Ontology based Query Processing in Database Management Systems

Chokri Ben Necib and Johann-Christoph Freytag

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

Abstract

The use of semantic knowledge in its various forms has become an important aspect in managing data in database and information systems. In the form of integrity constraints, it has been used intensively in query optimization for some time. Similarly, data integration techniques have utilized semantic knowledge to handle heterogeneity for query processing on distributed information sources in a graceful manner. Recently, ontologies have become a "buzz word" for the semantic web and semantic data processing. In fact, they play a central role in facilitating the exchange of data between the several sources. In this paper, we present a new approach using ontology knowledge for query processing within a single relational database to extend the result of a query in a semantically meaningful way. We describe how an ontology can be effectively exploited to rewrite a user query into another query such that the new query provides additional meaningful results that satisfy the intention of the user. We outline a set of query transformation rules and describe by using a semantic Model the necessary criteria to prove their validity.

Keywords: Databases, Ontologies, Semantic Knowledge, Query Processing

1 Introduction

Semantic knowledge in its various forms including meta-models and integrity constraints is becoming an important aspect in managing data in database management and information systems: Semantic query optimization techniques have emerged in the 90s to complement the traditional approaches to reducing processing costs and to overcoming the heterogeneity problem in a distributed processing environment [7, 12, 3, 1]. Here, semantic rules about the data such as integrity constraints are the basis for reformulating user queries into more efficient, but semantically equivalent queries, which return the same answer in less time or with less resources. There are also several mechanisms in knowledge databases that use semantic knowledge from 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) as answers. These answers include some generalized, neighborhood or associated information that characterizes the data results [10]. Moreover, the intentional knowledge is stored in the database; thus the user can retrieve this additional knowledge as well.

In recent years, semantic knowledge in the form of *ontologies* has proven to be a powerful support for the techniques used for managing data. Ontologies promise solutions to the problems of semantic interoperability and data heterogeneity in querying distributed databases. An ontology might be used

to capture the semantic content of each source and unify the semantic relationships between their data structures such as the attribute properties and relation names. Thus, users should not care about where and how the data are organized in the sources. For this reason, systems like OBSERVER [16] and TAMBIS [20] allow users to formulate their queries over an ontology without directly accessing the data sources. Since the ontology defines the set of terms to be used in a query, the users must be familiar with the content of the ontology. However, using a large ontology to navigate and to select appropriate terms causes many difficulties. In our approach, the user does not have to deal with the ontology directly; he can formulate his queries over the database as usual. In this case, it is the responsibility of the query processor to reformulate the query using the ontology associated with that database.

On the other hand, ontologies might enhance the functionality of the search engines on the web by adding semantics to the information content of web pages. Ontologies are used to define the meaning of the terms emerging on the web pages and these can be used to make inferences to get more information related to the objects of interest [2].

In this paper, we present a new approach using ontologies for query processing within a single relational database management system. We assume that a preexisting ontology is associated with a database and provides the context of its objects. We show how an ontology can be exploited effectively to reformulate a user query such that the resulting query provides additional meaningful results meeting the intention of the user. A query can be defined by a set of projections over the objects satisfying a set of conditions. These conditions are defined by a set of terms and determine the answer. If a user wants to retrieve information from a database about a certain object, he might use terms, which do not exactly match the database values (due to the mismatch between the user's and the database designer's world views). However, there might be values in the database that are syntactically different from one another but semantically equivalent to the user terms and that express the same intention of the user. Wee address this issue as a semantic problem rather than as a pattern matching problem. We define two terms as semantically equivalent if they have the same meaning, i.e. they define the same concept with respect to the ontology. For example, if two terms are synonyms, they are semantically equivalent. As a result, 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 is further satisfying his expectations. For example, if two terms are synonyms, they are semantically equivalent. As a result, 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 is further satisfying his expectations. For example, let us assume that a user intends to query a database of products to get some information about the product "computer". Thus, the user will not obtain all the related instances from the database unless he know in advance that the database contains additional values that are semantically synonyms for "computer" such as "calculator" or "data processor". By considering these terms in the query the user will get more results from the database.

We use an ontology as a semantic layer over a database to describe the semantic relationships between the database values in order to transform user queries to other meaningful queries. To this end, a set of transformation rules must be developed taking into account possible mappings between the database and the ontology content. We assume the preexistence of an ontology associated with the database; but we point out its main features to fit the semantics of the database and assert the validity of such rules. Therefore, we develop a semantic model and basic criteria like correctness and completeness.

Our approach can be appropriate for the databases where some attributes are enumerated from a list of terms. For example, in product databases, the product items are described according to a collection of standard terms [19]. These terms are organized in taxonomies.

The remainder of this paper is organized as follows. In section 2 we discuss some preliminaries. In section 3, we present the problem by means of an example. In section 4, we illustrate our approach. In section 5, we describe a semantic model to validate this approach ,and in section 6 we reach our conclusion.

2 Preliminaries

2.1 Ontology

Nowadays, the term *"Ontology"* or *"ontologies"* is intensively used in artificial intelligence and information systems areas. However, there is no clear definition of what an ontology is. Often, we find in the literature definitions that are general or tailored according to the domain where the application is developed. The term *"Ontology"* is sometime used as a synonym for other terms such as *"Controlled Vocabulary"*, or *"Taxonomy"*, or *"Knowledge Base"*. This is due to the overlapping of some common features of these concepts. Since it is not the goal of this paper to discuss the definition of an ontology, we first give our own definition of this notion and then a short comparison with other similar notions. Readers, who are interested in the different meanings of "Ontology" are referred to [8, 9, 17, 4].

Informally, we define an ontology as an intentional description of what's known about the essence of the entities in a particular domain of interest using abstractions, also called *concepts* and their *relationships* . Basically, the hierarchical organization associated to the concepts through the inheritance (`"ISA"`) relationship constitutes the backbone of an ontology. Other kinds of relationship like the aggregation (`"PartOf"`) or Synonym (`"SynOf"`) or application specific relationships might exist.

