Towards a Knowledge Base Management System (KBMS): An Ontology-Aware Database Management System (DBMS)

Henrique C. M. Andrade*
Department of Computer Science
University of Maryland, College Park
hcma@cs.umd.edu

Joel Saltz
Department of Computer Science
University of Maryland, College Park
and
Johns Hopkins Hospital
Department of Pathology
saltz@cs.umd.edu

Abstract

This paper aims to provide limited knowledge awareness to a conventional DBMS (Database Management Systems). This goal is achieved by extending an off-the-shelf DBMS (Postgresql in our case) in such way that it becomes ontology aware. The concept of ontology is used in our approach as a way of formalizing knowledge and relationships among objects in a domain of interest. Our solution is compounded by two main pieces: an external knowledge server and a set of functions to extend the DBMS. We argue that our solution is both powerful in the sense of supporting knowledge retrieval in the queries, and generic, in the sense that it can be deployed in any DBMS with the support for user-defined functions. Two application domains that can benefit from our approach are data mining and ad hoc query processing in hypothesis exploration environments (e.g. medical research). We also argue that our approach is original in how it pushes a conventional DBMS towards having features like the ones expected from Knowledge Base Management Systems (KBMS).

1 Introduction

Commercial relational DBMSs are tailored to efficiently support fixed format data models in what is known as **data management**. Nevertheless the upcoming demands in data analysis are pushing the technological frontiers to allow that two new other dimensions be supported by such systems: **the object management** and **the knowledge management**. We are specially interested in the second issue.

As defined by Stonebraker and Kemnitz [11], knowledge management entails the ability to store "rules" (as defined in First Order Logic) that are part of the semantic of an application. These rules allow the derivation of data that is not directly stored in the database. Later, we will see that our

*Work partially supported by CNPq (grant 200.167/97-9).

www.manaraa.com

approach accomplishes the same thing by using another formalism, instead of storing rules, and we will argue that this solution is actually much more flexible.

A number of application domains would benefit of the knowledge management capabilities, and, therefore, a simple, powerful, and efficient mechanism to add the knowledge dimension to an off-the-shelf DBMS can be rather useful.

Before we draw an analysis of how this was accomplished, we introduce the concept of ontology.

O'Leary [8] defines an **ontology** as "an explicit specification of a conceptualization". This knowledge-based specification typically describes a taxonomy of the relationships that defines the knowledge. Within the context of knowledge-management systems, ontologies are "specifications of discourse in the form of a shared vocabulary". Informally, [3] remarks that an ontology usually provides some help into describing facts, beliefs, hypotheses, and predictions about the world in general, or in a limited domain, if that is what is needed.

According to [8], ontology or taxonomy issues are emerging as one of the most important problems in knowledge management, mainly as a medium to formalize knowledge and to allow information sharing based on a common vocabulary [5].

Therefore what used to be a problem of AI, is now becoming a much broader issue because several application domains are making use of ontologies to add the knowledge dimension to their tools.

Bench-Capon [2] citing an earlier research points out a couple of motivations for using ontologies as the way of organizing information. Among them, we deem the following as the most important ones:

1. Knowledge sharing: ultimately it would allow that a federation of knowledge bases be able to solve problems by exchanging messages accordingly to the query.

2. Verification of a knowledge base: which is the validation of data as provided by traditional systems.

The database community is also finding applications for the concept of ontologies, but with a different target. One of the interesting applications is the *ad hoc* query generation problem in complex domains.

Weinstein correctly points out in [15] that relational technology is suited to applications with highly standardized data, because these applications can fit well into a design that breaks the data into tables by normalization. On the other hand, in complex domains, normalization can produce a plethora of tables, destroying the efficiency and maintainability.

This problem clearly comes up during the query generation. Conventionally, query generation is done using GUI interfaces that build the query using knowledge about the underneath data description (e.g. relations and relationships) in such a way that a SQL command is assembled and sent to the database engine.

