

Using Ontologies for Database Query Reformulation

Chokri Ben Necib and Johann-Christoph Freytag

Humboldt-Universität zu Berlin, Germany,
{necib,freytag}@dbis.informatik.hu-berlin.de

Abstract. Query reformulation techniques based on semantic knowledge have been used in two ways in database management systems, namely for query optimization and for data integration. For the former approach, the main goal of query reformulation is to rewrite a user query into another one that uses less time and/or less resources during the execution. The main goal of the latter approach is to translate a user query into a set of queries which fit best the structure of the distributed sources. When using those query optimization strategies the transformed queries are equivalent to the submitted ones, i.e. they provide the same answer set, whereas when using data integration strategies a loss of information is expected. This paper presents a new approach of query reformulation using ontology semantics for query processing within a single relational database system. Here, the aim of the query reformulation is to extend the result of a given query in a semantically meaningful way. In fact, our approach shows how an ontology can effectively be exploited to rewrite a user query into another one such that the new query provides additional meaningful results that satisfy the intention of the user. Based on practical examples and their usefulness we develop a set of reformulation rules. In addition, we prove that the results of the reformulations are semantically correct by using a logical model.

1 Introduction

In recent years, a large number of research approaches have used semantic knowledge for supporting data management to overcome problems caused by the increasing growth of data in local databases, and the variety of its format and model in distributed databases.

The use of semantic knowledge in its various forms including meta-models, semantic rules, and integrity constraints can improve query processing capabilities by transforming user queries into other semantically equivalent ones, which can be answered in less time and/or with less resources. Sometimes, the system does not need to scan the complete database to answer a query and a short answer might satisfy the user needs. This emerging database aspect is known as *semantic query optimization*. Furthermore, There are also several mechanisms in knowledge databases that use semantic knowledge based on a set of intentional knowledge including deduction rules, generalized rules, and concept hierarchies

in order to provide an "intelligent answer" for queries. "Intelligently answering" a query refers to providing the user with intentional answers in addition to the data (facts). These answers include some generalized, neighborhood, or associated information which characterizes the data results [1]. Currently, research work on the Semantic Web and data integration are focusing on using *ontologies* as semantic support for data processing. Ontologies have proven to be useful to capture the semantic content of data sources and to unify the semantic relationships between heterogenous structures. Thus, users should not care about where and how the data are organized in the sources. For this reason, systems like OBSERVER [2] and TAMBIS [3] allow users to formulate their queries over an ontology without directly accessing the data sources.

In this paper, we present a new approach on how to improve the answers of queries based on semantic knowledge expressed in ontologies. Given a database, we assume the existence of an ontology which is associated with the database and which provides the context of its objects. We show how an ontology can be exploited effectively to reformulate a user query such that the new query can provide more "meaningful" results meeting the intention of the user. A query can be defined by a set of selections and projections over database objects satisfying a set of conditions. These conditions are defined by a set of terms and determine the answer to the query. If a user wants to retrieve information from a database about certain objects, he might use terms, which do not exactly match the database values (due to the mismatch between the user's world view and the database designer's world view). However, there might be values in the database that are syntactically different from user's terms but have the same meaning and express the same intention of the user. We address this issue as a semantic problem rather than as a pattern matching problem. As a consequence, if we consider semantics in query processing, the number of results for the transformed query might increase or decrease. In both cases the user receives an answer that meets his expectations compared to the result without using additional knowledge. Compared to query optimization methods the aim of our approach is not to speed up query processing but to provide users with additional meaningful answers. To this end, we develop a set of reformulation rules and give practical examples to show their usefulness. In addition, we prove that the results of the transformations are semantically correct by using a mathematical model. Finally, we define two consistency criteria, correctness and completeness, to validate our framework.

The remainder of this paper is organized as follows. The next section present a brief description of related work. In section 3 we give a definition of an ontology and one of its formal representation. In section 4 we state the problem and motivate it by means of an example. In section 5 we illustrate our approach for query processing and propose necessary reformulation rules. In section 6 we discuss a framework to validating these rules, and finally we conclude the paper.

2 Related Work

Work related to query reformulation using semantic knowledge has emerged in two different research areas: Semantic query optimization and global information processing area.

Semantic query optimization. The basic idea of semantic query optimization (SQO) is to reformulate a query to another more efficient query, which is semantically equivalent, i.e. provides the same answer [4, 5, 6, 7]. Here, SQO approaches use semantic knowledge in various forms including semantic rules and range rules. Range rules states facts about the range of values of a given attribute, whereas semantic rules define the regularity of data for a given database. Therefore, these rules can be driven from the non-uniform distribution of values in a database. Expressing semantics in the form of horn clause sets allows the optimizer to make possible reformulations on an input query involving the insertion of new literals, or the deletion of literals, or the refuting the entire query. Several approaches on SQO have been developed to address different aspects of query processing: In [8] semantic rules have been used to derive useful information, which can reduce the cost of query plans. In [9, 10] algorithms have been developed for optimizing conjunctive sub-queries. To this end, learning techniques have been applied to generate semantic (operational) rules from a database automatically [11]. While the previous approaches are based on extracting semantic knowledge from the underlying database, current research approaches use knowledge from additional source [12, 13].

Data integration. The challenging problem in global information systems is the lack of uniformity when processing queries. Promising solutions are based on using semantic knowledge in the form of meta data which describes the content of each source [14]. For example, in [2] all information sources are described by one global ontology which provides shared semantics for users. Queries are then formulated in terms of that ontology. In [3], however, each information source is described by one or more ontologies. Users formulate their queries over one of these ontologies and it is the responsibility of the system to reformulate them into sub-queries, executes them on each source, and manages the different answers. To this end, the system relies on the ontological relationships between the different ontologies such as synonym, hyponym and hypernym relationships. Note that most of work in this field addresses the issue of resolving semantic conflicts using semantics of relationships between schema elements of the sources rather than between their extensions.

3 What is an Ontology

Since ontologies are the basic key features of our approach, it is necessary to outline our point of view of what an ontology is. We give also a formal representation of an ontology.

3.1 Definition

The term "Ontology" or "ontologies" is becoming frequently used in many contexts of database and artificial intelligence researches. However, there is not a unique definition of what an ontology is [15, 16, 17, 18]. An initial definition was given by Tom Gruber: "an ontology is an explicit specification of a conceptualization" [15]. However, this definition is general and remains still unsatisfied for many researchers. In [16] Nicola Guarino argues that the notion of "conceptualization" is badly used in the definition. We note that many real-world ontologies already combine data instances and concepts [16]. Our definition differ from this point of view as we show later .