The term "Controlled Vocabularies" (CVs) is commonly used in the field of linguistics, to mean a set of standardized terms with commonly agreed semantics for a specific domain within user communicate [14]. A special kind of a controlled vocabulary is a *thesaurus* . A common feature of an ontology and a thesaurus is that they contain a large set of special terms concerning a certain domain and provide a clear understanding of their meanings. Furthermore, both an ontology and a thesaurus use relationships among the terms to represent these meanings. However, most of the relationships used in a thesaurus are different from those used in an ontology. In addition, they are usually ambiguous and less specified. For example, the relationships Broader Term (BT) and Narrower Term (NT) indicating that a term has broader meaning than another term and vise versa, indicate sometimes the specialization and the part-whole aspects at the same time [21]. Moreover, such inverse relationships are not explicitly represented in an ontology. Finally, a thesaurus deals with terms whereas an ontology deals with concepts but uses terms to represent these concepts. In general, a concept is not a word and it is not specific to a given natural language [13]. Thesaurus are dependent upon a specific natural language (or multiple language in case of Multilanguage thesaurus).

The term "Taxonomy" refers the classification of entities, whether they are terms or objects, in a hierarchical structure according to the sub/super class paradigm. Thus, there is only one type of relationship relating these entities, namely the ISA-relationship. For this reason, if we reduce the types of relationships in an ontology to only the ISA-types to represent concepts, the ontology will be equivalent to a taxonomy.

Moreover, the use of the term "Ontology" can be confused with the use of the term "Knowledge Base". A knowledge base for the AI-community consists of two parts: A terminology Box, called *"T-Box"* and an assertions Box, called *"A-Box"* . The T-Box comprises a set of concepts and their definitions. It includes usually a taxonomy of terms related to each other with `ISA`-relationships. The A-Box comprises a set of instances of these concepts, called the universe of discourse, and a set of assertions between them. The common feature of Ontologies and knowledge bases is that both represent knowledge. However, knowledge bases provide in addition instances, for which knowledge is applied and inferred. Thus, if we reduce a knowledge base to the T-Box, we can say that the an ontology is equivalent to the resulting knowledge base.

"What does an ontology look like?" and "How can it be created?" still remain struggling topics for researchers but what they all agree upon is that an ontology must play the following role: An ontology must provide knowledge in the form of concise and unambiguous concepts and their meanings. This knowledge can be shared and reused from different agents i.e. human or/and machines.

2.2 Graphical Representation of an Ontology

In this section, we introduce a graph based representation of an ontology and set the associated graph operations. We agree that The graphical representation is more appropriate than the text based one found in the literature [13]. This representation conveys the properties of an ontology in a simple, clear and structured model.

Formal representation. Formally, we define an ontology as a set $\zeta = \{c_1, \ldots, c_n\}$ and a set $\Re = \{$"`ISA`","`SynOf`", "`PartOf`"$\}$ where $c_i \in \zeta$ is a concept name, and $r_i \in \Re$ is the type of the binary relation relating two concepts (c_i and r_i are non-null strings). Other domain specific types can also exist. 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 [24]. Further, we consider that the terms "concept" and "concept name" have the same meaning.
We represent an ontology as a directed graph $G(V, E)$ (DAG) where V is a finite set of vertices and E is a finite set of edges: Each vertex of V is labeled with a concept and each edge of E represents the relation between two concepts. Formally, the label of a node $n \in V$ is defined by a function $N(n) = c_i \in \zeta$ that maps n to a string from ζ. The label of an edge $e \in E$ is given by a function $T(e)$ that maps e to a string from $\Re$.
Finally, an ontology is given by the set $O = \{G(V, E), \zeta, \Re, N, T\}$.

Figure 1 gives an example of a graph representation of a selected portion from the ontology "Product". A part of this ontology is adopted from an ontology described in [11].

Graph operations. In order to navigate the ontology graph, we define the following primitive operations: ISAChild, PartOfChild, ISAParent, and PartOfParent and two sets of concepts: $DESC$ and $SYNs$. We need these operations and sets to identify nodes in the graph, which hold concepts that are of interest for a query manipulation.

Given two nodes $n_1 = node(c_1)$ and $n_2 = node(c_2)$

- n_2 = ISAChild(n_1) iff n_2 = child(n_1) and T[(n_1,n_2)] = "`ISA`"
- n_2= PartOfChild(n_1) iff n_2= child(n_1) and T[(n_1,n_2)] = "`PartOf`"
- n_2 = ISAParent(n_1) iff n_2= parent(n_1) and T[(n_1,n_2)] = "`ISA`",
- n_2 = PartOfParent($n1_1$) iff n_2= parent(n_1) and T[($n1$,n_2)] = "`PartOf`"
- n_2= SynOf(n_1) iff T[(n_1,n_2)] = "`SynOf`"
- $DESC(r, c) = \{s \in \zeta \mid \forall e \in E \wedge e \in P(node(c) - node(s)) \wedge T(e) = r\}$
- $SYNs(c) = \{s \in \zeta \mid \forall e \in E \wedge e \in P(node(c) - node(s)) \wedge T(e) =$ "`SynOf`"$\}$

Informally, $DESC(r, c)$ gives the set of all concepts in O obtained by retrieving recursively all the labels of the children nodes related with the node of c by following only the links of type r. Similarly, $SYNs(c)$ gives the set of all synonyms of c in O. We denote by $P(n_1 - n_2)$ the directed path between two nodes n_1 and n_2.

