}) for each derived tree as follows:

$$C_{wo}(t_{i+j}) = C_{wo}(t_i) + C_{wo}(t_j) + C_{wo}(t_i + t_j)$$

That is the cost of a tree t_{i+j} obtained by combining t_i and t_j is the sum of the cost of each of these trees plus the cost incurred by combining these two trees. We define this latter cost to be proportional to the distance separating the actual position (ap_i) of the tree (t_i) being substituted/adjoined in from its required position (rp_i). If t_i is substituted/adjoined at position n to the right (left) of the anchor of a tree t_j with position p_j, then the actual position of t_i is $pj + n$ ($pj - n$) and the cost of combining t_i with t_j is $\mid pj + n - rp_i \mid /\alpha$ ($\mid pj - n - rp_i \mid /\alpha$) where we empirically determined α to be 100[6].

Finally, the total score of a tree reflects the relation between the cost of the built tree, i.e. its word order cost, and its semantic coverage, i.e. nb. of literals from the input semantics:

$$S(t_i) = \begin{cases} -(|\text{literals}| - 1) & C_{wo}(t_i) = 0 \\ C_{wo}(t_i)/(|\text{literals}| - 1) & \text{otherwise} \end{cases}$$

The total score is defined by cases. Those trees with $C_{wo} = 0$ get a negative value according to their input coverage (i.e. those that cover a larger subset of the input semantics are favored as the trees in the agenda are ordered by increasing total score). Conversely, those trees with $C_{wo} > 0$ get a score that is the word order cost proportional to the covered input.

In effect, this scoring mechanism favors trees with low word order cost and large semantic coverage. The beam search will select those trees with lowest score.

[6]In the current implementation we assume that $n = 1$. Furthermore, as t_i might be a derived tree we also add to $C_{wo}(t_i + t_j)$ the cost computed on each tree t_k used in the derivation of t_i with respect to t_j.

4.2.3 Referring Expression Generation

The referring expression (RE) module takes as input the sequence of phrase structure trees output by the surface realiser and uses heuristics to decide for each NP whether it should be verbalised as a pronoun, a definite or an indefinite NP. These heuristics are based on the linear order and morpho-syntactic information contained in the phrase structure trees of the generated sentences.

5 Experiments and evaluation

We conducted evaluation experiments designed to address the following questions:

- Does the scoring mechanism appropriately capture the ordering constraints on the generated queries ? That is, does it ensure that the generated queries respect the strict total order of the query tree linearisation ?
- Does our grammar based approach produce more fluent and less ambiguous NL query than the initial template based approach currently used by Quelo ?
- Does the automatic extraction of lexicons from ontology support generic coverage of arbitrary ontologies ?

We start by describing the grammar used. We then report on the results obtained for each of these evaluation points.

5.1 Grammar and Lexicon

We specify an FB-LTAG with unification semantics which covers a set of basic constructions used to formulate queries namely, active and passive transitive verbs, adjectives, prepositional phrases, relative and elliptical clauses, gerund and participle modifiers. The resulting grammar consists of 53 FB-LTAG pairs of syntactic trees and semantic schema.

To ensure the appropriate syntax/semantic interface, we make explicit the arguments of a relation using the variables associated with the nodes of the query tree. Thus for instance, given the rightmost query tree shown in Figure 1, the flat semantics input to surface realisation is {*Man(x), Person(y), House(w), Beautiful(w), RichPerson(z), marriedTo(x,y), livesIn(x,w), ownedBy(w,z)*}.

For each ontology, a lexicon mapping concepts and relations to FB-LTAG trees is automatically derived from the ontology using (Trevisan, 2010)'s approach. We specify for each experiment below, the size of the extracted lexicon.

5.2 Linearisation

In this first experiment, we manually examined whether the incremental algorithm we propose supports the generation of NL queries whose word order matches the linearisation of the input query tree.

We created four series of queries such that each serie is a sequence $q_1 \dots q_n$ where q_{i+1} is an increment of q_i. That is, q_{i+1} is derived from q_i by adding, removing or substituting to q_i a concept or a relation. The series were devised so as to encompass the whole range of possible operations at different points of the preceding query (e.g., at the last node/edge or on some node/edge occurring further to the left of the previous query); and include 14 revisions on 4 initial queries.

For all queries, the word order of the best NL query produced by the generator was found to match the linearisation of the DL query.

5.3 Fluency and Clarity

Following the so-called *consensus model* (Power and Third, 2010), the current, template based version of Quelo generates one clause per relation[7]. Thus for instance, template-based Quelo will generate (5a) while our grammar based approach supports the generation of arguably more fluent sentences such as (5b).

(5) a. I am looking for a car. Its make should be a Land Rover. The body style of the car should be an off-road car. The exterior color of the car should be beige.

b. I am looking for car whose make is a Land Rover, whose body style is an off-road car and whose exterior color is beige.

We ran two experiments designed to assess how fluency impacts users. The first experiment aims to assess how Quelo template based queries are perceived by the users in terms of clarity and fluency, the second aims to compare these template based queries with the queries produced by our grammar-based approach.