Informally, we define an ontology as an intentional description of what is known about the essence of the entities in a particular domain of interest using abstractions, also called *concepts* and the *relationships* among them. Basically, the hierarchical organization of concepts through the inheritance ("ISA") relationship constitutes the backbone of an ontology. Other kinds of relationship like part-whole ("PartOf") or synonym ("SynOf") or application specific relationships might exist. To the best of our knowledge, there is no work until now addressing the issue of using ontology relationships at the database instance level. In [19] we compare the term "ontology" with similar terms such as "Controlled Vocabulary", "Taxonomy", and "Knowledge Base". Despite the disagreement upon a common meaning of an "ontology", the role of ontologies that must play is clear: Ontologies should provide a concise and unambiguous description of concepts and their relationships for a domain of interest. Ontologies are shared and reused by different agents i.e. human or/and machines.

Formally, we define an ontology as a set ζ and a set \mathfrak{R} as follows: $\zeta = \{c_1, \dots, c_n\}$ and $\mathfrak{R} = \{"ISA", "SynOf", "PartOf"\}$, where $c_i \in \zeta$ is a concept name, and $r_i \in \mathfrak{R}$ is the type of the binary relation relating two concepts (c_i and r_i are non-null strings). Other domain-specific types may also exist. At the top of the concept hierarchy we assume the existence of a universal concept, called "Anything", which represents the most general concept of an ontology. In the literature, the word "concept" is frequently used as a synonym for the word "concept name". Hence, for the design of an ontology only one term is chosen as a name for a particular concept [20]. Further, we assume that the terms "concept" and "concept name" have the same meaning.

3.2 Formal Representation

This section presents a graph-based representation of an ontology. We introduce its basic formal settings, and some related operations relevant to further discussions.

Graph oriented model. We represent an ontology as a directed graph $G(V, E)$, where V is a finite set of vertices and E is a finite set of edges: Each vertex of V is labelled with a concept and each edge of E represents the inter-concept relationship between two concepts. Formally, the label of a node $n \in V$ is defined

by a function $N(n) = c_i$ that maps n to a string c_i from ζ . The label of an edge $e \in E$ is given by a function $T(e) = r_i$ that maps e to a string r_i from \mathfrak{R} .

Figure 1 gives an example of a graph representation of a fragment of the ontology "Product". A part of this ontology is adopted from an ontology described in [21].

In summary, an ontology is the set $O = \{G(V, E), \zeta, \mathfrak{R}, N, T\}$.

Graph operations. In order to navigate the ontology graph, we define the following sets of concepts: *Rparent*, *DESC*, *SUBT*, *SYNs*, *PARTs* and *WHOLEs*. We need these operations to identify nodes in the graph, which hold concepts that are of interest for our query reformulations.

Let $P_{ths}(n_1 - n_2)$ be a set of directed paths between two nodes n_1 and n_2 . We denote by $node(c)$ the node labelled by a concept c , and by $child(n)$ and $parent(n)$ the child-node and parent-node of a node n , respectively.

Given two nodes $n_1 = node(c_1)$ and $n_2 = node(c_2)$ the operation are formulated as follows:

- $Rparent(r, c_1) = c_2$ iff $n_2 = parent(n_1)$ and $T[(n_2, n_1)] = r$
- $DESC(r, c) = \{s \in \zeta \mid \exists p \in P_{ths}(node(c) - node(s)) : \forall e \in p, T(e) = r\}$
- $SYNs(c) = \{s \in \zeta \mid \exists p \in P_{ths}(node(c) - node(s)) : \forall e \in p, T(e) = \text{"SynOf"}\}$
- $SUBT(c) = \{s \in \zeta \mid \exists P_{ths}(node(c) - node(s))\}$

Informally, $Rparent(r, c)$ returns the label of the parent node of a concept c by following an edge of type r . $DESC(r, c)$ returns the set of all concepts in O whose nodes are children of the node of c by means of descending edges of type r . Similarly, $SUBT(c)$ returns all descendants of c for any edge-type and $SYNs(c)$ returns the set of all synonyms of c in O . In addition, we define an *Outgoings*(n) as a set of edge-types going out from a node n and *PARTs*(c) as the set of concepts whose nodes are related to the $node(c)$ through the edges of type "PartOf". Here, two cases must be distinguished:

- Case 1: If $Outgoings(node(c)) \ni \text{"PartOf"}$ then $PARTs(c) = A \cup B \cup C$, where
 - $A = DESC(\text{"PartOf"}, c)$
 - $B = DESC(\text{"ISA"}, a), a \in A$
 - $C = SYNs(h) \cup SYNs(l), h \in A$ and $l \in B$

Informally, $PARTs(c)$ is the set of concepts obtained by retrieving the labels of all nodes that are PartOf-children of the $node(c)$ together with their ISA-descendants and synonyms.

- Case 2: If $Outgoings(node(c)) \ni \text{"ISA"}$ then $PARTs(c) = PARTs(s_i)$

where

$$s_i \in A \text{ and } \forall (s_1, s_2) \in A^2 \text{ } PARTs(s_1) = PARTs(s_2), A = DESC(\text{"ISA"}, c).$$

Informally, $PARTs$ of a concept c is defined recursively in terms of its sub-concepts. It is equal to the $PARTs$ of one of its sub-concepts (if they have the same $PARTs$).

Inversely, we define *WHOLEs* of a given concept c as the set of concepts c_i such that $c \in PARTs(c_i)$.

