An Intelligent Query Interface with Natural Language Support

Paolo Dongilli and **Enrico Franconi**
KRDB Centre, Faculty of Computer Science, Free University of Bozen-Bolzano
Dominikanerplatz 3, 39100 Bozen, Italy
lastname@inf.unibz.it

Abstract

The project described by the present paper aims at building a bridge between Intelligent Query Interfaces and Natural Language Generation technologies. The idea is to have a query interface enabling the users to access heterogeneous data sources by means of an integrated ontology. This paper shows how we are redesigning our intelligent query interface by rendering the logic-based queries in natural language, leveraging the results achieved to-date by applied Systemic-Functional Linguistics.

Introduction

The project described by the present paper aims at building a bridge between Intelligent Query Interfaces and Natural Language Generation technologies. The idea is to have a query interface enabling the users to access heterogeneous data sources by means of an integrated ontology. The query interface supports the users in formulating a precise conjunctive query where the intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The main challenge is that the underlying conjunctive query must be presented to the user in natural language and the stepwise refinements of the query made by the user are refinements that maintain the grammaticality of the sentences representing the query. This can be considered a step towards an assisted question answering system where the question is not freely inserted by the user, but instead the user is guided in building it according to the entities found in the knowledge domain chosen.

This paper is organized in two main parts: first it gives a close look to our work in terms of intelligent query interfaces, then it shows how we are enhancing the system we devised, by adding natural language support to query formulation.

Overview of an Intelligent Query Interface

In general terms, query interfaces are meant to support users in formulating a precise query against a database described by a specific *data model*. Queries are specified by means of special purpose *query languages*, where a query language is a set of formally defined operators allowing requests to be expressed to a database. By executing a query, the user expects that the produced results extracted from the stored data are coherent with the intended meaning of the request. The most widely used database query languages have been programming languages which require knowledge about language syntax, technical background, and information of both the system application domain and its interaction mechanisms. Such languages do not help to understand the meaning of data, nor do they provide any guidance in satisfying the user's needs. In general, they do not fulfil the requirements of user friendliness and ease of use (Catarci *et al.* 1997).

Our recent contribution to the design of an intelligent query interface (Dongilli, Franconi, & Tessaris 2004; Catarci *et al.* 2004) dealt with a relatively new problem, namely providing the user with a visual interface to query heterogeneous data sources through an integrated ontology; only little work has been done in this specific context (see, e.g., the extensive survey on Visual Query Systems (VQS) (Catarci *et al.* 1997). Our proposal stems from the preliminary work in (Bresciani & Franconi 1996; Franconi 2000; Bresciani, Nori, & Pedot 2000; Bresciani & Fontana 2002). Similar work from the point of view of the visual interface paradigm, but without the well founded support of a logic-based semantics was carried out in (Proper 1994) and in the context of the TAMBIS project (Murray, Goble, & Paton 1998; Bechhofer *et al.* 1999). (Benzi, Maio, & Rizzi 1999) contains an interesting approach from the point of view of the visual interface, but the system does not consider a logic-based semantics. In fact, only recently research has started to have a serious interest in query processing and information access supported by (logic-based) ontologies. Recent work has come up with proper semantics and with advanced reasoning techniques for query evaluation and rewriting using views under the constraints given by the ontology – also called view-based query processing (Ullman 1997; Calvanese, Giacomo, & Lenzerini 2000). This means that the notion of accessing information through the navigation of an ontology modelling the information domain has its formal foundations.

A closer look will be given now to the first intelligent query interface we designed, which will be referred to as Query Interface (QI) hereinafter. First the user perspective will be analyzed, then we will delve into the internals of the QI explaining what conjunctive queries are, and how reasoning services are employed

The User Perspective

Initially the user is presented with a choice of different query scenarios which provide a meaningful starting point for the

query construction. In other terms the user can choose the domain (ontology) where she wants to query on. The interface guides then the user in the construction of a query by means of a diagrammatic interface, which enables the generation of precise and unambiguous query expressions.