3 Motivation and Problem Statement

Data semantics, as defined in [22], is *the meaning of data and the reflection of the real world*. Since designers perceive the real world differently, there exist more than a single way to represent the existing objects and their relationships. The real world objects might have complex structures and dynamic behaviors. Thus, capturing the semantics completely from the real world seems to be impossible i.e. there does not exist any model which can define all the aspects of the real world objects. For example, relational database systems overcome the limitations of the relation model by adding a set

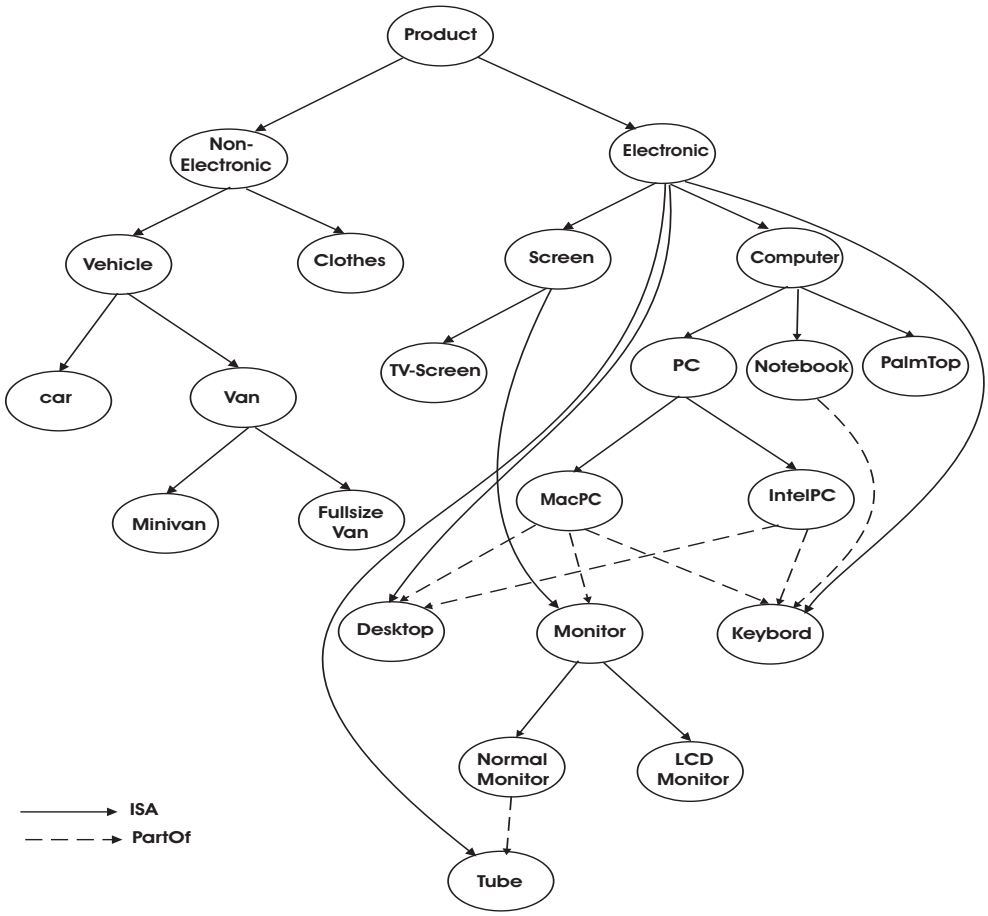

Figure 1: Product Ontology

of integrity constraints to maintain data that is consistent and to provide support for semantic rules such as cardinality, containment, and type hierarchies [18].

We believe that a new generation of DBMSs requires additional semantic supports for a flexible and efficient data management. This includes facilities for data integration, query optimization and meaningful query processing. The later problem is addressed in our paper. The basic idea, is to give the DBMS the ability to deal with the queries both at the semantic as well as the syntactic level. In fact, if a user attempts to retrieve information about objects from the database, the answer to his query might not meet his expectations. This might be one to the following reasons:

First, there might be semantic ambiguities between the terms used in the query and the database values that represent these objects (vocabulary problems). In fact, the user's perception of real world objects might not match exactly that of the database designer. Second, there might be different ways to formulate the query using semantic equivalent terms. We define two sets of terms to be equivalent if their relevant concepts and relationships in the ontology identify the same concept. There might be several such sets. We will specify this definition in our future work. Therefore, when the user formulate his query, he might use terms cover partially these semantics. Third, some results in the answer might not be related to the same context associated with the query. The context must be defined by the user.

Now, we give an example that can illustrate these reasons and our ideas throughout the paper:
We consider the ontology 'Product' given in figure 1, denoted by O_1. This ontology describes several products. We assume that we have a simple relational database, denoted by DB_1, including 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(AID, Name, Model, Price)
AID: Article identifier
Name: Name of the article
Model: Model of the article
Price: Price of the article
PrimaryKey(AID)

COMPONENT (S-Part-ID, M-Part-ID)
M-Part-ID: Main part identifier
S-Part-ID: Second part identifier
Foreign-Key(M-Part-ID) TO ARTICLE
Foreign-Key(S-Part-ID) TO ARTICLE
Primary-Key(S-Part-ID)

Suppose, at present, that DB_1 contains the following instances as shown in the tables 1 and 2.