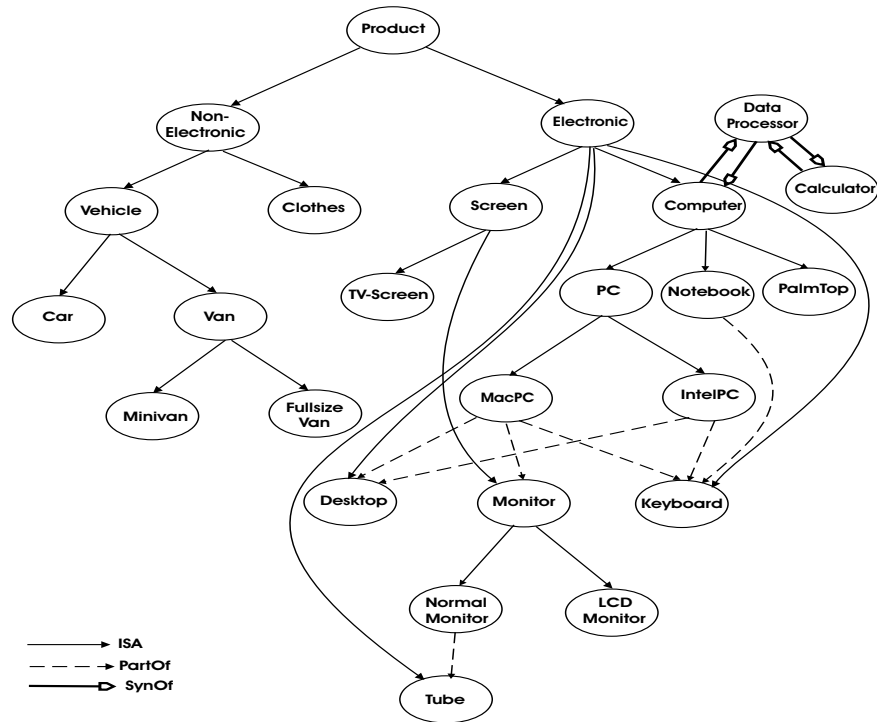


Fig. 1. Product Ontology

4 Problem Motivation

The problem of query processing addressed in this paper is neither a query optimization problem nor a problem of query expansion over a global integrated system. The aim of query optimization is to reduce the costs of processing while the aim of query expansion in global systems is to overcome the heterogeneous problem. In this paper we address the problem of semantically querying a single database using semantic knowledge from an ontology. We introduce a new semantic approach dealing with this problem and prove its soundness. The goal of our approach is to provide a user with "meaningful answers" to his queries. We consider databases organized in the relational data model because it is well-defined and widely used in practice. However, it should be emphasized that our approach can be applied to other databases in more expressive data models such as the object-relational model with little extensions.

The basic idea is to give the DBMS the ability to deal with user queries both at the semantic as well as the syntactic level. Here, we intend to find answers to user's request that can meet his intention in a way that can be partially independent from the query syntax. To illustrate this point of view, let us have the following example:

Assume that we have an ontology O_1 and a relational database DB_1 . O_1 describes product concepts (see Figure 1). DB_1 contains information about technical items of a store and includes two relations called 'Article' and 'Component': The relation Article contains a set of items described by the attributes 'name', 'model' and 'price'. The relation component contains the parts belonging to each item. The relational schema of DB_1 is the following:

ARTICLE(A-ID, Name, Model, Price)	COMPONENT(S-ID, M-ID)
A-ID: Article identifier	M-ID: Main part identifier
Name: Name of the article	S-ID: Second part identifier
Model: Model of the article	F-Key(M-ID) TO ARTICLE
Price: Price of the article	F-Key(S-ID) TO ARTICLE
P-Key(A-ID)	P-Key(S-ID,M-ID)

We suppose, at present, that DB_1 contains the instances as shown in the tables 1 and 2. Querying the database DB_1 to retrieve information about ar-

A-ID	Name	Model	Price
123	computer	ibm	3000 \$
124	intelPc	toshiba	5000 \$
125	notebook	dell	4000 \$
127	pc	compaq	2500 \$
128	product	hp	3000 \$
129	monitor	elsa	1000 \$
135	keyboard	itt	80 \$
136	desktop	ibm	1000 \$
140	macPc	mac	2000 \$
141	calculator	siemens	1500 \$

Table 1. Article relation

S-ID	M-ID
123	129
123	135
123	136
124	129
124	135
124	136
125	135
127	129
127	135
127	136
128	129
128	135
128	136
140	129
140	135
140	136
141	135

Table 2. Component relation

ticle "computer" means also information about articles "data processor" and "calculator" because these terms are synonymous with the term "computer". This interpretation can be deduced from the ontology O_1 . Consequently, if a user formulates his query specifying only the term "computer" he might miss other tuples in the database concerning "data processor" and "calculator". This example seems to be simple but there could be more complicated situations depending on the nature of the query as we shall see later. In fact, the difference between the user's perception of real world objects and the database designer

who registers information about these objects might cause semantic ambiguities including the "vocabulary problem". Thus, it is hard for the DBMS to solve such semantic problems without additional sources like ontologies. For this example, the ontology O_1 provides additional meanings for the database values of DB_1 related to a certain attribute name. We propose a method that can reformulate user queries by substituting a set of query terms by other sets, which are semantically equivalent. We define two set of terms to be semantically equivalent if their corresponding concepts together with their related relationships identify the same concept with respect to a given ontology. For example, two terms which are synonymous are semantically equivalent. The result of a possible query reformulation could lead to retrieving more tuples than expected by the original query and therefore can meet the user's expectations.

In summary, the problem can be stated as follows:

Given a database DB , an ontology O and a user query Q , how to find a reformulated query Q' of Q by using O , such that Q' returns to the user more meaningful results than Q .

5 Query Reformulation Rules

In this section, we present four rewriting rules and apply them to examples. The validation of these rules and the proof of their soundness will be discussed in the next section. First, we introduce some notations that allow us a clear representation of queries and their transformations throughout the rest of the paper:

Given an ontology $O = \{G(V, E), \zeta, \mathfrak{R}, N, T\}$ and a database DB . Let U be a set of attributes A_1, \dots, A_n with domains $dom(A_i)$. Let D be the set of all attribute domains. The database schema is defined as a set of relation schemas R_1, \dots, R_m with $R_i \subseteq U$. We denote by $PKEYS(U)$, the set of primary Keys and by $FKEYS(R_i, R_j)$ the set of foreign keys in R_i to R_j . Furthermore, we choose the Domain relational calculus (DRC) to represent user queries of the form $Q = \{s \mid \psi(s)\}$, where s is a tuple variable and $\psi(s)$ is a formula built from atoms and a collection of the logical operators " \wedge " and " \vee " (for more details see [22]).

In order to extract ontology semantics related to the content of the associated database, a connection between the ontology and the database must be established i.e. the ontology concepts and relationships must relate to the database elements. We refer to this relation as mappings. We assume that these mappings already exist. Currently, there is no automatic method for creating these mappings but semi-automatic methods based on linguistic matchings might be adequate for this purpose [23]. Let δ_1 be the mapping that represents matchings between relation names of DB and concepts of ζ called *relation-concepts*; δ_2 be the mapping that represents matchings between attribute names of DB and concepts of ζ called *attribute-concepts*, and δ_3 be the mapping that represents matchings between database values and concepts of ζ called *value-concepts*. Fi-

nally, let δ_4 be the mapping that represents matchings between a pair of attribute names of DB and relationship types of \mathfrak{R} .

5.1 Vocabulary-Rule

This rule bridges the semantic gap of the vocabularies used in the user's query and those stored in the database. Intuitively, the rule reformulates the query condition by using semantically equivalent terms derived from a given ontology following synonym- and specialization relationships.