Query expressions are compositional, and their logical structure is not flat but tree shaped; i.e. a node with an arbitrary number of branches connecting to other nodes. A query is composed by a list of terms (classes) coming from the ontology; e.g. “off-roader” or “car dealer”. Branches are constituted by properties (relations or attributes) with their value restriction which is a query expression itself. Referring to the underlying ontology, a *relation* is an association between a concept and another generic concept, while an *attribute* is an association between a concept and a simple data-type class (String, Boolean, Integer,...) or a concept subsumed by a simple data-type class.

For instance, in “car sold by car dealer”, “sold by” is a relation, “car” and “car dealer” are ontology terms (see figure 1). “Land Rover” is the value (or restriction) of the concept “model” which is subsumed by the simple data-type class “String”.

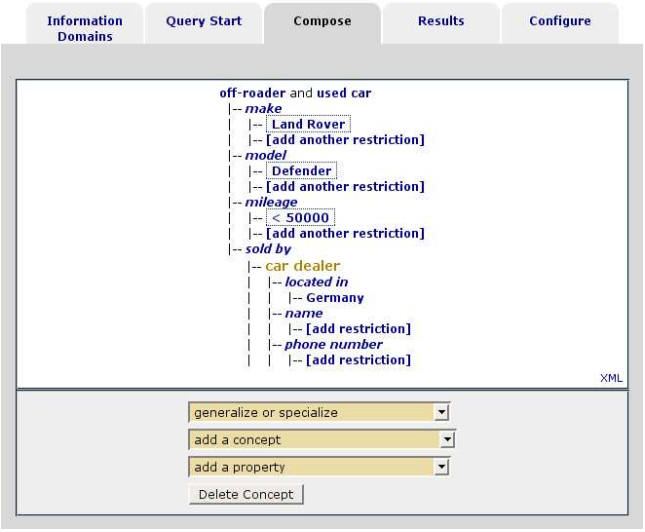

Figure 1: Query example (in the automotive domain) with focus set on the concept “car dealer”.

The manipulation of the query is always restricted to a well defined, and visually delimited, subpart of the whole query called *focus*. The compositional nature of the query language induces a natural navigation mechanism for moving the focus across the query expression (nodes of the corresponding tree). A constant feedback of the focus is provided on the interface by means of the kind of operations which are allowed. The system suggests only the operations which are *compatible* with the current query expression, in the sense that do not cause the query to be unsatisfiable (see paragraph “Reasoning Services and Query Interface” below). This is verified against the formal model describing the data sources.

The QI is divided into two panes: the upper one (*query display pane*) representing the tree-shaped query composed by the user, and the bottom one which is the *query manipulation pane*.

After focusing on an ontology term in the query display pane, the user can perform the following operations in the manipulation pane: (i) generalize or specialize the term, (ii) add a compatible concept, (iii) substitute term with an equivalent one, (iv) add a property (relation or attribute), (v) delete the focused concept.

With the first operation the user can choose among a list of more general or specific concepts where the selected one will substitute the focused term; e.g. in figure 1 the concept “off-roader” could be focused, substituted with the more general concept “car” and specialized again into “sedan”.

In the refinement by compatible terms (ii), the selected term is simply added to the focus as unary query term. The system driven by the reasoner suggests terms from the ontology whose overlap with the focus can be non-empty (the *compatibility* requirement). In the example, “used car” was among the compatible terms for the concept “off-roader”, and was added by the user. The concept “new car” was also among the compatibles, but now it is no more visible because of the disjointness with “used car”.

It can be the case (not happening for the focus in figure 1) that there are terms in the ontology which are equivalent to the logical expression represented by the query with the focus acting as root. In this case the user is offered to replace the whole subtree with the equivalent term by the activation of the “Replace Equivalent” button.

The property extension (iv) enables the user to add attributes (e.g. “car dealer *with e-mail* e-mail_address”) or relations (e.g. “car dealer *located in* country”), and these actions correspond to the creation of a new branch of the query tree.

Finally (iv), the deletion of the focused term has as consequence the deletion of the incoming property connected to it.