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-Part-ID	M-Part-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

When a user want to retrieve information about computers from DB_1, he may submit a query that looks like

```
Q1:  SELECT * FROM article WHERE name ='computer'.
```

In this query, the user intention concerns the object "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 the general 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" . The database designer might use any of the specialized terms to specify the article "computer". If the user does not know these semantics in advance, he will not obtain all the relevant instances from DB_1. Thus, a meaningful processing of Q_1 has to transform the user query by adding these terms to the query condition. As consequence, the result of the transformed query might be greater than the result of the previous query and satisfy more of the user's needs. Note that without a semantic support, like the ontology, it is hard for the DBMS query processor to solve such vocabulary ambiguities. In this case, the ontology provides additional meanings for the database values related to a certain attribute name. These meanings are expressed through the relationships between the corresponding concepts. Now, suppose the user wants to get information about the article "PC". His query may look like

```
Q2:  SELECT * FROM article WHERE name = 'PC'.
```

In this query, the user's intention concerns the object "PC". According to the ontology O_1 the concept "PC" is specialized into the concepts "MacPC" and "IntelPC". In addition, following the PartOf-links, a "PC" has three parts: a "desktop" , a "monitor" and " a "keyboard". If we assume, that

all the PC-objects in the database must be composed exactly of these parts and that there do not exist any other objects composed of these, then we can find another way to characterize the same object "PC" by means of its components. With regard to the definition mentioned earlier, we can say that the terms "desktop", "monitor" and "keyboard" can build a new query condition which is semantically equivalent to the condition of Q_2. Therefore, it is not surprising that the tuples 123 and 128 with attribute names "computer" and "product" meet fully the intention of the user. When a user poses a Q_2-query to the database DB_1, these tuples will certainly be missed. The DBMS query processor has to extend the user query considering these semantics in addition to those suggested for the previous query Q_1. Note that in this case the number of tuples in the answer result will also increase. These examples clearly show which problems we are intended to solve:

"How can database queries be transformed using an ontology?"
"How can these transformations be validated?"
"How should the ontology look with respect to the database?"

Formally, the problem can be stated as follows:
Given a database DB, an ontology O and a user query Q, how to find a rewriting Q' of Q by using O, such that Q' returns to the user possibly more or fewer meaningful results than Q.

4 Ontology based Query Processing

We address the problem above and choose the Domain relational calculus (DRC) formalism 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 collection of operators to be defined shortly:
The atoms of formulas $\psi(s)$ are of three types: $u \in R$, $v\ \theta\ w$, and $u\ \theta\ t$, where R is a relation name and u, v and w are tuple variables, and t is a constant. θ is an arithmetic comparison operator ($=$, $<$ and so on). An occurrence of a variable in a formula can be bound or free. Formulas and variables in Q can be also defined recursively using the logic operators "$\wedge$" and "$\vee$ (see [23])

Our approach consists of three phases: preprocessing, execution and post-processing phase.
In the preprocessing phase, we transform a user query into another one based on new terms extracted from the ontology associated with this database. To this end, a set of transformation rules and of mapping functions must exist. The mapping functions have to map database types such as relation names, attribute names and data values to the concepts in the ontology. The transformation rules must contribute to:

- Expand the queries by changing their select condition α using terms synonymous with t and the terms specifying its concept. To achieve this purpose, the ontology must have the capabilities of reasoning over the synonym and hyponym relations, and
- Substitute the query conditions with other conditions that are semantically equivalent. Building such rules with respect to the database and the ontology is not an easy task.

However, the rules should not be developed according to ideal alone but they must lead to results closer to the user expectations. In addition, we need a semantic model that can reflect semantics of the transformations made at the syntax level. This might assert the validity of these rules. In the next section we propose this model. In the execution phase, the transformed query is processed using available query strategies of the system and the result is checked. If the result is empty, we perform the third phase. In the post-processing phase, we attempt to generate a more generalized answer to the query. That is, for each mapped attribute in the query condition three steps are performed:

(1) We substitute its value with the term of the corresponding parent concept by ascension of the ontology relations one level

(2) We execute the query again. If the answer set is still empty then we continue the substitution process in step 1 using much higher level concepts.

(3) We repeat step 2 until we find tuples in the answer set or no substitution is possible i.e. we achieve the root concept node.

5 Semantic Model

Given an ontology $O = \{G(V, E), \zeta, \Re, N, T\}$ and a database DB. Let U be a set of the attributes $A_1, \ldots, A_n$ where each attribute domain, $dom(A_i)$, is considered finite. The database schema is defined as a set of relation schema R_1 ,...,R_m where the attributes of each relation schema belong to U. The set of attributes that are primary keys is denoted by $PRIMKEY(U)$. Let also $U(R_i)$ be the set of attributes of U that define the schema of R_i. In addition, let M_1 be defined as the function that represents the mapping of the relation names into the concepts in O, called *relation-concepts*; let M_2 be the function that represents the mapping of attribute names into the concepts in O, called *attribute-concepts*, and let M_3 be the function that represents the mapping of the attribute values into concepts in O, called *value-concepts*.

We propose a semantic model that enables us to validate the syntactic transformations of the database queries proposed earlier. This model is defined as a particular ontology, denoted O^*,which is an extension of the original ontology O. Additional concepts and relationships are introduced as specified below. Note that there is no single consensual methodology for the ontology design [15]. Thus, we would not restrict our ontology development to particular guidelines for the building of the ontology. In fact, the purpose of this work is not to discuss how to develop or integrate an ontology. These issues alone are challenging problems for the researcher. For more interests to these topics, readers are recommended to refer to [6, 24].

Additional Concepts

For each relation R of the database new concepts are created to represent the relation name, its attributes (expect the primary key attribute) and their domain attribute values unless such concepts already exist in O. In this case, we adopt the following naming conventions for the concepts:

(1) No single term can name two different concepts.

(2) The name of an attribute-concept is prefixed by the name of its relation concept.

These conventions are defined not only to make the extension of the ontology easier, but also to help avoid ambiguities in distinguishing concepts. We define the Id-concepts as the set of value-concepts that represent the values of the primary-key attributes. We denote by ID the set of nodes labeled by Id-concepts. We denote also the new set of relation-concepts, attribute-concepts, and value-concepts as C_{RN}, C_A and C_V respectively.

