Semantics driven support for query formulation

Paolo Dongilli, Enrico Franconi, and Sergio Tessaris

Free University of Bozen-Bolzano, Italy
<*lastname*>@inf.unibz.it

Abstract

In this paper we describe the principles of the design and development of an intelligent query interface, done in the context of the SEWASIE (SEmantic Webs and AgentS in Integrated Economies) European IST project. The SEWASIE project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The query interface is meant to support a user in formulating a precise query – which best captures her/his information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The final purpose of the tool is to generate a conjunctive query ready to be executed by some evaluation engine associated to the information system.

1 Introduction

In this paper we describe the principles of the design and development of an intelligent query interface, done in the context of the SEWASIE (SEmantic Webs and AgentS in Integrated Economies) European IST project. The SEWASIE project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The query interface is meant to support a user in formulating a precise query – which best captures her/his information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The final purpose of the tool is to generate a conjunctive query (or a non nested Select-Project-Join SQL query) ready to be executed by some evaluation engine associated to the information system.

The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The ontology defines a vocabulary which is richer than the logical schema of the underlying data, and it is meant to be closer to the user's rich vocabulary. The user can exploit the ontology's vocabulary to formulate the query, and she/he is guided by such a richer vocabulary in order to understand how to express her/his information needs more precisely, given the knowledge of the system. This latter task – called *intensional navigation* – is the most innovative functional aspect of our proposal. Intensional navigation can help a less skilled user during the initial step of query formulation, thus overcoming problems related with the lack of schema comprehension and so enabling her/him to easily formulate meaningful queries. Queries can be specified through an iterative refinement process supported by the ontology through intensional navigation. The user may specify her/his request

This work has been partially supported by the EU projects Sewasie, KnowledgeWeb, and Interop.

using generic terms, refine some terms of the query or introduce new terms, and iterate the process. Moreover, users may explore and discover general information about the domain without querying the information system, giving instead an explicit meaning to a query and to its subparts through classification.

In the literature there are several approaches at providing intelligent visual query systems for relational or object oriented databases (see [10] for an extensive survey). However, to our knowledge, the work presented in this paper is among the first well-founded intelligent systems for query formulation support in the context of ontology-based query processing. The strength of our approach derives from the fact that the graphical and natural language representation of the queries is underpinned by a formal semantics provided by an ontology language. The use of an appropriate ontology language enables the system engineers to precisely describe the data sources, and their implicit data constraints, by means of a system global ontology (see [9]). The same ontology is leveraged by the query interface to support the user in the composition of the query, rather than relying on a less expressive logical schema. The underlying technology used by the query interface is based on the recent work on query containment under constraints (see [8; 16]).

The paper is organised as follows. Firstly we present the system w.r.t. user viewpoint, with the functionalities of the interface, then we describe the semantics and the reasoning services supporting the query interface. These include the query language expressiveness, the ontology support to the query formulation, and the natural language verbalisation issues. Finally, we discuss related work and we draw some conclusions.

2 Query interface: 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. The interface guides 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. This structure corresponds to the natural linguistic concepts of noun phrases with one or more propositional phrases. The latter can contain nested noun phrases themselves.

A query is composed by a list of terms coming from the ontology (classes); e.g. “Supplier” and “Multinational”. Branches are constituted by a property (attributes or associations) with its value restriction, which is a query expression itself; e.g. “selling on Italian market”, where “selling on” is an association, and “Italian market” is an ontology term.

The focus paradigm is central to the interface user experience: manipulation of the query is always restricted to a well defined, and visually delimited, subpart of the whole query (the *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. This is verified against the formal model describing the data sources.

One of the main requirements for the interface is that it must be accessed by any HTML browser, even in presence of restrictive firewalls. This constraints the its design, which overall appearance is shown in Figure 1. The interface is composed by three functional elements. The

first one (top part) shows a natural language representation of the query being composed, and the current focus. The second one is the query manipulation pane (bottom part) containing a diagram representing the focus and its terminological context, together with tools to specialise the query. Finally, a query result pane containing a table representing the result structure. The first two components are used to compose the query, while the third one is used to specify the data which should be retrieved from the data sources. Because of lack of space, in this paper we concentrate on the query building part. Therefore we wont discuss the query result pane, which allows the user to define the columns of a table which is going to organise the data from the query result.

Query textual representation The first component consists of a text box representing the query expression in a natural language fashion. The user selects subparts of the query for further refinement. The selection defines the current focus, which will be represented in the diagrams described in the following sections. The selected subexpression can be modified (refined or extended) by means of the query manipulation pane.