A First Glance at the System Internals

Conjunctive Queries Since the interface is built around the concept of classes and their properties, we consider conjunctive queries composed by unary (classes) and binary (attributes and relations) terms. The body of a query can be considered as a graph as e.g. the one shown in figure 2 in which variables (and constants) are nodes, and binary terms are edges. A query is connected (or acyclic) if the corresponding graph is acyclic. Given the form of query expressions composed by the interface introduced above, we restrict ourselves to acyclic connected queries. This restriction has been adopted because, up till now, we did not find any graphical representation for cyclic queries intuitive enough to be easily understandable and usable by a non-trained user. Note that the query language restrictions do not affect the ontology language, where the terms are defined by a different (in our case more expressive) language. The complexity of the ontology language is left completely hidden to the user, who doesn’t need to know anything about it.

To transform any query expression in a conjunctive query we proceed in a recursive fashion starting from the top level, and transforming each branch. A new variable is associated to each node: the list of ontology terms corresponds to the list of unary terms. For each branch, the binary query term corresponding to the property is then added, and its restriction is recursively expanded in the same way.

Let us consider a part of the query of figure 2: “*Find off-roader, make Land Rover, model Defender, sold by car dealer located in Germany*”.

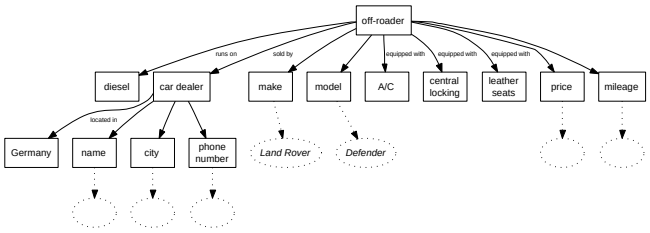

Figure 2: Graphical representation of a query.

Firstly, a new variable (x_1) is associated to the top level "off-roader". Assuming that the top level variable is by default part of the distinguished variables, the conjunctive query becomes

$$\{x_1|\text{off-roader}(x_1), ...\},$$

where the dots mean that there is still part of the query to be expanded. Then we consider the property "sold by", with its value restriction "car dealer": this introduces a new variable $x_{1,1}$. The remaining properties of the concept "car dealer" are then similarly expanded, generating the conjunctive query

$$\begin{aligned}\{x_1|\ &\text{off-roader}(x_1), \text{sold\_by}(x_1, x_{1,1}), \text{car\_dealer}(x_{1,1}),\\ &\text{located\_in}(x_{1,1}, x_{1,1,1}), \text{Germany}(x_{1,1,1}),\\ &\text{has\_make}(x_1, x_{1,2}), x_{1,2} \in \{LandRover\},\\ &\text{has\_model}(x_1, x_{1,3}), x_{1,3} \in \{Defender\}\} \quad (1)\end{aligned}$$

This transformation is bidirectional, so that a connected acyclic conjunctive query can be represented as a query expression by dropping the variable names. As a matter of fact, the system is using this inverse transformation since the internal representation of queries is as conjunctive queries.

Reasoning Services and Query Interface Reasoning services w.r.t. the ontology are used by the system to drive the query interface. In particular, they are used to discover the terms and properties (with the related concepts) which are proposed to the user to manipulate the query. The main idea is to be as less restrictive as possible on the requirements for the ontology language. In this way, the same technology can be adopted for different frameworks, while the user is never exposed to the complexity (and peculiarities) of a particular ontology language.

In our context, an ontology is composed by a set of predicates (unary, binary), together with a set of constraints restricting the set of valid interpretations (i.e. databases) for the predicates. The kind of constraints which can be expressed defines the expressiveness of the ontology language. Note that these assumptions are general enough to take account of widely used modelling formalisms, like UML for example.

We do not impose general restrictions on the expressiveness of the ontology language; however, we require the availability of two *decidable reasoning services*: *satisfiability* of a conjunctive query, and *containment test* of two conjunctive queries, both w.r.t. the constraints. If the query language includes the empty query (i.e. a query whose extension is always empty), then query containment is enough (a query is satisfiable if and only if it is not contained in the empty query).

Although our approach is not tight to any ontology language, in the test implementation of our system we are using Description Logics (DLs). The reasons for this choice lie in the facts that DLs can capture a wide range of widespread modelling frameworks, and the availability of efficient and complete DL reasoners.