Formally, let be $t_0 \in D$, the vocabulary-rule is formulated as follows:

Rule 1:

IF $Q = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R \wedge x_i \theta t_0\}$
and $\exists c_0 \in \zeta, \delta_3(t_0) = c_0$

THEN $Q' = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R \wedge (\bigvee_{k=0}^m x_i \theta t_k)\}$

where $t_k \in I_0 \cup I_1, 0 \leq k \leq m = |I_0 \cup I_1|$
 $I_0 = \{t \in D \mid \delta_3(t) \in DESC("ISA", c_0)\} \cup \{t_0\},$
 $I_1 = \{t \in D \mid \delta_3(t) \in SYNs(c), c \in DESC("ISA", c_0)\}.$

This rule formalizes the idea behind the example mentioned in previous section.

Query Example 1. A user might submit a query to the database DB_1 as the follows:

$$Q_1 = \{(x_1, x_2, x_3, x_4) \mid (x_1, x_2, x_3, x_4) \in ARTICLE \wedge x_2 = "computer"\}.$$

However, according to the ontology O_1 the concept "Computer" is synonymous with the concepts "Data Processor" and "Calculator". Furthermore, it has a broader meaning than the specialized concepts "Notebook" and "PalmTop". Intuitively, the ISA-relationship implies a strong similarity between a concept and its sub-concepts. Since the ISA-relationship is transitive, the same argument can be applied to further specialization i.e. "MacPC" and "IntelPC". Thus, by applying the rule above we get the following new query:

$$Q'_1 = \{(x_1, x_2, x_3, x_4) \mid (x_1, x_2, x_3, x_4) \in ARTICLE \wedge (x_2 = "computer" \vee x_2 = "data - processor" \vee x_2 = "calculator" \vee x_2 = "pc" \vee x_2 = "notebook" \vee x_2 = "palmtop" \vee x_2 = "macpc" \vee x_2 = "intelpc")\}.$$

Note, that the answer to the reformulated query Q'_1 will include new tuples identified by the ids: 124, 125, 127, 140 and 141. We conclude that the result of Q'_1 is more meaningful (in the sense of "more complete" according to the semantic knowledge) than of Q_1 .

5.2 Part-Whole Rule

The basic idea of this rule is the use of "part-whole" properties to discover new database objects, which are closely related to a given user query. Based on the semantic relationship "PartOf" this rule can rewrite a user query by substituting the query terms by new terms which are semantically equivalent according to our definition (see section 1). For this rule, the concepts corresponding to the new terms together with the "PartOf"-relationships specify the same concepts which correspond to the original query terms. Thus, the same type of the object specified in the query can be defined in another way by using an alternative set of terms.

Formally, let be $t_0 \in D$ and $R_1, R_2 \in DB$. The part-whole rule is formulated as follows:

Rule 2:

IF $Q = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R_1 \wedge x_i \theta t_0\}$

and $\exists c_0 \in \zeta, \delta_3(t_0) = c_0$

and $\exists A_1, A_2 \in FKES(R_2, R_1) \mid \delta_4(A_1, A_2) = \text{"PartOf"}$

and $\forall c_i \in \zeta, c_i \neq c_0, PARTS(c_0) \not\subseteq PARTS(c_i)$

THEN $Q' = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R_1 \wedge x_i \theta t_0\} \cup$
 $\{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R_1 \wedge [\exists(y_{11}, \dots, y_{n1}) \mid (y_{11}, \dots, y_{n1}) \in R_2 \wedge$
 $x_1 = y_{11} \wedge \exists(z_{11}, \dots, z_{n1}) \in R_1 \wedge (z_{11} = y_{21} \wedge z_{i1} = s_1)] \wedge \dots \wedge$
 $[\exists(y_{1m}, \dots, y_{nm}) \mid (y_{1m}, \dots, y_{nm}) \in R_2 \wedge x_1 = y_{1m} \wedge \exists(z_{1m}, \dots, z_{nm}) \in$
 $R_1 \wedge (z_{1m} = y_{2m} \wedge z_{im} = s_m)]\}$

where $s_j \in I_0 = \{t \in D \mid \delta_3(t) \in PARTS(c_0)\}, 1 = < j = < m = |I_0|$.

Query Example 2. For instance, if a user wants to retrieve information about the article "pc" from the database DB_1 , his submitted query may look like:

$$Q_2 = \{(x_1, x_2, x_3, x_4) \mid (x_1, x_2, x_3, x_4) \in ARTICLE \wedge x_2 = \text{"pc"}\}.$$

If we investigate the ontology O_1 we deduce that a "pc" is composed out of three parts: a "desktop", a "monitor" and "a "keyboard". Assuming, that all the PC-objects in the database are composed exactly out of these parts, which do not participate in the composition of any other object, enables the identification of PCs by means of their components. Thus, the set of terms {"desktop", "monitor", "keyboard"} and the term "pc" are semantically equivalent.

By applying this rule to the query Q_2 we get the following reformulated query:

$$Q'_2 = \{(a_1, a_2, a_3, a_4) \mid (a_1, a_2, a_3, a_4) \in ARTICLE \wedge a_2 = \text{"pc"}\} \cup$$

$$\{(a_1, a_2, a_3, a_4) \mid (a_1, a_2, a_3, a_4) \in ARTICLE \wedge [\exists y_1, y_2 \mid (y_1, y_2) \in$$

$$COMPONENT \wedge a_1 = y_1 \wedge \exists (b_1, b_2, b_3, b_4) \in ARTICLE \wedge$$

$$(y_2 = b_1 \wedge b_2 = \text{"monitor"})] \wedge [\exists z_1, z_2 \mid (z_1, z_2) \in COMPONENT \wedge$$

$$a_1 = z_1 \wedge \exists (c_1, c_2, c_3, c_4) \in ARTICLE \wedge (z_2 = c_1 \wedge c_2 = \text{"keyboard"})] \wedge$$

$$[\exists u_1, u_2 \mid (u_1, u_2) \in COMPONENT \wedge a_1 = u_1 \wedge [\exists d_1, d_2, d_3, d_4$$

$$(d_1, d_2, d_3, d_4) \in ARTICLE \wedge u_2 = d_1 \wedge d_2 = \text{"desktop"}])]\}$$

As results, the tuples 123 and 128 with attribute values "computer" and "product" meet also fully the user's intention. When a user poses the Q_2 -query to the DB_1 -database, these tuples will certainly be missed. The DBMS query processor has to extend the user query considering these semantics in addition to those suggested by the previous rule. Note that in this case the number of tuples in the answer result will also increase.