Although the query verbalisation does not provide accounts of the query structure, the system is aware of the nesting (and so is the user). The system provides the feedback on the nesting by means of navigation in the query expression when the user is interested in selecting a subpart of the query. When a node is selected, then the system automatically selects the whole subtree rooted at the node selected by the user.

It is important to stress that, although natural language is used as feedback to represent the query, this is used in generation mode only. Since the user does not write queries directly, there is no need to parse any natural language sentence or to resolve linguistic ambiguities.

Query manipulation pane The elements in the pane represent the current selection, and the operations allowed in its context. It is organised as a diagram showing the taxonomic context of the selection (the central part), and tools enabling the user to build the query expression.

The central part of the interface is occupied by the diagram allowing what we call *substitution by navigation*; i.e. the possibility of substituting the selected portion of the query with a more specific or more general terms.

The central part in the diagram shows the main term of the focus. While the surrounding terms are either more specific or more general w.r.t. the query expression *from the focus viewpoint*. For example, w.r.t. the query showed in Figure 1 with the focus on the first term ("Supplier"), the terms "Merchant" and "Agent" are more general term in the ontology, while "Retailer" and "Wholesaler" are more specific. By selecting one of these terms, the user can substitute the whole focus with the selected term. The purpose of the substitution group is twofold: it enables the replacement of the focus and it shows the position of the selection w.r.t. the terms in the ontology.

It can be the case that in the ontology there are terms which are equivalent to the selected part. In this case the user is offered to replace the selection with the equivalent term by the activation of the `Replace Equivalent` button.

A different refinement enabled by the interface is by *compatible terms*. These are terms in the ontology whose overlap with the focus can be non-empty. These ontology terms can be added to the head of the selection by using the `Add Concept` pop-up menu. For example, "Student" is among the compatible terms for the focus "Employee", but "Textile" is not. The compatible terms are automatically suggested to the user by means of appropriate reasoning task on the ontology describing the data sources.

Analogously, the user can add properties to the focus: *associations* (e.g. “Industry with sector”), and/or *attributes* (e.g. “Employee whose name is”). This can be performed by means of a `Add Property` pop-up menu, which presents the possible alternatives. Name and value restrictions for each property are verbalised using meta information associated to the terms in the ontology. For example, the association “with sector” with the restriction “Textile” is shown as “belonging to the textile sector”.

Note that the terms and the properties proposed by the system depend on the overall query expression, not only on the focus. This means that subparts of the query expression, taken in isolation, would generate different suggestions w.r.t. those in their actual context in the query.

Sub-queries can be associated to new names by means of a `Define` button. This process corresponds to the definition of a new named view. These newly introduced names can be used to shorten the query expression, or as a simple mechanism to extend the ontology to build a customised user’s viewpoint.

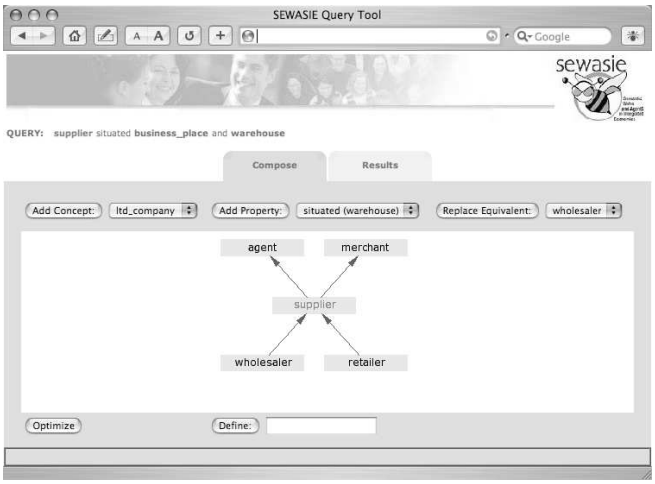

Figure 1: Query building interface.

3 Query interface: inside the box

In this section we describe the underpinning technologies and techniques enabling the user interface described in the previous sections. We will start by describing our assumptions on the query language, followed by system perspective over the described query building process. The whole system is supported by formally defined reasoning services which are described in Section 3.2. Finally, we introduce the verbalisation mechanism which enables the system to show the queries in a natural language fashion.

3.1 Conjunctive queries

Since the interface is build around the concept of classes and their properties, we consider conjunctive queries composed by unary (classes) and binary (attribute and associations) terms.