We adopted the Description Logics $\mathcal{SHIQ}$ (see (Horrocks & Sattler 2002)) which is expressive enough for our purposes, and for which there are state of the art reasoners. Note that the adoption of $\mathcal{SHIQ}$ allow us to use ontologies written in standard web ontology languages like OWL–DL (Horrocks & Patel-Schneider 2003).

Our implementation uses the DL reasoner Racer (Haarslev & Möller 2001), which fully supports the $\mathcal{SHIQ}$ DL. The interaction with the DL reasoner is based on the DIG 1.0 interface API (Bechhofer, Möller, & Crowther 2003), a standard to communicate with DL reasoners developed among different DL systems implementors. This choice makes our system independent from a particular DL reasoner, which can be any DIG based one (as e.g. FACT (FaCT 2005), RACER (Racer 2005) or PELLET (Pellet 2005)).

Adding Natural Language Support

As described above, the strength of the intelligent QI we designed and developed at the KRDB Centre of the Free University of Bozen-Bolzano lays behind the solid formal foundations characterizing it in terms of information access through the navigation of an ontology.

Usability evaluation tests accompanying the user-centered design methodology we adopted, highlighted the user satisfaction after achieving the specific query-composition tasks assigned (Catarci *et al.* 2005). We also noticed that the amount of time spent to construct queries is independent of the domain expertise of the user, confirming that the QI is usable by both domain-experts and non-domain-expert users.

On the other side, though, we intended to enhance the system from the man-machine interaction viewpoint. Instead of a tree-shaped query the intention is to have it expressed in natural language in order to increase the level of user friendliness of the QI. This does not mean that we intend to allow the user to ask any natural language question but that the QI, given the domain of interest, guides the user during the stepwise creation of a NL query within the boundaries of the chosen domain.

The New User Perspective

The new Query Interface (QI) which is under development, is divided as before into two main areas as shown in the screenshot of figure 3.

The upper area embeds the query represented as one or more NL sentences. The concept selected by the user (*off-roader*) is the current focus; it is marked in every sentence in which it appears, clearly also when it is in pronominal form. The user can select only concepts and restrictions, since we noticed that it wasn't natural at all for the user to interact with properties within the query area; furthermore the only action permitted was the deletion of the property, operation that is now accomplished by deleting the concept representing the range of the same property.

The bottom area, instead, shows all operations that can be done on the focused concept: it is possible to delete it (deleting the whole subtree and the incoming property), choose

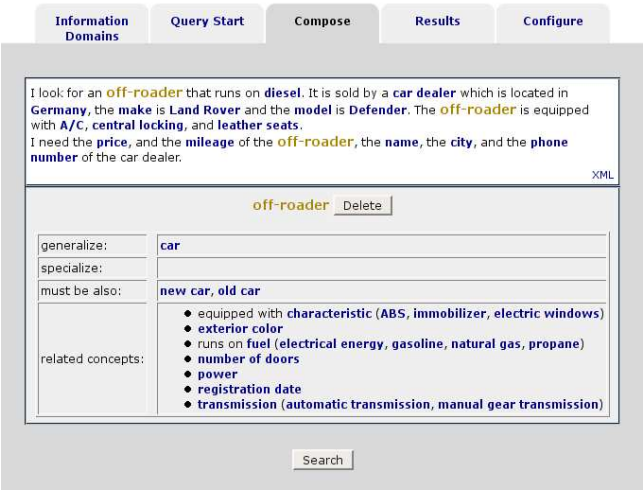

Figure 3: Query example in the new QI, with focus set on the concept "off-roader".

a generalization, a specialization, or add a compatible concept. Related concepts can be chosen: here, in order to reduce the quantity of information shown to the user, the name of attributes (usually *has...*) are hidden and only the range concepts are visible (e.g. *exterior color, number of doors, power...*). On the contrary, relations between concepts are shown (*equipped with..., runs on...*).

Every action done in the bottom area reflects on the whole NL query which is regenerated again: noun phrases representing concepts are changed (generalization or specialization of the focus), coordinate clauses are added (when selecting compatible concepts), other coordinate/subordinate clauses are added or new sentences written (when related concepts are chosen).