Additional Relationships

Because new concepts might be created as described above, one needs to link them with the existing concepts and/or with each other. To this end, additional binary relationships holding new semantics are introduced. The types of these relationships are defined as:

- `ValueOf`:
  This is the type of the relationship that relates each value-concept to the attribute-concept. Formally, $\forall A_i \in U \setminus PRIMKEY(U), \forall w \in dom(Ai), T(node(M_3(w)), node(M_2(A_i))) =$ "`ValueOf`". Note that the `ValueOf`-relationship has nothing to do with Id-concepts.
- `HasA`:
  This is the type of the relationship between the relation-concepts and the attribute-concepts. Formally, $\forall\ A_i \in R_i,\ R_i \in DB,\ T(node(M_1(R_i)), node(M_2(A_i))) =$ "`HasA`".

- `InstanceOf`:

  This is the type of the relationship that relates the Id-concepts to their corresponding relation-concepts so that $\forall A_i \in PRIMKEY(U)$, $A_i \in R_i$ and $id \in dom(A_i)$, $T(node(M_3(id)), node(M_1(R_i))$= `"InstanceOf"` .

- `TupleVal`:

  `TupleVal` is defined as the type of relationship relating the concepts associated with the attribute values of each tuple. This relationship is represented in the graph of O^* as a directed arc going out from the Id-concept node to other value-concept nodes associated to each attribute value of a given tuple.
  Formally, given a tuple $\mu \in R_i$, $\mu : U \rightarrow dom(U)$, $\forall\ A_i \in U(R_i)$, and $ID \in PRIMKEY(R_i)$, $T(node(M_3(\mu(ID)), node(M_3(\mu(A_i))) =$ `"TupleVal"`.

Summary

O^* is defined as $O^* = \{G^*(V,E), \zeta^*, \Re^*, N, T\}$, where $\zeta^* = \zeta \cup C_{RN} \cup C_A \cup C_V$,
and $\Re^* = \Re \cup \{$"`ValueOf`","`HasA`","`InstanceOf`","`TupleVal`"$\}$

Figure 2 describes the semantic model for the product ontology. For the sake of a good visibility, we reduce the graph to some nodes and edges.

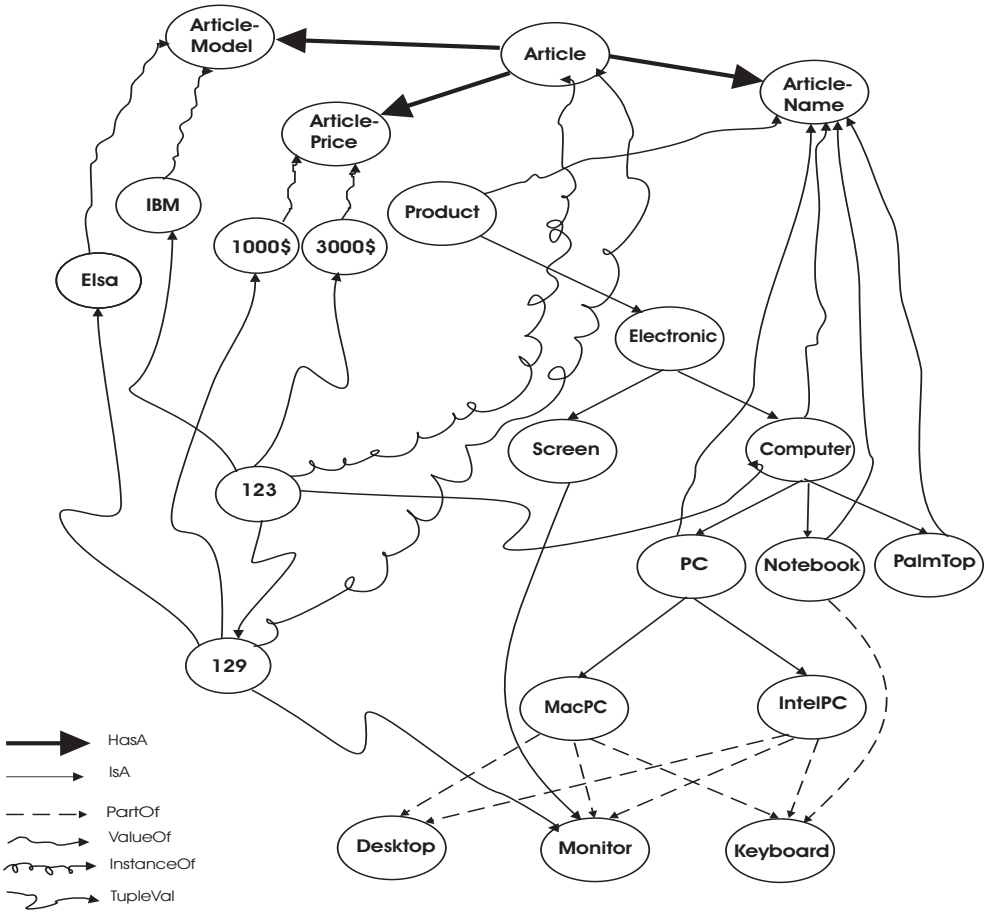

Figure 2: A portion of the Semantic Model for Product Ontology

Definitions

For the ease of semantic reasoning on the concepts in the ontology graph of O^*, we introduce the graph operator: *SelectRel* . This operator will be used in the following sections.

SelectRel Operator:

This operator returns all edge types of the path between two value-concept nodes in G^* that are connected with two other id-nodes via edges of type "TupleVal".
Semantically, if two id-nodes are adjacent (there is a common edge of type "TupleVal") then the semantic relationship between the represented concepts can be deduced from the result of the SelectRel operation on these nodes. We assume that the PartOf-relationship semantically dominates the "ISA" one. This means that, if a path between two nodes consists of edges of types "ISA" and "PartOf" then the semantic relationship between the concepts, which label these nodes, is of type "PartOf". For example, if the semantic relationship between two database instances is of part-whole type then there exist in O^* a "PartOf"-relationship between two value-concepts, which are related to id-concepts of the tuple- identifier. In other word, the part-whole semantic between two database tuples will be reflected in the semantic model through relevant concept-values. Thus, if we want to know which semantics relate the tuple 123 and the tuple 123, we have to operate SelectRel on their corresponding id-nodes. As a result, we get two types of relations "ISA" and "PartOf". Due to the assumption above, we conclude that the object identified by 129 is part of that object identified by 123.