The body of a query can be considered as a graph in which variables (and constants) are nodes, and binary terms are edges. A query is connected (or acyclic) when for the corresponding graph the same property holds. Given the form of query expressions composed by the interface introduced in Section 2, we restrict ourselves to acyclic connected queries. This restriction is dictated by the requirement that the casual user must be comfortable with the language itself.[1] Note that the query language restrictions do not affect the ontology lan-

[1] Our technique can deal with disjunction of conjunctive queries, even with a limited form of negation applied to single terms. See [8; 16] for the technical details.

guage, 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, it is then added the binary query term corresponding to the property, and its restriction is recursively expanded in the same way.

Let us consider for example the query "Supplier and Multinational corporation selling on Italian market located in Europe", with the meaning that the supplier is located in Europe. Firstly, a new variable (x_1) is associated to the top level "Supplier and Multinational corporation". Assuming that the top level variable is by default part of the distinguished variables, the conjunctive query becomes

$$\{x_1 \mid \text{Suppl}(x_1), \text{Mult\_corp}(x_1), \ldots\},$$

where the dots mean that there is still part of the query to be expanded. Then we consider the property "selling on", with its value restriction "Italian market": this introduces a new variable $x_{1,1}$. The second branch is expanded in the same way generating the conjunctive query

$$\{x_1 \mid \text{Suppl}(x_1), \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}.$$

This transformation is bidirectional, so that a connected acyclic conjunctive query can be represented as a query expression (in the sense of Section 2) by dropping the variable names. As a matter of fact, the system is using this inverse transformation since the internal representation of queries is conjunctive queries.

Since a query is a tree, the focus corresponds to a selected sub-tree. It is easy to realise that each sub-tree is univocally identified by the variable corresponding to a node. Therefore, the focus is always on variable, and moving the focus corresponds to selecting a different variable. Modifying a query sub-part means operating on the corresponding sub-tree modifying the corresponding query tree.

Substitution by navigation corresponds to substitute the whole sub-tree with the chosen ontology term. The result would be a tree composed by a single node, without any branch, whose unary term is the given ontology term. In the *refinement by compatible terms*, the selected terms are simply added to the root node as unary query terms. For the *property extension*, adding an attribute or associations corresponds to the creation of a new branch. This operation introduces a new variable (i.e. node) with the corresponding restriction. When an attribute is selected, and a constant (or an expression) is specified, then this is added as restriction for the value of the variable.

3.2 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 their restrictions) which are proposed to the user to manipulate the query.

Our aim 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 iff it is not contained in the empty query). As described in Section 2, the query building interface represents the available operations on the query w.r.t. the current focus; i.e. the variable which is currently selected. Therefore, we need a way of describing a conjunctive query from the point of view of a single variable. The expression describing such a viewpoint is still a conjunctive query; which we call *focused*. This new query is equal to the original one, with the exception of the distinguished (i.e. free) variables: the only distinguished variable of the focused query is the variable representing the focus. In the following we represent as q^x the query q focused on the variable x. For example, the query
$q \equiv \{x_1, x_{1,2} \mid \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}$,
focused in the variable $x_{1,1}$ would simply be
$q^{x_{1,1}} \equiv \{x_{1,1} \mid \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}$.

The operations on the query expression require two different types of information: *hierarchical* (e.g. substitution by navigation), and on *compatibility* (e.g. refinement and new properties).

Let us consider the substitution by navigation with the more specific terms (the cases with more general and equivalent terms are analogous). Given the focused query q^x, we are interested to the unary atomic terms T s.t. the query $\{y \mid T(y)\}$ is contained in q^x and it is most general (i.e. there is no other query of that form contained in q^x, and containing $\{y \mid T(y)\}$).

Refinement by compatible terms and the addition of a new property to the query require the list of terms "compatible" with the given query. In terms of conjunctive queries, this corresponds to add a new term to the query. The term to be added should "join" with the query by means of the focused variable, and must be compatible in the sense that the resulting query should be satisfiable. This leads to the use of satisfiability reasoning service to check which predicates in the ontology are compatible with the current focus. With unary terms this check corresponds simply to the addition of the term $T(x)$ to the focused query q^x, and verify that the resulting query is satisfiable.