5.3 Support Rule

The basic idea of this rule is the use of structural information about objects by means of exploiting one of the intrinsic properties of the **PartOf**-relationship, namely transitivity. In fact, by using the transitivity property additional information about the components can be deduced. Thus, this information can be used to reformulate a user query as illustrated by the following example:

Assume that we have a database DB_2 containing entries about molecules. DB_2 has a relation, called 'Compound', which describes molecule compounds as shown in table 3:

COMPOUND(C-ID, Name, Component)	C-ID	Name	Component
C-ID: Compound Identifier	10	carbon dioxide	carbon
Name: Molecule name	20	protein	amino-acid
Component: Compound Component	30	enzyme	amino-acid
P-Key(C-ID)	40	dna	nucleotide
	50	water	hydrogen
	60	rna	nucleotide
	70	methane	carbon
	80	ethane	carbon

Table 3. Compound relation

In addition, we suppose that we have a portion of a biological ontology, denoted by O_2 (see figure 2). This ontology describes molecules and their components.

Query Example 3. We suppose now that a user asks for information about all molecular compounds that contain carbons using DB_2 . His query can be represented as follows:

$$Q_3 = \{x_2 \mid (x_1, x_2, x_3) \in COMPOUND \wedge x_3 = "carbon"\}.$$

By submitting the query Q_3 to DB_2 , the DBMS returns three tuples including the carbon dioxide, the methane and the ethane compound names. However, if we investigate the ontology O_2 , we can deduce that carbons are also contained in other molecules among which are: amino acid, carbon dioxide, ribonucleotide, and deoxyribonucleotide molecules. Therefore, an appropriate transformation of Q_3 based on O_2 is the following:

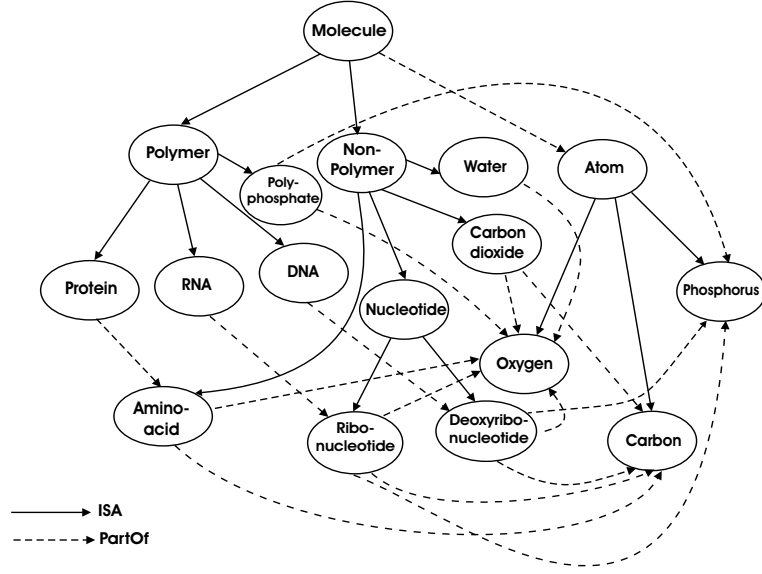


Fig. 2. Biological Ontology

$$Q'_3 = \{x_2 \mid ((x_1, x_2, x_3) \in \text{COMPOUND} \wedge (x_3 = \text{"carbon"} \vee x_3 = \text{"protein"} \vee x_3 = \text{"carbon dioxide"} \vee x_3 = \text{"nucleotide"} \vee x_3 = \text{"ribonucleotide"} \vee x_3 = \text{"deoxyribonucleotide"} \vee x_3 = \text{"dna"} \vee x_3 = \text{"rna"} \vee x_3 = \text{"amino acid"}))\}.$$

A user might not be aware of such information. Surprisingly, if we apply this semantics to the query Q_3 , then the answer will include four additional tuples related to RNA, DNA, enzyme, and protein compound names.

Formally, let be $t_0 \in D$ and $R \in DB$, the support rule is formulated as follows:

Rule 3:

$$\begin{aligned} \text{IF} \quad & Q = \{x_i \mid (x_1, \dots, x_n) \in U(R) \wedge x_p \theta t_0\} \\ & \text{and } \exists c_0 \in \zeta, \delta_3(t_0) = c_0 \\ & \text{and } \exists A_i, A_p \in U, \delta_4(A_i, A_p) = \text{"PartOf"} \\ \text{THEN} \quad & Q' = \{x_i \mid (x_1, \dots, x_n) \in R \wedge [(x_p \theta t_0) \bigvee_{k=1}^m (x_p \theta t_k)]\} \\ & \text{where } t_k \in I_0, 1 \leq k \leq m = |I_0|, I_0 = \{t \in D \mid \delta_3(t) \in \text{WHOLEs}(c_0)\} \end{aligned}$$

5.4 Feature Rule

This rule exploits domain-specific relationships of the ontology to take advantages of additional features about objects in the database instance. In fact, this

rule can rewrite user queries by using such specific features. Intuitively, we illustrate this idea by this simple example:

Assuming we have a database DB_3 of a store, which contains entries about some fruit goods. The DB_3 schema contains a relation, called 'Goods', which includes the name of each good, its color and its quantity (in kg). An instance of DB_3 and a description of the relation 'Goods' are given as follows:

GOODS(G-ID, Name, Color, Quantity)

G-ID: Good Identifier

Name: Good name

Color: Good Color

Quantity: Quantity of good Color

P-Key(G-ID)

G-ID	Name	Color	Quantity
11	kiwi	green	30
12	banana	yellow	50
13	apple	red	70
14	cherries	red	20
15	peach	red	38
16	red-apple	red	52
17	plums	red	45

Table 4. Goods relation

In addition, we suppose that we have an ontology, denoted by O_3 , which describes fruit concepts and their relationships. Figure 3 shows a portion of O_3 . There, an additional domain-specific relationship-type ("HasColor") is used. This type means that if two concepts, say A and B, have a relationship of type "HasColor", then the objects referred by A have the color referred by B. Furthermore, another type of relationship ("ISAspecific") is introduced. This type is similar to "ISA" but with additional restrictions for the related concept's objects. In fact, "ISAspecific" is always combined with one or more domain-specific relationships when it is used to describe concepts. If a concept A subsumes a concept B through "ISAspecific" then the objects represented by A and B are distinguished only by a set of features specified by means of the domain-specific relationships associated with B. For example, the objects referred by the concept "Red Apple" are more specified than those referred by "Apple" since their color can be determined ("red").