Formally, let id_1, $id_2 \in ID(G^*)$

$SelectRel(G^*, id_1, id_2)$= $\{R_i \in R \,|\, R_i = T(x,y) \wedge \exists\, n_1, n_2 \in V(G^*) \wedge T(id_1, n_1) =$ "TupleVal" $\wedge T(id_2, n_2) =$ "TupleVal" $\wedge\ [(x,y) \in P(n_1 - n_2) \vee (x,y) \in P(ISAChild(n_1) - ISAChild(n_2))]\}$.

We denote by $|SelectRel_{PartOf}(G^*, id_1, id_2)|$ the number of "PartOf"-labels returned by the *SlectRel* operator.

5.1 Logical Interpretation of the Model

In this section, we want to express the semantic model in a logical framework using the First Order Language (FOL) [5]. The later representation will be useful for formulating the criteria related to our semantic model. From a logical point of view, O^* is a theory Γ, which consists of an Interpretation I and a set of well formed formulas. I is defined by the set of individuals Δ, called universe of discourse, and an interpretation function $\cdot^I$.

Formally, Γ:

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

$ISA^I = \{(a,b) \in \Delta^2 | T(node(a), node(b)) =$ "ISA"$\}$
$SYN^I = \{(a,b) \in \Delta^2 | T(node(a), node(b)) =$ "SynOf"$\}$
$PARTOF^I = \{(a,b) \in \Delta^2 | T(node(a), node(b)) =$ "PartOf"$\}$
$HASA^I = \{(a,b) \in \Delta^2 | T(node(a), node(b)) =$ "HasA"$\}$
$VALUEOF^I = \{(a,b) \in \Delta^2 | T(node(a), node(b)) =$ "ValueOf"$\}$
$INSTANCEOF^I = \{(a,b) \in \Delta^2 | T(node(a), node(b)) =$ "InstanceOf"$\}$
$TUPLEVAL^I = \{(a,b) \in \Delta^2 | T(node(a), node(b)) =$ "TupleVal"$\}$
$WHOLE^I = \{a \in \Delta | \forall b_1 b_2 c.\ ISA(a,b_1) \wedge ISA(a,b_2) \wedge PARTOF(b1,c) \rightarrow PARTOF(b2,c)\}$
$HASPART^I = \{a \in \Delta | \forall b \exists p.\ ISA(a,b) \rightarrow PARTOF(b,p)\}$
$Key^I = \{a \in \Delta | \exists b. T(node(a), node(b)) =$ "instanceOf"$\}$

$\forall x.\ ISA(x,x)$
$\forall x.\ SYN(x,x)$
$\forall x.\ PARTOF(x,x)$
$\forall xyz.\ ISA(x,y) \wedge ISA(x,z) \rightarrow ISA(x,z)$

$\forall x.y\ SYN(x,y) \leftrightarrow SYN(y,x)$
$\forall xyz.\ SYN(x,y) \wedge SYN(x,z) \rightarrow SYN(x,z)$
$\forall xyz.\ SYN(x,y) \wedge SYN(x,z) \rightarrow SYN(x,z)$
$\forall xyz.\ PARTOF(x,y) \wedge PARTOF(x,z) \rightarrow PARTOF(x,z)$
$\forall xy\ \ \exists z.\ TUPLEVAL(x,y) \rightarrow INSTANCEOF(x,z)$
$\forall xy\ \exists\ z.\ VALUEOF(x,y) \rightarrow HASA(z,y)$
$\forall xyz.\ VALUEOF(y,z) \wedge ISA(x,y) \rightarrow VALUEOF(x,z)$
$\forall xyz.\ VALUEOF(y,z) \wedge SYN(x,y) \rightarrow VALUEOF(x,z)$
$\forall xyz.\ \exists\ w.\ INSTANCEOF(x,y) \wedge HASA(y,z) \rightarrow TUPLEVAL(x,w) \wedge VALUEOF(w,z)$
$\forall xyz.\ ISA(x,y) \wedge SYN(y,z) \leftrightarrow ISA(x,z)$
$\forall xyz.\ ISA(x,z) \wedge SYN(x,y) \leftrightarrow ISA(y,z)$
$\forall xyz.\ PARTOF(x,y) \wedge SYN(x,z) \leftrightarrow PARTOF(z,y)$
$\forall xyz.\ PARTOF(x,y) \wedge SYN(y,z) \leftrightarrow PARTOF(x,z)$
$\forall xyz.\ PARTOF(x,y) \wedge ISA(y,z) \leftrightarrow PARTOF(x,z)$
$\forall xyz.\ WHOLE(x) \wedge ISA(x,y) \wedge PARTOF(y,z) \leftrightarrow PARTOF(x,z)$
$\forall xy\ COMMONPART(x,y) \leftrightarrow \forall z_1 z_2 ISA(x,z_1) \wedge ISA(x,z_2) \wedge WHOLE(z_1) \wedge WHOLE(z_2) \wedge PARTOF(z_1,y) \wedge PARTOF(z_2,y)$

5.2 Correctness and Completeness Criteria

We define two criteria, *correctness* and *completeness*, for the validation of the transformation rules. The basic idea underlying these criteria is that, if we reflect any syntax transformation of the query on the semantic level it must be correct i.e. it will not violate the semantic model. Symmetrically, the semantic model is defined as complete if the mapping of concepts, which represent the result of the transformed query, are reflected by the database i.e. the corresponding values in the database are stored consistently.

To define formally these criteria, we need the following preliminary definitions:

Definition 1: An attribute A of U is said to be *covered* by O, if each value of its domain is represented by a concept in O.
Formally, $\forall\ x \in dom(A), \exists\ c \in \zeta \mid M_3(x) = c$

Definition 2: A relation R is said to be *partially covered* by O, if there exist an attribute A of R which is covered by O.