The addition of a property requires the discovery of both a binary term and its restriction: the terms to be added are of the form $\{x \mid R(x, y), T(y)\}$ if the focused variable is x. As for the refinement by compatible terms, the system should check all the different binary predicates from the ontology for their compatibility. This is practically performed by verifying the satisfiability of the query $q^x \bowtie \{x \mid R(x, y)\}$, for all atomic binary predicates R in the signature and where y is a variable not appearing in q.[2] Once a binary predicate R is found to be compatible with the focused query, the restriction is selected as the most general unary predicate T such that the query $q^x \bowtie \{x \mid R(x, y), T(y)\}$ is satisfiable.

[2]Here $\bowtie$ represents a natural join.

3.3 Using a Description Logics Reasoner

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 [15]); 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 (see [14]).

For space limitations we are not going to describe in detail the underlying $\mathcal{SHIQ}$ DL; the reader is referred to the above mentioned bibliographic references. The ontology contains unary (concepts) and binary (roles) predicates, and the constraints are expressed by means of inclusion axioms between concept or role expressions. One of the key features of $\mathcal{SHIQ}$ is the possibility of expressing the inverse of a role; which is extremely useful for converting tree–shaped queries into DL concept expressions.

Given the restriction to tree–shaped conjunctive query expressions, together with the availability of inverse roles, a focused query (see Section 3.2) corresponds to a concept expression (see [17]). Therefore, all the reasoning tasks described in Section 3.2 correspond to standard DL reasoning services. Again, this is not a restriction imposed by the underlying technology, since general conjunctive queries can be dealt with techniques described in [8; 16].

The idea behind the transformation of a query expression into a single concept description is very simple, and it is based on the fact that a concept expression can be seen as a query with a single distinguished variable. To focus the query on a variable, we start from the variable itself, then we traverse the query graph by encoding binary terms into DL existential restrictions and dropping the variable names. The fact that queries are tree–shaped ensures that variable names can be safely ignored. Let us consider for example the query expression

$$\{\text{Mult\_corp}(x_1), \text{Italian}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1})\}.$$

The DL expression corresponding to the query focused on $x_{1,1}$ is

$$(\text{It\_market} \sqcap \exists \text{sell\_on}^{-}(\text{Mult\_corp} \sqcap \text{Italian}));$$

where sell_on^{-} corresponds to the inverse of sell_on role.

As explained in Section 3.2, we need two kinds of information: hierarchical and compatibility. These, in the DL framework, are provided by the standard reasoning services of satisfiability and taxonomy position of a concept expression respectively. The first service verifies the satisfiability w.r.t. a knowledge base; while the second classifies a concept expression (i.e., provides it w.r.t. the ISA taxonomy of concept names).[3] Reasoning tasks described in Section 3.2 can be straightforwardly mapped into satisfiability and classification.

For example, checking the compatibility of the term Italian with the query

$$\{\text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1})\},$$

is performed by checking the satisfiability of the concept

$$\text{Italian} \sqcap \text{Mult\_corp} \sqcap \exists \text{sell\_onIt\_market}.$$

Compatibility of binary terms is performed analogously by using an existential restriction; e.g., $\exists \text{sell\_on}\top$.[4] To discover the restriction of a property we use classification instead of

[3] DL systems usually provide an efficient way of obtaining the taxonomic position of a given concept expression.

[4] Note the use of the $\top$ concept representing the whole domain (any possible concept).

repeated satisfiability. The idea is to classify the query focused on the variable introduced by the property. For example, to discover the restriction of sell_on applied to the query expression

$$\{x_1 \mid \text{Mult\_corp}(x_1), \text{Italian}(x_1)\},$$

we classify the expression $\exists \text{sell\_on}^{-}(\text{Mult\_corp} \sqcap \text{Italian})$. The DL reasoner returns the list of concept names more general and equivalent to the range of the relation sell_on, when restricted to the domain $(\text{Mult\_corp} \sqcap \text{Italian})$. This is exactly the information we need to discover the least general predicate(s) which can be applied to the property in the given context.

Our implementation uses the DL reasoner Racer (see [12]); which fully supports the $\mathcal{SHIQ}$ DL. The interaction with the DL reasoner is based on the DIG 1.0 interface API (see [1]), 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 substituted with any DIG based one.

3.4 Query verbalisation

The system always presents the user with a natural language transliteration of the conjunctive query. This is performed in an automatic way by using meta information associated with the ontology terms, both classes and properties. The verbalisation of the ontology terms must be provided in advance by the ontology engineers. For the verbalisation we use an approach similar to the one adopted by the Object Role Modelling framework (ORM, see [13; 19]).