Query Example 4. We suppose now that a user wants to retrieve all tuples from DB_3 concerning 'red Apple'. His query can be represented as follows:

$$Q_4 = \{(x_1, x_2, x_3, x_4) \mid (x_1, x_2, x_3, x_4) \in GOODS \wedge x_2 = "red\ apple"\}.$$

Obviously, the answer from the current DB_3 database to the Q_4 -query contains only the tuple 16. However, according to the ontology O_3 , red apples are apples whose color is red. By exploiting this semantic information, a query can be reformulated to obtain more meaningful results. In fact, the tuple 13 meets the user's intention given by Q_4 . Therefore, an appropriate transformation of Q_4 can lead to the following new query:

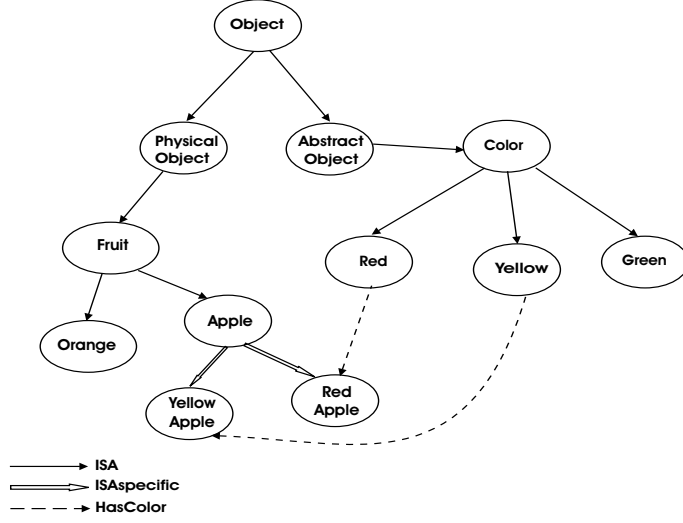


Fig. 3. Fruit Ontology

$$Q'_4 = \{(x_1, x_2, x_3, x_4) \mid (x_1, x_2, x_3, x_4) \in \text{GOODS} \wedge (x_2 = \text{"red apple"}) \vee (x_2 = \text{"apple"} \wedge x_3 = \text{"red"})\}.$$

Formally, let be $t_0 \in D$, $R \in DB$ and ξ be the set of domain relationship-types. The feature-rule is formulated as follows:

Rule 4:

IF $Q = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R \wedge x_p \theta t_0\}$

and $\exists c_0, c_p \in \zeta \mid \delta_3(t_0) = c_0 \wedge \delta_2(A_p) = c_p$

and $\exists A_i, \dots, A_j \in U(R) \mid \delta_2(A_k) = c_k \wedge \delta_4(A_p, A_k) = r_k \in \xi$

and $\exists c_q \in \zeta \mid c_q = R\text{parent}(\text{"ISAspecific"}, c_0)$

and $c_0 \in SUBT(c_p) \cap SUBT(c_k)$

THEN $Q' = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R \wedge [(x_p \theta t_0) \vee (x_p \theta t_p \bigwedge_{k=i}^j (\bigvee_{h=1}^m x_k \theta t_{kh}))]\}$

where $\delta_3(t_p) = c_q, t_{kh} \in I_0 \cup I_1 \cup I_2$, $m = |I_0 \cup I_1 \cup I_2|$

$I_0 = \{t \in D \mid \delta_3(t) = c_{kh}\}$

$I_1 = \{t \in D \mid \delta_3(t) \in DESC(\text{"ISA"}, c_{kh})\}$

$I_2 = \{t \in D \mid \delta_3(t) \in SYNs(c_{kh}) \vee \in SYNs(a), a \in DESC(\text{"ISA"}, c_{kh})\}$

$c_{kh} \in DESC(\text{"ISA"}, c_k) \cap RParent(r_k, c_0), i = < k = < j = < n,$

$k \neq p, 1 = < h = < m.$

6 Semantic Model and Rule Validations

Based on the rules defined, this section illustrates how the transformations of the proposed rules can be validated. To this end, we use a semantic model and

two consistency criteria, which are introduced in our previous paper [19]. For the sake of the limited space of the paper, we briefly describe this model and the related criteria. For more details the reader is referred to [19].

6.1 Semantic Model

The semantic model is stated as an extension of the given ontology, denoted by O^* , which includes new concepts and additional relationship-types. The new concepts represent relation names, attribute names and attribute values of the database unless they already exist. We denote these concepts by NC_{RN} , NC_{AN} and NC_V , respectively. Furthermore, we call *id-concepts* the concepts that represent id-values of the database. The additional relationships have to relating these concepts to the existing ones or to each other. Their types are defined as follows:

- "ValueOf" is the type of relationship that relates each value-concept to its associated attribute-concept.
- "HasA" is the type of relationship between relation-concepts and attribute-concepts.
- "InstanceOf" is the type of relationship that relates an Id-concept to its associated relation-concept.
- "TupleVal" is the type of relationship that relates value-concepts to each other, which are associated with a particular tuple.

Figure 4 shows a portion of the semantic model O_1^* related to the ontology O_1 and the database DB_1 (see section 4).

In summary, O^* is defined as a set $O^* = \{G^*, \zeta^*, \mathfrak{R}^*, N, T\}$, where $\zeta^* = \zeta \cup NC_{RN} \cup NC_{AN} \cup NC_V$, and $\mathfrak{R}^* = \mathfrak{R} \cup \{\text{"ValueOf"}, \text{"HasA"}, \text{"InstanceOf"}, \text{"TupleVal"}\}$.

We define two consistency criteria, *correctness* and *completeness*, which aim at asserting the soundness of this framework. Intuitively, correctness means that any results of the reformulated query, say Q' , can be "derived" in the extended ontology O^* i.e. the concepts and relationships corresponding to database objects in the results of Q' must be correctly represented in the model O^* . On the other hand, completeness ensures that any tuple that is "derived" in O^* for a given query Q' should also be in the answer of Q' i.e. the value-concepts together with their relationships corresponding to the results of Q' at the semantic level must be reflected in the database instance. More details are found in [19].

Logical Interpretation. By using the First Order Language (FOL) the semantic model O^* is defined as a theory T which consists of an Interpretation I and a set of well formed formulas [12]. I is specified by the set of individuals ζ^* and an interpretation function \cdot^I . In the following, we describe the interpretation of O_1^* .