We move behind the new interface now, to see how the system is being extended.

Inside the New System

The Query Interface (QI) (figure 4) exists in two versions: a web application and a stand-alone Java program. The ontology-based query construction mechanism relies upon reasoning services accessible through a DIG protocol (Bechhofer, Möller, & Crowther 2003) implemented by a DIG API we provided. There are some other APIs we developed, as e.g. the one for the communication with software agents to whom the query-answering process is demanded. This won't be described here, to leave space for the description of the query composition infrastructure.

The extensions required to add natural language flavor to the QI, as described in the previous section, consist mainly of two modules: a natural language generator and a conversion tool that given a conjunctive query and optionally a user profile, generates a sentence plan in SPL (Sentence Plan Language), required as input by the NL generator. The following two paragraphs will be devoted to the description of these modules and the resources they need to operate.

Natural Language Generator As generation environment we are using KPML (Komet-Penman multilingual), a grammar development environment from the University of

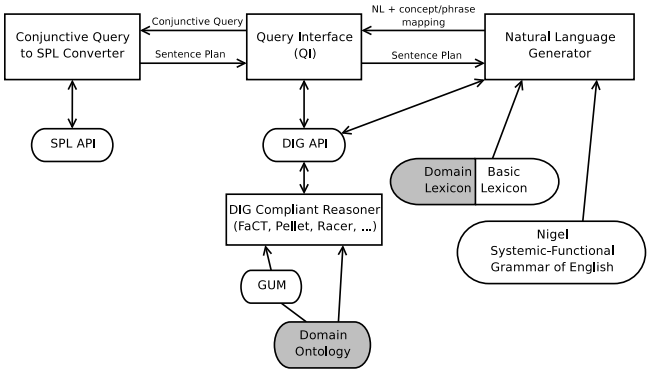

Figure 4: Internals of the new system with NL support.

Bremen (Bateman 1997). KPML is a complex application, well known for extensive multilingual systemic-functional grammar (SFG) editing and NL generation. In particular we are testing its generation module coupled with the latest version of the Nigel grammar for English and the Penman Upper Model (UM), a general task- and domain-independent linguistically motivated ontology used for mediating between domain knowledge and the natural language generation (NLG) system. In terms of Systemic-Functional Linguistics (SFL) (Halliday & Matthiessen 2004), the UM reflects the *ideational metafunction*; this is why it is often called the *ideation base*.

The Nigel grammar has been under development since the early 1980s, when it was used within the Penman project for English generation. It was mainly developed by Christian Matthiessen on the foundation of work by Michael Halliday. Since then, many people have contributed to various parts of its coverage. References to the underlying linguistic grammatical description are described in (Halliday & Matthiessen 2004; Matthiessen 1995; Martin, Matthiessen, & Painter 1997).

The KPML generation engine, available in LISP only, is being rewritten (code-name: JENERATE) by our group in cooperation with the University of Bremen (Germany), using J2EE technologies, for a higher level of portability and integration with other applications, where our Query Interface is the first candidate. The resources loaded in JENERATE will be the same as in KPML with the exception of the UM that will be replaced by the Generalized Upper Model (GUM). GUM (Bateman, Magnini, & Fabris 1995) is a multilingual evolution of the Penman UM; its initial representation language was Loom, but now it's being ported to other knowledge representation languages as OWL. It will also be accessible as DIG file, and loaded by JENERATE through the DIG API.

Linguistic resources as the grammar and the basic lexicon will be the same, modulo a translation from LISP to XML according to the DTD/XSD proposed by the University of Bremen. In figure 4 the reader can notice also a *domain ontology* and a *domain lexicon*: these represent the only two resources that need to be changed whenever we want to switch from a knowledge domain and generate NL in another domain. A new lexicon of terms pertaining the chosen domain can be added as separate file from the basic lexicon. On the contrary, plugging in a new domain ontology needs some more effort, since it has to to be merged with the GUM.

The fact that we generate in English, doesn't mean that it's the only language available for generation. There are linguistic resources available (or 'almost' available) for Bulgarian, Chinese, Czech, Greek, Japanese, Russian, German, and Spanish.