Each class name in the ontology has associated a short noun phrase (usually one or two words), which represents the term in a natural language fashion. For example, to the class *PStudent* is associated "Postgraduate student" The user will see only the associated sentence, while *PStudent* is just used in the internal ontology representation.

For (binary) associations the ontology engineer has to provide two different verbalisations for the two directions. For example, let assume that the ontology states that the association *occ_room* links the two classes *PStudent* and *Room*. Then the engineer associates to the association the verbalisation "occupies" for the direction from *PStudent* to *Room*, and the verbalisation "is occupied by" for the other direction.

Attributes need one direction only, since they are never used from the point of view of the basic data type. In this case, the engineer is only required to provide the attribute verbalisation from the point of view of the class.

4 Discussion

The work proposed in this paper deals with a relatively new problem, namely providing the user with a visual interface to query heterogeneous data sources through an integrated ontology (that is, a set of constraints), and a specific literature does not exist yet. By looking at the extensive survey on Visual Query System (VQS) presented in [10] it easy to see that only little work has been done in the specific context we are dealing with. Some preliminary work was done by one research group [4; 11; 6; 5]. 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 the context of the Tambis project [18; 2]. Also [3] contains some interesting approach from the point of view of the visual interface, but again the system has a different background semantics.

In fact, only recently research has started to have a serious interest in query processing and information access supported by 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 [20; 7]. This means that the notion of accessing information through the navigation of an ontology modelling the information domain has its formal foundations.

This paper has presented the first well-founded intelligent user interface for query formulation support in the context of ontology-based query processing. This paper hopefully proved that our work has been done in a rigorous way both at the level of interface design and at the level of ontology-based support with latest generation logic-based ontology languages such as description logics, DAML+OIL and OWL. However, there are open problems and refinements which have still to be considered in our future work.

The system uses the verbalisations described in Section 3.4 to transform the conjunctive query into a natural language expression closer to the user understanding. In the course of the SEWASIE project some effort will be dedicated to explore semi-automatic techniques to rephrase the expressions in more succinct ways without loosing their semantic structure.

Another important aspect to be worked out is the understanding of the effective methodologies for query formulation in the framework of this tool, a task that needs a strong cooperation of the users in its validation. This will go in parallel with the interface user evaluation, which is just starting at the time of writing this paper.[5] The other crucial aspect is the efficiency and the scalability of the ontology reasoning for queries. We are currently experimenting the tool with various ontologies in order to identify possible bottlenecks.

We would like to thank Tiziana Catarci, Tania Di Mascio, and Giuseppe Santucci, for their valuable suggestions and discussions on the user interface. Moreover, the support of Ralf Möller and Volker Haarslev with the Racer reasoner has been essential for the development of our system prototype.

References

[1] Sean Bechhofer, Ralf Mller, and Peter Crowther. The dig description logic interface. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, 2003.

[2] Sean Bechhofer, Robert Stevens, Gary Ng, Alex Jacoby, and Carole A. Goble. Guiding the user: An ontology driven interface. In *UIDIS 1999*, pages 158–161, 1999.

[3] Francesca Benzi, Dario Maio, and Stefano Rizzi. VISIONARY: a viewpoint-based visual language for querying relational databases. *J. Vis. Lang. Comput.*, 10(2):117–145, 1999.

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

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

[5] An on-line prototypical version of the query building tool, with a toy ontology without lexicalisation, is available at the URL `http://dev.eurac.edu:8090/sewasie/`.

[6] Paolo Bresciani, Michele Nori, and Nicola Pedot. A knowledge based paradigm for querying databases. In *Database and Expert Systems Application*, volume 1873 of *Lecture Notes in Computer Science*, pages 794–804. Springer Verlag, 2000.

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

[8] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[9] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *Proc. of the 6th Int. Conf. on Cooperative Information Systems (CoopIS'98)*, pages 280–291, 1998.

[10] Tiziana Catarci, Maria Francesca Costabile, Stefano Levialdi, and Carlo Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[11] Enrico Franconi. 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"*, 2000.

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

[13] Terry A. Halpin. Augmenting UML with fact orientation. In *HICSS*, 2001.

[14] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.

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

[16] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to decide query containment under constraints using a description logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.

[17] Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[18] Norman Murray, Carole Goble, and Norman Paton. A framework for describing visual interfaces to databases. *J. Vis. Lang. Comput.*, 9(4):429–456, 1998.

[19] `http://www.orm.net`, 2003.

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