Assessing Quelo template-based queries Using the Quelo interface, we generated a set of 41 queries chosen to capture different combinations of concepts and relations. Eight persons (four native speakers of English, four with C2 level of competence for foreign learners of English) were then asked to classify (a binary choice) each query in terms of clarity and fluency. Following (Kow and Belz, 2012), we take *Fluency* to be a single quality criterion intended to capture language quality as distinct from its meaning, i.e. how well a piece of text reads. In contrast, *Clarity/ambiguity* refers to ease of understanding (Is the sentence easy to understand?). Taking the average of the majority vote, we found that the judges evaluated the queries as non fluent in 50% of the cases and as unclear in 10% of the cases. In other words, template based queries were found to be disfluent about half of the time and unclear to a lesser extent. The major observation made by most of the participants was that the generated text is too repetitive and lacks aggregation.

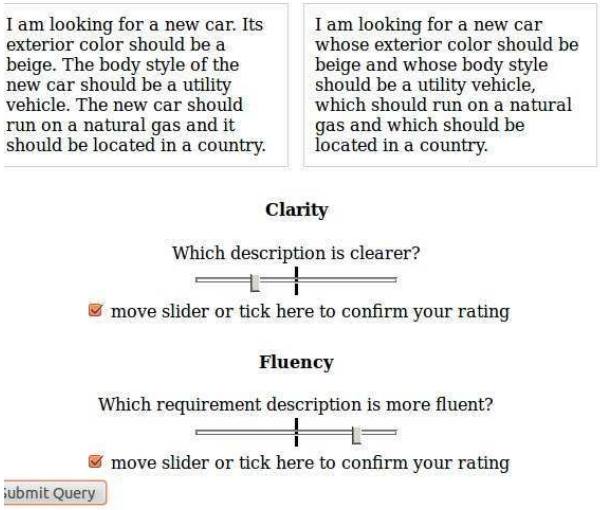


Figure 3: Online Evaluation.

Comparing template- and grammar-based queries In this second experiment, we asked 10 persons (all proficient in the English language) to compare pairs of NL queries where one query is produced using templates and the other using our grammar-based generation algorithm. The evaluation was done online using the LG-Eval toolkit (Kow and Belz, 2012) and geared to collect relative quality judgements using visual analogue scales. After logging in, judges were given a description of the task. The sentence pairs were displayed as shown in Figure 3 with one sentence to the left and the other to the right. The judges were instructed to move the slider to the left to favor the sentence shown on the left side of the screen; and to the right to favor the sentence appearing to the right. Not moving the slider means that both sentences rank equally. To avoid creating a bias,

[7] This is modulo aggregation of relations. Thus two subject sharing relations may be realised in the same clause.

the sentences from both systems were equally distributed to both sides of the screen.

For this experiment, we used 14 queries built from two ontologies, an ontology on cars and the other on universities. The extracted lexicons for each of these ontology contained 465 and 297 entries respectively.

The results indicate that the queries generated by the grammar based approach are perceived as more fluent than those produced by the template based approach (19.76 points in average for the grammar based approach against 7.20 for the template based approach). Furthermore, although the template based queries are perceived as clearer (8.57 for Quelo, 6.87 for our approach), the difference is not statistically significant ($p < 0.5$). Overall thus, the grammar based approach appears to produce verbalisations that are better accepted by the users. Concerning clarity, we observed that longer sentences let through by document planning were often deemed unclear. In future work, we plan to improve clarity by better integrating document planning and sentence realisation.

5.4 Coverage

One motivation for the symbolic based approach was the lack of training corpus and the need for portability: the query interface should be usable independently of the underlying ontology and of the existence of a training corpus. To support coverage, we combined the grammar based approach with a lexicon which is automatically extracted from the ontology using the methodology described in (Trevisan, 2010). When tested on a corpus of 200 ontologies, this approach was shown to be able to provide appropriate verbalisation templates for about 85% of the relation identifiers present in these ontologies. 12 000 relation identifiers were extracted from the 200 ontologies and 13 syntactic templates were found to be sufficient to verbalise these relation identifiers (see (Trevisan, 2010) for more details on this evaluation).

That is, in general, the extracted lexicons permit covering about 85% of the ontological data. In addition, we evaluated the coverage of our approach by running the generator on 40 queries generated from five distinct ontologies. The domains observed are cinema, wines, human abilities, disabilities, and assistive devices, e-commerce on the Web, and a fishery database for observations about an aquatic resource. The extracted lexicons contained in average 453 lexical entries and the coverage (proportion of DL queries for which the generator produced a NL query) was 87%.

Fuller coverage could be obtained by manually adding lexical entries, or by developing new ways of inducing lexical entries from ontologies (c.f. e.g. (Walter et al., 2013)).

6 Conclusion

Conceptual authoring (CA) allows the user to query a knowledge base without having any knowledge either of the formal representation language used to specify that knowledge base or of the content of the knowledge base. Although this approach builds on a tight integration between syntax and semantics and requires an efficient processing of revisions, existing CA tools predominantly make use of ad hoc generation algorithms and restricted computational grammars (e.g., Definite Clause Grammars or templates). In this paper, we have shown that FB-LTAG and chart based surface realisation provide a natural framework in which to implement conceptual authoring. In particular, we show that the chart based approach naturally supports the definition of an incremental algorithm for query verbalisation; and that the added fluency provided by the grammar based approach potentially provides for query interfaces that are better accepted by the human evaluators.