Definition 3: Two id-concepts id_1 and id_2 are said to be *semantically dependent* if and only if $SelectRel(G^*, node(id_1), node(id_2)) \neq \emptyset$

Correctness Criterion

Formally, An extended ontology O^* is said to be a correct model for a relation R if and only if:
$\forall id_1, id_2 \in dom(ID)$, $ID \in PRIMKEY(R)$, and $ic_1 = M_3(id_1)$ and $ic_2 = M_3(id_2)$

(1) IF $T(node(ic_1), node(ic_2)) =$ "`TupleVal`"
THEN ic_1 and ic_2 are semantically dependent,
and

(2) IF $|SelectRel_{PartOf}(G^*, node(ic_1), node(ic_2))| \neq \emptyset$
THEN $|SelectRel_{PartOf}(G^*, node(ic_1), node(ic_2))| = 1$

The intuition behind the first condition, is that if two database tuples are related to each other, then there exist in O^* at least one semantic relationship between the two value-concepts associated to two attribute values of the tuples. For example, if we examine the semantic model of the product ontology (see figure 2), then we deduce that the relation between the tuples 123 and 129 (see relation

component) is reflected by the semantic relationship of the concepts "computer" and "monitor". That is, the object 129 is part of the object 123.

The intuition behind the second condition, is that only a PartOf-relation level is allowed for all the database instances i.e. if item A is part of item B and item B is part of item C than the database does not store explicitly the relation: Item A is part of item C.

Completeness Criterion

The extended ontology O^* is *complete* if and only if:

(1) $\forall id_1 a_v p.\ Key(id_1) \wedge TUPLEVAL(id, a_v) \wedge WHOLE(a_v) \wedge PARTOF(a_v, p) \rightarrow \exists\ id_2 Key(id_2) \wedge TUPLEVAL(id1, id_2) \wedge TUPLEVAL(id_2, p)$,

and

(2) $\forall id_1 a_v p\ \exists id_2\ Key(id_1) \wedge\ TUPLEVAL(id, a_v) \wedge HASPART(a_v) \wedge COMMONPART(a_v, p) \rightarrow Key(id_2) \wedge TUPLEVAL(id_1, id_2) \wedge TUPLEVAL(id_2, p)$

are satisfied.

Axiom (1) denotes that each decomposition of a concept in the ontology must reflect the same decomposition for the associated values in the database instance. In this case, the decomposition is said to be mandatory for the database instances. For example, each instance of the Database DB1 where the article name is "PC" should have a "desktop", "monitor" and "keyboard" instance. In addition, the condition asserts when the PartOf-relationship is transitive with respect to the ISA-relationship. A concept, say B, is a part of a concept, say A, if B is a part of all the sub-concepts of A. For example, the concept "monitor" is a part of the concept "PC" because it's a part of both concepts "MacPC" and "IntelPC", which are sub-concepts of "PC".

Axiom (2) denotes that if all the sub-concepts of a concept, say A, have a common part concept, say P, then each database instance reflecting A must be minimally related to an instance, which reflects P. For example, suppose that the concept "palmtop" does not exit in the ontology "Product". Thus, for each tuple of the database where the article name is "computer" must be related to another tuple where the article name is "keyboard".

Summary

Based on the criteria above a database instance is *consistent* with respect to an ontology O if

(1) O^* is a correct model for the database, and

(2) O^* is a complete model for the database.

5.3 Example of a Transformation Rule

Now, we present a possible transformation rule and illustrate its validation using our proposed semantic model. This will help illustrating the basic ideas of our approach in this paper.
intuitively, this rule derive terms from the ontology, which are synonymous with terms used in the query conditions and other terms that specialize them. The query example Q_1 in the section 3 is related to this rule.

Formally, let D be the set of domain attributes of a database, $t_0 \in D$, and $c_0 \in O$.

IF $\quad Q = \{(x_1, \ldots, x_n)\ \mid (x_1, \ldots, x_n) \in R \wedge x_i \theta t_0\}$ and $M_3(t_0) = c_0$

THEN $\quad Q' = \{(x_1, \ldots, x_n)\ \mid (x_1, \ldots, x_n) \in R \wedge\ [(x_i \theta t_0) \vee (x_i \theta t_1) \vee \ldots \vee (x_i \theta t_m)]\}$

where $t_k \in I_0 \cup I_1, 1 \leqq k \leqq m = |I_0 \cup I_1|$
$I_0 = \{t \in D \mid M_3(t) \in DESC_{isA}(c_0)\}$, and
$I_1 = \{t \in D \mid M_3(t) \in SYNs(c),\ c \in I_0\}$

We note that this rule might increase the result set, say S_Q, provided by Q. This augmentation is not arbitrary but it is proved by the semantic model O^* associated with the database: According to O^*, the tuple identifier of S_Q are represented by id-concepts, which are linked with value-concepts, and the relation-concept through `TupleVal` and `InstanceOf`-relationship, respectively. Formally, this set is given by the following:

$$\Omega_{BT} = \{x \mid W(x)\}$$
$$= \{x \mid TUPLEVAL(x, A_V) \wedge INSTANCEOF(x, R_N) \rightarrow VALUEOF(A_V, A_N)\},$$

where x is a variable and A_V, A_N, and R_N are constants. O^* interprets the rule as the existence of additional value-concepts, which are semantically related to those representing terms in the condition of the query Q. We call the id-concepts, which are related to the later ones *virtual tuple-concepts* and the semantic relationship between them *DrivedTupleVal*. Formally, this type of relationship can be expressed by a predicate $DRIVEDTUPLEVAL$ as follows:

$$W'(x) : \forall x\, \exists\, z\, DRIVEDTUPLEVAL(x, y) \rightarrow TUPLEVAL(x, z) \wedge [ISA(A_V, z) \vee SYN(A_V, z)].$$