The dynamic input of JENERATE that drives NL generation is the logical specification SPL (Sentence Plan Language), the interface between the application and the generator. SPL (Kasper 1989) incorporates *ideational*, *interpersonal* and *textual* specification, metafunctions specified in Systemic-Functional Linguistics (Halliday & Matthiessen 2004). The ideational content is bound to the GUM, in the sense that entities from the ideation base are mapped onto ideational grammatical functions (as e.g. `ACTOR` or `ACTEE` in the SPL excerpt below). On the other side interpersonal and textual items are specific responses to the inquiries which are raised by the generator towards the grammar.

If we want to generate the sentence '*The car must run on diesel, and the make must be Land Rover.*', the conjunctive query and the SPL expression would be:

$$\begin{aligned}\{x_1 |\ & \mathrm{car}(x_1),\\ & \text{run-on}(x_1, x_{1,1}),\\ & \text{diesel}(x_{1,1}),\\ & \text{make}(x_1, x_{1,2}),\\ & x_{1,2} \in \{LandRover\}\}\end{aligned} \quad (2)$$

```
((S1 S2)
  (S1 / run-on
      :MODALITY must
      :ACTOR (C1 / car)
      :ACTEE (C2 / diesel)
      )
  (S2 / PROPERTY-ASCRIPTION
      :MODALITY must
      :DOMAIN (C3 / make)
      :RANGE
      (C4 / QUALITY
          :LEX Land-Rover)))
```

where `S1` and `S2` represent two coordinate sentences, `car`, `diesel` and `make` are concepts, `Land Rover` is the property restriction, and `run-on` is a reified relation subsumed by `DISPOSITIVE-MATERIAL-ACTION` (see below). `S2` represents instead a `PROPERTY-ASCRIPTION`.

A small excerpt of the domain ontology (LOOM) is shown below:

```
(defconcept run-on
  :is (:and Penman-kb::DISPOSITIVE-MATERIAL-ACTION
       :primitive))
(penman::annotate-concept run-on :lex-items (run-on))

(defconcept vehicle
  :is (:and Penman-kb::Object :primitive))

(defconcept car
  :is (:and vehicle :primitive))
```

and a lexical entry of the domain lexicon:

```
(LEXICAL-ITEM
    :NAME   run-on
    :SPELLING   "run on"
    :SAMPLE-SENTENCE   "These cars run on unleaded gasoline."
    :FEATURES   (NOT-OBJECTNOTREQUIRED NOT-BAREINFINITIVECOMP
                 NOT-ADJECTIVECOMP NOT-COPULA NOT-MAKECOMP
                 NOT-TOCOMP NOT-QUESTIONCOMP NOT-SUBJECTCOMP
                 INDIRECTOBJECT DISPOSAL-VERB DO-VERB
                 EFFECTIVE-VERB PASSIVE OBJECTPERMITTED
                 NOT-PARTICIPLECOMP NOT-THATCOMP
                 NOT-CASEPREPOSITIONS LEXICAL EDPARTICIPLEFORM
                 PASTFORM S-IRR UNITARYSPELLING INFLECTABLE VERB)
    :PROPERTIES ((INGPARTICIPLEFORM "running on")
                 (THIRDSINGULARFORM "runs on")
                 (PASTFORM "ran on")
                 (EDPARTICIPLEFORM "run on")))
```

The output of the Natural Language Generator is not simply plain text: noun phrases within each clause need to be tagged with and mapped back to the concepts or value restrictions they refer to in the conjunctive query. This is necessary to pilot the query interface which must show the user which noun phrases can be clicked to change the focus, and which ones represent the current focus.

Conjunctive Query to SPL mapper The remaining module that composes the new system is devoted to the conversion of a given conjunctive query (hereinafter called CQ) into SPL. Considering that the highest-level rank in the Nigel grammar of English is the clause-complex, with a single SPL file it is possible to cover sentences handling relationships between main and subordinate clauses (*I look for an off-roader that runs on diesel*) or between coordinate clauses (*The make is Land Rover, and the model is Defender*).