In most information systems, this approach is satisfactory because the queries and the query domain of interest being described are well defined and self-contained. Nonetheless, some applications may need more complex ways of generating queries, specially for what we call *data exploratory tasks*.

An example of an application scenario will make such issues blatant. Consider the medical research domain. According to [9], typical medical databases have attributes with enormous name

www.manaraa.com

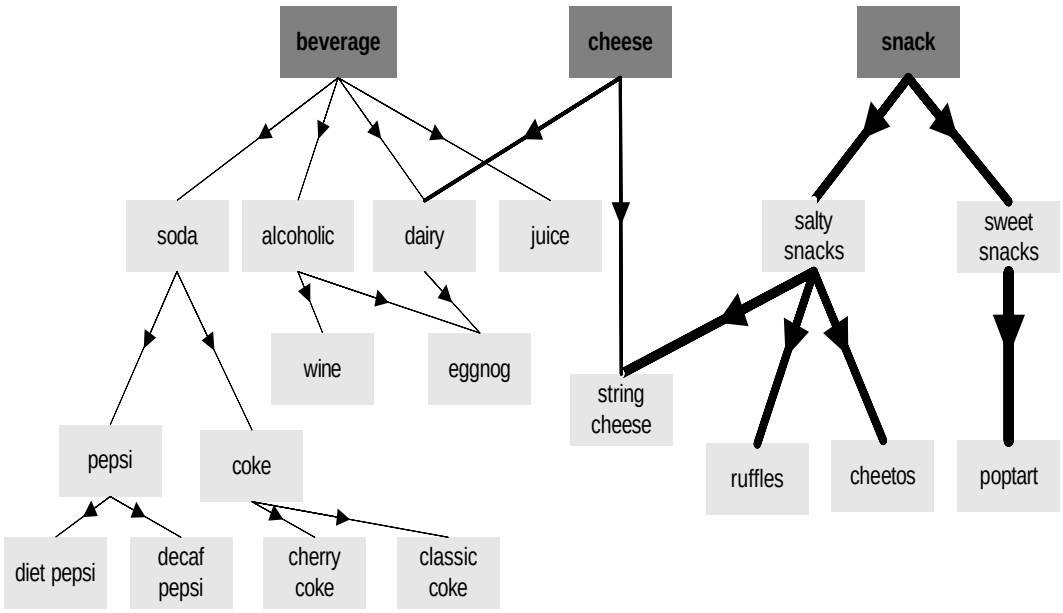

Figure 1: Excerpt of a $product$ ontology. The weight of the edges represents objects originated from a root category (in this example, $beverage, cheese$, and $snack$).

spaces. In addition to that, some attributes have semantic dependencies (ontological dependencies) with each other and the interpretation of one attribute instance may depend on another one.

Stoffel et alli [9] describe a mechanism to overcome some of the hurdles of using conventional DBMS just presented using two main ideas. Firstly, it stores some meta-data describing an attribute hierarchy (represented as a DAG) that even support the creation of new derived attributes dynamically. Secondly, it uses a graphical query tool which is able to explore the hierarchy and generate complex queries like *find all patients with cultures growing gram negative rods*. In this query example, the powerfulness of their approach consists in the fact that the query retrieves patients for whom the organism is a either a "gram-negative-rod" or any of its sub-categories (according to the hierarchy of *gram-negative rods*).

The main advantage of this approach is that the users (in this case the medical specialists) are able to express more complex queries without the burden of becoming experts on the underlying data model [10].

Stoffel et alli [10] also present the idea of **semantic indexing**. It consists in building ontology-aware indices that would allow a system to retrieve data grouped by ontological concepts, essentially, allowing efficient retrieval of tuples semantically associated.

These solutions are not generic enough, but they are a step in the right direction.

Another data exploratory task is data mining. Many of the data mining approaches generate rules based solely on the contents of the database. Nevertheless, the utilization of some *background knowledge* can supplement the discovery process and generate rules with semantical meanings, based, for example, on aggregations over an ontology.