We denote by Ω_{VT}, the set of virtual tuple-concepts and express it as follows:

$$\Omega_{VT} = \{x \mid \exists\, z\, DRIVEDTUPLEVAL(x, z)\}.$$

As a result, if we unify the sets Ω_{BT} with Ω_{VT}, we get then a set of individuals from Δ, which represents id-concepts of the result of the query Q'. We denote this set by Ω.
Formally,

$$\Omega = \Omega_{BT} \cup \Omega_{VT}$$
$$\Omega = \{x \mid \exists\, z\, TUPLEVAL(x, z) \wedge INSTANCEOF(x, R_N) \rightarrow VALUEOF(z, A_N) \wedge [ISA(A_V, z) \vee SYN(A_V, z)]\}.$$

6 Conclusion and Outlook

Today, Database management systems face challenging problems in dealing with the huge amount of data and the variety of its format. Thus, current database systems not only need additional supports for manipulating data but also for understanding its meaning. Semantic knowledge in its various forms become a necessary tool for enhancing the usefulness and flexibility of data management, especially in integrating data from multiple sources and in optimizing the queries. In fact, this makes the database aware of the semantics of its stored values and thus provides better ways to answer a query request. Conventional database querying does not always provide answers to users, which fully meet their expectations. One of the reasons, is that the query is treated at only the syntactical level.
In this paper, we have presented an approach for the query processing that processes the query at both the syntactical and the semantical level. Our approach allows to generate answers, which contain enough informative and meaningful results for the user. We use the ontology as a semantic tool for processing data in a single database management system. We have showed how can we capture semantics between database objects and use them for reformulating the user queries. We have outlined the basic features of the rules that allow these reformulations. Then we presented a semantic model and the basic criteria to validate any transformations made at the syntactical level.
Our approach can be appropriate for the databases where some attributes are enumerated from a list of terms. For example, in product databases, the product items are described according to a collection of standard terms [19].
Currently, we are developing a set of transformation rules for use in relational database systems. Although these rules might not be ideal, we hope that they can bring more insight into the nature of

query answers. We believe that using ontologies for managing data will provide meaningful information to answer a database query.
In the future, we will investigate how to use ontologies to generate knowledge answers which are compact and intuitive for the user and describe the characteristics of the query results.

References

[1] K. Aberer and G. Fischer. Semantic query optimization for methods in object-oriented database systems. In *IEEE International Conference Data Engineering*, pages 70–79, 1995.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web, a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. In *Scientific American*, 2001.

[3] U. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. In *ACM Transactions on DatabaseSystems*, pages 162–207, 1990.

[4] B. Chandrasekaran, J.R. Josephson, and V.R. Benjamins. What are ontologies, and why do we need them? In *IEEE Intelligent Systems*, pages 20–26, 1999.

[5] E. Franconi. Description logics. Course at the International Doctoral school of Technology and Science at the Aalborg University, Denmark, 2002.

[6] A. Gomez-Perez, M. Fernandez-Lpez, and O. Corcho. *Ontological Engineering*. Springer Verlag, London Ltd, 2003. To be published.

[7] J. Grant, J. Gryz, , J. Minker, and L. Raschid. Semantic query optimization for object databases. ICDE, November 1997.

[8] T.R Gruber. A translation approach to portable ontology specifications. In *Knowledge Acquisition (5) No. 2, USA*, pages 199–220, 1993.

[9] N. Guarino and P. Giaretta. Ontologies and knowledge bases: towards a terminological clarification. In *Knowledge Building Knowledge Sharing,ION Press*, pages 25–32, 1995.

[10] J. W. Han, Y. Huang, N. Cercone, and Y. J. Fu. Intelligent query answering by knowledge discovery techniques. In *IEEE Trans*, pages 373–390, 1996.

[11] AA. Kayed and R.M. Colomb. Extracting ontological concepts for tendering conceptual structures. *Data and Knowledge Engineering*, 41(1-4), 2001.

[12] L.V.S. Lakshmanan and R. Missaoui. On semantic query optimization in deductive databases. In *IEEE Inter-national Conference on Data Engineering*, pages 368–375, 1992.

[13] D.B. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, Reading, Massachusetts, 1990.

[14] L. Liu, M. Halper, J. Geller, and Y. Perl. Controlled vocabularies in OODBs: Modeling issues and implementation. In *istributed and Parallel Databases*, pages 37–65, 1999.

[15] F. Lopez. Overview of methodologies for building ontologies. In *the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends.Intelligent Systems*, pages 26–34, 2001.

[16] E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Conference on Cooperative Information Systems*, 41:14–25, 1996.

[17] N.F. Noy and C. D. Hafner. The state of the art in ontology design. *AI Magazine*, 3(18):53–74, 1997.

[18] C.W. Olofson. Addressing the semantic gap in databases: Lazy software and the associative model of data. Bulletin), 2002.

[19] B. Omelayenko. Integrating vocabularies: Discovering and representing vocabulary maps. *The Semantic Web-ISWC 2002, First International Semantic Web Conference, Sardinia, Italy*, pages 206–220, 2002.

[20] N.W. Paton, R. Stevens, P. Baker, C.A. Goble, S. Bechhofer, and A. Brass. Query processing in the TAMBIS bioinformatics source integration system. *Statistical and Scientific Database Management*, pages 138–147, 1999.

[21] H.S. Pinto and J. P. Martins. A methodology for ontology integration. In *the First International Conference on Knowledge Capture (K-CAP)*, pages 368–375, 2001.

[22] A. Sheth. Data semantics: what, where and how. Technical Report Preprint CS-01-99, TR-CS-95-003, LSDIS Lab, Dept. of CS, Univ. of GA, 1995.

[23] J.D. Ullman. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, 1988.

[24] M: Uschold and M: Grüninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.