The CQ needs to be partitioned into a certain number of chunks, where each one is mapped into an SPL file and later transformed into a sentence by the NL generator. We are not able yet to estimate how many possible ways there are, given a CQ, to partition it and create SPL files that once rendered as NL sentences build an acceptably fluent English representation of the query. Following a heuristic approach, we're starting to build a corpus of CQs associating each one with one possible verbalization. The queries will be sorted on the basis of a complexity model, similar to the one used in (Catarci *et al.* 2005) to validate the metrics used in measuring the system usability. In this context instead, the complexity of a CQ will help in estimating the number of CQ partitions and consequently SPL files that have to be created.

A possible mapping approach that works in most of the cases is to do a first coarse split of the CQ into three main parts: a first part where the root concept is introduced; the second part where the root concept is further described along with all the other concepts in the query; the third part where the concepts without restriction are listed and presented as information that the user is seeking. Referring to the graphical query representation of figure 2 and the resulting verbalization of figure 3, the root concept is *off-roader*, and the concepts without restriction are *name*, *city*, and *phone number* (related to the concept *car dealer*); *mileage* and *price* (referred to the concept *off-roader*).

The first chunk is modelled in SPL language as a `DISPOSITIVE-MATERIAL-ACTION`, where the system describes what the user is looking for (*off-roader*) using the first person singular (*I look for an off-roader*), verbalizing optionally the relation with another concept (*. . . that runs on diesel.*).

In the second part the root concept is further described: a number of maximum three related concepts are described in another SPL fragment using coordinate clauses, adding, if present, further relations of the newly introduced concepts. The creation of other SPL files goes on, visiting the remaining concepts of the tree. Our experiments have shown that a depth-first visit leads to a more fluent verbalization in most of the cases rather than a breadth-first approach.

The third and last part of the sentence plan production regards the concepts with no restrictions mentioned above, which can be included in groups of coordinate sentences of possibly three concepts each (e.g. *I need the price, and the mileage of the off-roader, the name, the city, and the phone number of the car dealer.*).

Conclusions

In this paper we presented the evolution of an intelligent query tool developed at the KRDB centre of the Free University of Bozen-Bolzano, tool which is now subject to major changes for the sake of a higher user friendliness. Conjunctive queries are being represented in natural language (En-

glish) by means of a NL generation engine using J2EE technologies and derived from the KPML system from the University of Bremen. Reasoning services using the DIG protocol will be used both for driving the user interface and also for accessing the domain model and the linguistic ontology GUM (Generalized Upper Model) during the NL generation phase. SPL (Sentence Plan Language) will be leveraged as interface between conjunctive queries and the generator. A critical task is represented by the module for the remapping of a conjunctive query into SPL. For this task, results from Rhetorical Structure Theory (RST) will be taken into consideration, in particular the constructive version of RST used for multisentence text generation (Hovy 1988).

References

Bateman, J. A.; Magnini, B.; and Fabris, G. 1995. The generalized upper model knowledge base: Organization and use. In Mars, N. J. I., ed., *Towards very large knowledge bases: knowledge building and knowle dge sharing*, 60–72. Amsterdam: IOS Press.

Bateman, J. A. 1997. *KPML Development Environment: multilingual linguistic resource development and sentence generation*. German National Center for Information Technology (GMD), Inst itute for integrated publication and information systems (IPSI), Darmstadt, Germany. (Rel. 1.1).

Bechhofer, S.; Stevens, R.; Ng, G.; Jacoby, A.; and Goble, C. A. 1999. Guiding the user: An ontology driven interface. In *UIDIS 1999*, 158–161.

Bechhofer, S.; Möller, R.; and Crowther, P. 2003. The dig description logic interface. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*.

Benzi, F.; Maio, D.; and Rizzi, S. 1999. VISIONARY: a viewpoint-based visual language for querying relational databases. *J. Vis. Lang. Comput.* 10(2):117–145.

Bresciani, P., and Fontana, P. 2002. A knowledge-based query system for biological databases. In *Proceedings of FQAS 2002*, volume 2522 of *Lecture Notes in Computer Science*, 86–89. Springer Verlag.