In the future, we would like to investigate the interaction between context, document structuring and surface realisation. In our experiments we found out that this interaction strongly impacts fluency whereby for instance, a complex sentence might be perceived as more fluent than several clauses but a too long sentence will be perceived as difficult to read (non fluent). Using data that can now be collected using our grammar based approach to query verbalisation and generalising over FB-LTAG tree names rather than lemmas or POS tags, we plan to explore how e.g., Conditional Random Fields can be used to model these interactions.

Acknowledgments

We would like to thank Marco Trevisan, Paolo Guagliardo and Alexandre Denis for facilitating the access to the libraries they developed and to Natalia Korchagina and the judges who participated in the evaluation experiments.

References

Franz Baader. 2003. *The description logic handbook: theory, implementation, and applications*. Cambridge university press.

Nina Dethlefs, Helen Hastie, Heriberto Cuayáhuitl, and Oliver Lemon. 2013. Conditional Random Fields for Responsive Surface Realisation using Global Features. *Proceedings of ACL, Sofia, Bulgaria*.

Paolo Dongilli. 2008. Natural language rendering of a conjunctive query. *KRDB Research Centre Technical Report No. KRDB08-3). Bozen, IT: Free University of Bozen-Bolzano*, 2:5.

E. Franconi, P. Guagliardo, and M. Trevisan. 2010a. An intelligent query interface based on ontology navigation. In *Workshop on Visual Interfaces to the Social and Semantic Web, VISSW*, volume 10. Citeseer.

E. Franconi, P. Guagliardo, and M. Trevisan. 2010b. Quelo: a NL-based intelligent query interface. In *Pre-Proceedings of the Second Workshop on Controlled Natural Languages*, volume 622.

E. Franconi, P. Guagliardo, S. Tessaris, and M. Trevisan. 2011a. A natural language ontology-driven query interface. In *9th International Conference on Terminology and Artificial Intelligence*, page 43.

E. Franconi, P. Guagliardo, M. Trevisan, and S. Tessaris. 2011b. Quelo: an Ontology-Driven Query Interface. In *Description Logics*.

C. Gardent and L. Perez-Beltrachini. 2010. RTG based Surface Realisation for TAG. In *COLING'10*, Beijing, China.

B. Gottesman Gardent, C. and L. Perez-Beltrachini. 2011. Using regular tree grammar to enhance surface realisation. *Natural Language Engineering*, 17:185–201. Special Issue on Finite State Methods and Models in Natural Language Processing.

Paolo Guagliardo. 2009. Theoretical foundations of an ontology-based visual tool for query formulation support. Technical report, KRDB Research Centre, Free University of Bozen-Bolzano, October.

C. Hallett, D. Scott, and R. Power. 2007. Composing questions through conceptual authoring. *Computational Linguistics*, 33(1):105–133.

Eric Kow and Anja Belz. 2012. LG-Eval: A Toolkit for Creating Online Language Evaluation Experiments. In *LREC*, pages 4033–4037.

Alice H Oh and Alexander I Rudnicky. 2000. Stochastic language generation for spoken dialogue systems. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational systems-Volume 3*, pages 27–32. Association for Computational Linguistics.

R. Power and A. Third. 2010. Expressing owl axioms by english sentences: dubious in theory, feasible in practice. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1006–1013. Association for Computational Linguistics.

David Schlangen and Gabriel Skantze. 2009. A general, abstract model of incremental dialogue processing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 710–718. Association for Computational Linguistics.

David Schlangen, Timo Baumann, and Michaela Atterer. 2009. Incremental reference resolution: The task, metrics for evaluation, and a bayesian filtering model that is sensitive to disfluencies. In *Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 30–37. Association for Computational Linguistics.

H. R Tennant, K. M Ross, R. M Saenz, C. W Thompson, and J. R Miller. 1983. Menu-based natural language understanding. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 151–158. Association for Computational Linguistics.

Marco Trevisan. 2010. *A Portable Menuguided Natural Language Interface to Knowledge Bases for Querytool*. Ph.D. thesis, Masters thesis, Free University of Bozen-Bolzano (Italy) and University of Groningen (Netherlands).

K. Vijay-Shanker and A. Joshi. 1988. Feature based tags. In *Proceedings of the 12th International Conference of the Association for Computational Linguistics*, pages 573–577, Budapest.

Holger Wache, Thomas Voegele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hübner. 2001. Ontology-based integration of information-a survey of existing approaches. In *IJCAI-01 workshop: ontologies and information sharing*, volume 2001, pages 108–117. Citeseer.

Sebastian Walter, Christina Unger, and Philipp Cimiano. 2013. A corpus-based approach for the induction of ontology lexica. In *Natural Language Processing and Information Systems*, pages 102–113. Springer.