Let n_1 and n_2 be the nodes of two concepts a and b , respectively. Formally, T :

$$I = (\zeta^*, \cdot^I)$$

$$ISA^I = \{(a, b) \in \zeta^{*2} | T(n_1, n_2) = \text{"ISA"}\}$$

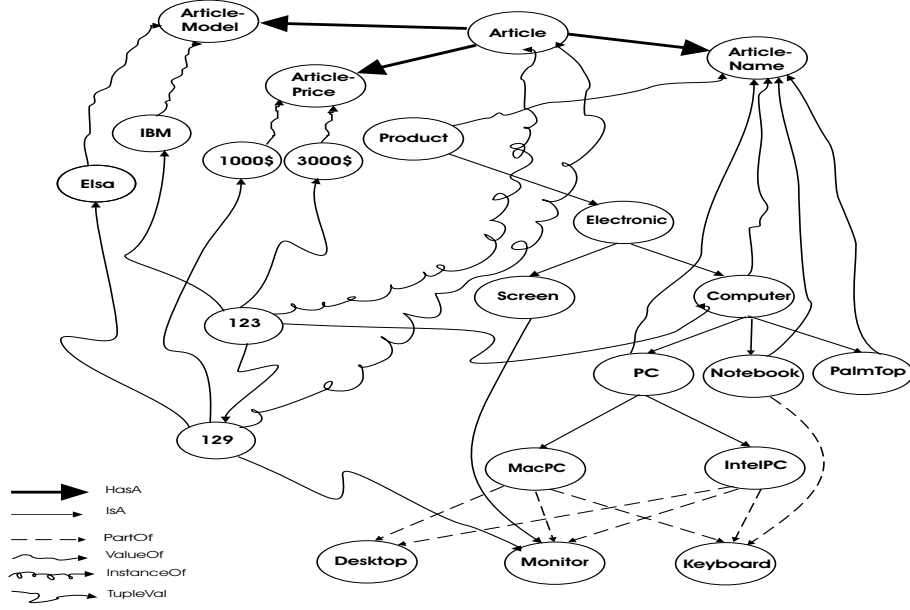


Fig. 4. A portion of the Semantic Model O_1^*

$$\begin{aligned}
SYN^I &= \{(a, b) \in \zeta^{*2} \mid T(n_1, n_2) = \text{"SynOf"}\} \\
PARTOF^I &= \{(a, b) \in \zeta^{*2} \mid T(n_1, n_2) = \text{"PartOf"}\} \\
HASA^I &= \{(a, b) \in \zeta^{*2} \mid T(n_1, n_2) = \text{"HasA"}\} \\
VALUEOF^I &= \{(a, b) \in \zeta^{*2} \mid T(n_1, n_2) = \text{"ValueOf"}\} \\
INSOF^I &= \{(a, b) \in \zeta^{*2} \mid T(n_1, n_2) = \text{"InstanceOf"}\} \\
Key^I &= \{a \in \zeta^* \mid \exists b. T(n_1, n_2) = \text{"InstanceOf"}\} \\
TUPVAL^I &= \{(a, b) \in \zeta^{*2} \mid T(n_1, n_2) = \text{"TupleVal"}\} \\
WHOLE^I &= \{a \in \zeta^* \mid \forall b_1 b_2 c. ISA(a, b_1) \wedge ISA(a, b_2) \wedge PARTOF(b_1, c) \rightarrow \\
&\quad PARTOF(b_2, c)\}
\end{aligned}$$

$$\begin{aligned}
&\forall x. ISA(x, x) \\
&\forall x. SYN(x, x) \\
&\forall x. PARTOF(x, x) \\
&\forall xyz. ISA(x, y) \wedge ISA(x, z) \rightarrow ISA(x, z) \\
&\forall x.y. SYN(x, y) \leftrightarrow SYN(y, x) \\
&\forall xyz. SYN(x, y) \wedge SYN(x, z) \rightarrow SYN(x, z) \\
&\forall xyz. ISA(x, y) \wedge SYN(y, z) \leftrightarrow ISA(x, z) \\
&\forall xyz. ISA(x, z) \wedge SYN(x, y) \leftrightarrow ISA(y, z) \\
&\forall xy \exists z. VALUEOF(x, y) \rightarrow HASA(z, y) \\
&\forall xy \exists z. TUPVAL(x, y) \rightarrow INSOF(x, z) \\
&\forall xyz. PARTOF(x, y) \wedge SYN(y, z) \leftrightarrow PARTOF(x, z) \\
&\forall xyz. PARTOF(x, y) \wedge SYN(x, z) \leftrightarrow PARTOF(z, y) \\
&\forall xyz. PARTOF(x, y) \wedge ISA(y, z) \leftrightarrow PARTOF(x, z) \\
&\forall xyz. PARTOF(x, y) \wedge PARTOF(x, z) \rightarrow PARTOF(x, z)
\end{aligned}$$

$$\begin{aligned}
&\forall xyz. VALUEOF(y, z) \wedge ISA(x, y) \rightarrow VALUEOF(x, z) \\
&\forall xyz. VALUEOF(y, z) \wedge SYN(x, y) \rightarrow VALUEOF(x, z) \\
&\forall xyz. \exists w. INSOF(x, y) \wedge HASA(y, z) \rightarrow TUPVAL(x, w) \wedge VALUEOF(w, z) \\
&\forall xyz. WHOLE(x) \wedge ISA(x, y) \wedge PARTOF(y, z) \leftrightarrow PARTOF(x, z) \\
&\forall xyz_1z_2. COMMONPART(x, y) \leftrightarrow ISA(x, z_1) \wedge ISA(x, z_2) \wedge PARTOF(z_1, y) \wedge \\
&\hspace{15em} PARTOF(z_2, y)
\end{aligned}$$

x, y, z, w, z_1, z_2 are variables

6.2 Rule Validation

We note that a common feature of the rules is that after applying a rule to a query Q , the results of the reformulated query might increase. We denote by S_Q and $S_{Q'}$ the result set of Q and Q' , respectively. This augmentation is not arbitrary but it is proved by the semantic model O^* . According to O^* , each tuple-identifier in S_Q is represented by an id-concept, which is related to value-concepts and a relation-concept through the `TupleVal` and `InstanceOf`-relationship, respectively. O^* interprets the reformulation results of a given rule as the existence of additional value-concepts, which are semantically related to those representing terms in the condition of Q . We call the id-concepts, which are related to the later ones, *virtual tuple-concepts* and the semantic relationship between them *DrivenTupleVal*. This type of relationship can be formally expressed by a predicate called `DrivenTUPLEVAL`. For brevity, we describe only an example of validation of the proposed rules using the available logical expressions from Γ .

Validation of Rule 1. Concerning this rule the S_Q -identifiers are formally expressed by the following set of individuals Ω_{B_1} , for which a formal W_1 is true. Formally,

$$\begin{aligned}
\Omega_{B_1} &= \{x \mid W_1(x)\} \\
&= \{x \mid TUPVAL(x, A_V) \wedge INSOF(x, R_N) \rightarrow VALUEOF(A_V, A_N)\},
\end{aligned}$$

where x is a variable, and A_V, A_N and R_N are constants.