Bresciani, P., and Franconi, E. 1996. Description logics for information access. In *Proceedings of AI*IA Workshop on Access, Extraction and Integration of Knowledge*.

Bresciani, P.; Nori, M.; and Pedot, N. 2000. A knowledge based paradigm for querying databases. In *Database and Expert Systems Application*, volume 1873 of *Lecture Notes in Computer Science*, 794–804. Springer Verlag.

Calvanese, D.; Giacomo, G. D.; and Lenzerini, M. 2000. Answering queries using views over description logics knowledge bases. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI 2000)*.

Catarci, T.; Costabile, M. F.; Levialdi, S.; and Batini, C. 1997. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing* 8(2):215–260.

Catarci, T.; Dongilli, P.; Mascio, T. D.; Franconi, E.; Santucci, G.; and Tessaris, S. 2004. An Ontology Based Visual Tool for Query Formulation Support. In *Proceedings ECAI 2004: the 16th Biennial European Conference on Artificial Intelligence*.

Catarci, T.; Dongilli, P.; Mascio, T. D.; Franconi, E.; Santucci, G.; and Tessaris, S. 2005. Usability evaluation tests in the SeWAsIE (SEmantic Webs and AgentS in Integrated Economies) project. In *Proceedings of the 11th International Conference on Human-Computer Interaction (HCII 2005)*.

Dongilli, P.; Franconi, E.; and Tessaris, S. 2004. Semantics Driven Support for Query Formulation. In *Proceedings of the 2004 International Workshop on Description Logics (DL 2004)*, volume 104.

FaCT. 2005. The FaCT System Website. `http://www.cs.man.ac.uk/~horrocks/FaCT/`.

Franconi, E. 2000. Knowledge representation meets digital libraries. In *Proc. of the 1st DELOS (Network of Excellence on Digital Libraries) workshop on "Information Seeking, Searching and Querying in Digital Libraries"*.

Haarslev, V., and Möller, R. 2001. Racer system description. In *Automated Reasoning: First International Joint Conference, IJCAR 2001*, volume 2083 of *Lecture Notes in Computer Science*. Springer-Verlag Heidelberg.

Halliday, M. A. K., and Matthiessen, C. M. 2004. *An Introduction to Functional Grammar*. London: Edward Arnold, 3rd edition.

Horrocks, I., and Patel-Schneider, P. F. 2003. Reducing OWL entailment to description logic satisfiability. In Fensel, D.; Sycara, K.; and Mylopoulos, J., eds., *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, 17–29. Springer.

Horrocks, I., and Sattler, U. 2002. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, 277–281.

Hovy, E. H. 1988. Planning coherent multisentential texts. In *The Proceedings of the 26th. Annual Meeting of the Association of Computational Linguistics*, 163–169. Buffalo, New York: Association for Computational Linguistics.

Kasper, R. T. 1989. A flexible interface for linking applications to PENMAN's sentence generator. In *Proceedings of the DARPA Workshop on Speech and Natural Language*. Available from USC/Information Sciences Institute, Marina del Rey, CA.

Martin, J. R.; Matthiessen, C. M. I. M.; and Painter, C. 1997. *Working with Functional Grammar*. London: Arnold.

Matthiessen, C. M. I. M. 1995. *Lexicogrammatical cartography: English systems*. Tokyo, Taipei and Dallas: International Language Science Publishers.

Murray, N.; Goble, C. A.; and Paton, N. W. 1998. A framework for describing visual interfaces to databases. *Journal of Visual Languages and Computing* 9(4):429–456.

Pellet. 2005. Pellet OWL Reasoner Website. `http://www.mindswap.org/2003/pellet/index.shtml`.

Proper, H. A. 1994. Interactive Query Formulation using Query By Navigation. Asymetrix Research Report 94-4, Asymetrix Research Laboratory, University of Queensland, Brisbane, Australia.

Racer. 2005. Racer Systems Website. `http://www.racer-systems.com/index.phtml`.

Ullman, J. D. 1997. Information integration using logical views. In *Proc. of the 6th Int. Conf on Database Theory (ICDT'97)*, 19–40.