The set of the virtual concepts provided by the rule transformation can be formally expressed by a set of individuals, denoted by Ω_{V_1} , for which a formal W'_1 is true. Formally, let z be a variable

$$W'_1(x) : \forall x \exists z \text{ DrivenTUPLEVAL}(x, y) \rightarrow TUPVAL(x, z) \wedge [ISA(A_V, z) \vee SYN(A_V, z)],$$

$$\begin{aligned}
\text{and } \Omega_{V_1} &= \{x \mid W'_1(x)\} \\
&= \{x \mid \exists z TUPVAL(x, z) \wedge [ISA(A_V, z) \vee SYN(A_V, z)]\}.
\end{aligned}$$

Therefore, if we unify the sets Ω_{B_1} with Ω_{V_1} , we obtain the set of individuals from Δ , which represents all id-concepts of the tuples in $S_{Q'}$. We denote this set Ω_1 . Formally,

$$\begin{aligned}
\Omega_1 &= \Omega_{B_1} \cup \Omega_{V_1} \\
&= \{x \mid \exists z TUPVAL(x, z) \wedge INSOF(x, R_N) \rightarrow VALUEOF(z, A_N) \wedge \\
&\hspace{15em} [ISA(A_V, z) \vee SYN(A_V, z)]\}.
\end{aligned}$$

7 Conclusion and Outlook

Recently, there is a growing interest in ontologies for managing data in database and information systems. In fact, ontologies provide good supports for understanding the meaning of data. They are broadly used in information integration systems to overcome problems caused by the heterogeneity of data and to optimize query processing among the distributed sources. In this paper, we use ontologies within single databases and present a new approach of query processing using semantic knowledge from a given ontology to reformulate a user query in such way that the query answer is more meaningful to the user. To this end, we propose a set of query reformulation rules and illustrate their effectiveness by some running examples. Although these rules might not be ideal, we hope that they can bring more insight into the nature of query answers. Furthermore, we use a semantic model and two basic criteria to prove the soundness of our approach. Based on this model we illustrate how the results of any syntactical reformulation of a query is semantically correct.

Our approach is appropriate for database applications, where some attributes are enumerated from a list of terms. For example, in product databases, items are described according to a collection of standard terms [24].

Currently, we develop additional reformulation rules and intend to address the problem of how to establish mapping information between the database objects and ontological concepts present in an ontology associated with a specific database.

References

- [1] Han, J.W., Huang, Y., Cercone, N., Fu, Y.J.: Intelligent query answering by knowledge discovery techniques. In: IEEE Trans. (1996) 373–390
- [2] Mena, E., Kashyap, V., Sheth, A., Illarramendi, A.: OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. Conference on Cooperative Information Systems **41** (1996) 14–25
- [3] Paton, N., Stevens, R., Baker, P., Goble, C., Bechhofer, S., Brass, A.: Query processing in the TAMBIS bioinformatics source integration system. Statistical and Scientific Database Management (1999) 138–147
- [4] Grant, J., Gryz, J., Minker, J., Raschid, L.: Semantic query optimization for object databases. ICDE (1997)
- [5] Lakshmanan, L., Missaoui, R.: On semantic query optimization in deductive databases. In: IEEE International Conference on Data Engineering. (1992) 368–375
- [6] D. Beneventano, S. Bergamaschi, C.S.: Description logics for semantic query optimization in object-oriented database systems. ACM Transaction on Database Systems. Volume 28: 1-50 (2003)
- [7] Aberer, K., Fischer, G.: Semantic query optimization for methods in object-oriented database systems. In: IEEE International Conference Data Engineering. (1995) 70–79

- [8] Hsu, C., Knoblock, C.A.: Semantic query optimization for query plans of heterogeneous multidatabase systems. *Knowledge and Data Engineering* **12** (2000) 959–978
- [9] Yu, C.T., Sun, W.: Automatic knowledge acquisition and maintenance for semantic query optimization. *IEEE Trans. Knowledge and Data Engineering* **1** (1989) 362–375
- [10] Sun, W., Yu, C.: Semantic query optimization for tree and chain queries. *IEEE Trans. on Data and Knowledge Engineering* **1** (1994) 136–151
- [11] Hsu, C.: Learning effective and robust knowledge for semantic query optimization (1996)
- [12] Peim, M., Franconi, E., Paton, N., Goble, C.: Query processing with description logic ontologies over object-wrapped databases. technical report, University of Manchester (2001)
- [13] Bergamaschi, S., Sartori, C., Beneventano, D., Vincini, M.: ODB-tools: A description logics based tool for schema validation and semantic query optimization in object oriented databases. *Advances in Artificial Intelligence*, 5th Congress of the Italian Association for Artificial Intelligence, Rome, Italy (1997)
- [14] Goni, A., Illarramendi, A.: An ontology connected to several data repositories: query processing steps. *Journal of Computing and Information (JCI)*, Volume 3, number 1, the 9th International Conference on Computing and Information (ICCI'98) (1998)
- [15] Gruber, T.: A translation approach to portable ontology specifications. In: *Knowledge Acquisition* (5) No. 2, USA. (1993) 199–220
- [16] Guarino, N., Giaretta, P.: Ontologies and knowledge bases: towards a terminological clarification. In: *Knowledge Building Knowledge Sharing*, ION Press. (1995) 25–32
- [17] Noy, N., Hafner, C.D.: The state of the art in ontology design. *AI Magazine* **3** (1997) 53–74
- [18] Chandrasekaran, B., Josephson, J., Benjamins, V.: What are ontologies, and why do we need them? In: *IEEE Intelligent Systems*. (1999) 20–26
- [19] Necib, C.B., Freytag, J.: Ontology based Query Processing in Database Management Systems. In: *Proceeding on the 6 th international on ODBASE 2003*. (2003) 37–99
- [20] Uschold, M., Grüninger, M.: Ontologies: principles, methods, and applications. *Knowledge Engineering Review* **11** (1996) 93–155
- [21] Kayed, A., Colomb, R.: Extracting ontological concepts for tendering conceptual structures. *Data and Knowledge Engineering* **41** (2001)
- [22] Ullman, J.: *Principles of Database and Knowledge-Base Systems*. Computer Science Press (1988)
- [23] Hernandez, M., Miller, R.J., Haas, L.M.: Clio: A semi-automatic tool for schema mapping. In: *ACM SIGMOD*. (2001)
- [24] Omelayenko, B.: Integrating vocabularies: Discovering and representing vocabulary maps. *The Semantic Web-ISWC 2002, First International Semantic Web Conference, Sardinia, Italy* (2002) 206–220