As an example from the classical *supermarket basket scenario* [1], instead of generating rules like $Coke \rightarrow Ruffles$ and $Pepsi \rightarrow Doritos$ (meaning, respectively, that a customer buying Coke would buy also Ruffles, and that a customer buying Pepsi would also buy Doritos), we would

www.manaraa.com

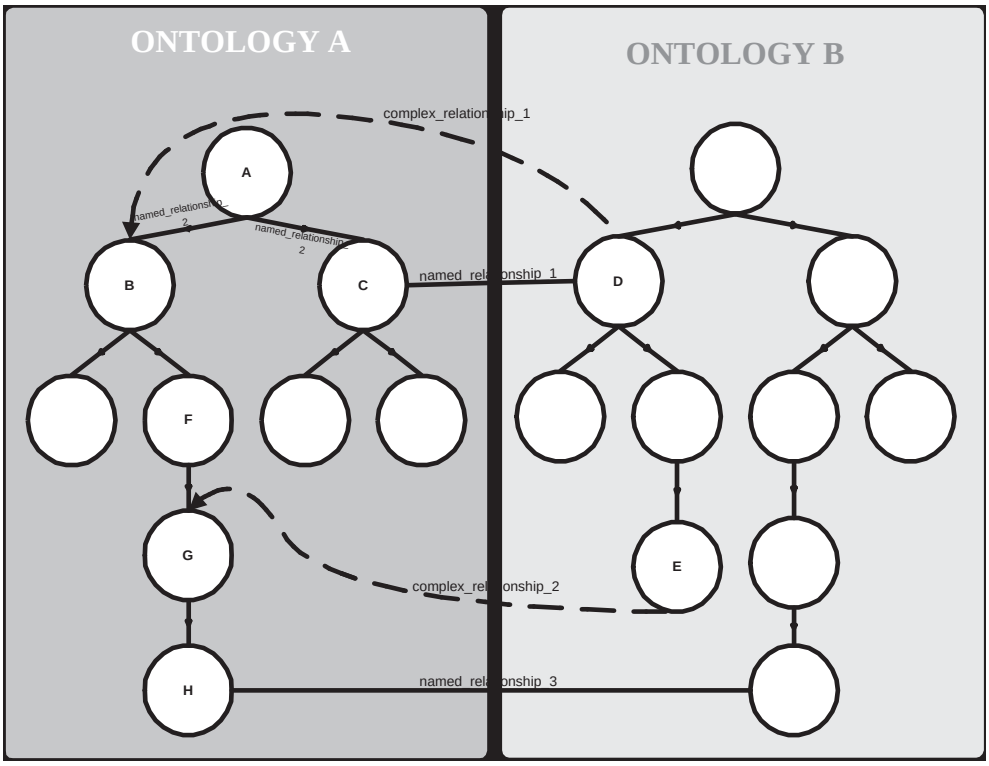

Figure 2: Two ontologies: A and B. A, B, C, F, G, and H are objects from ontology A, and D and E are objects from ontology B. The arcs show intra and inter-ontology relationships among objects.

generate something semantically more complex like $Soda \rightarrow SaltySnacks$.

Taylor [12] describes the implementation of such ideas in the context of ParkaDB which is a knowledge representation system developed by the PLUS group at the University of Maryland. Their authors claim that this approach led to the generation of rules that provide a "clearer" synopsis of the database. This is certainly achieved because instead of generating several potentially uninteresting rules, the system generates rules based on concepts of higher abstractions, possibly uncovering interesting relationships.

In the following section, we describe our approach which provides the infrastructure to support the two application scenarios we just discussed in a generic way. This goal is achieved by extending a conventional, off-the-shelf DBMS. We also draw some arguments pointing out that our approach drives a DBMS towards a KBMS in terms of features. Basically, it presents to the user stronger semantic capabilities in the query language and the possibility of storing knowledge in a very well structured way.

2 Extending a DBMS to support ontology-aware queries

Our approach pushes a conventional DBMS towards a KBMS in terms of capabilities and hence it is important to summarize some concepts. Formally, a knowledge-base management system (KBMS) is a system that [13]:

www.manaraa.com

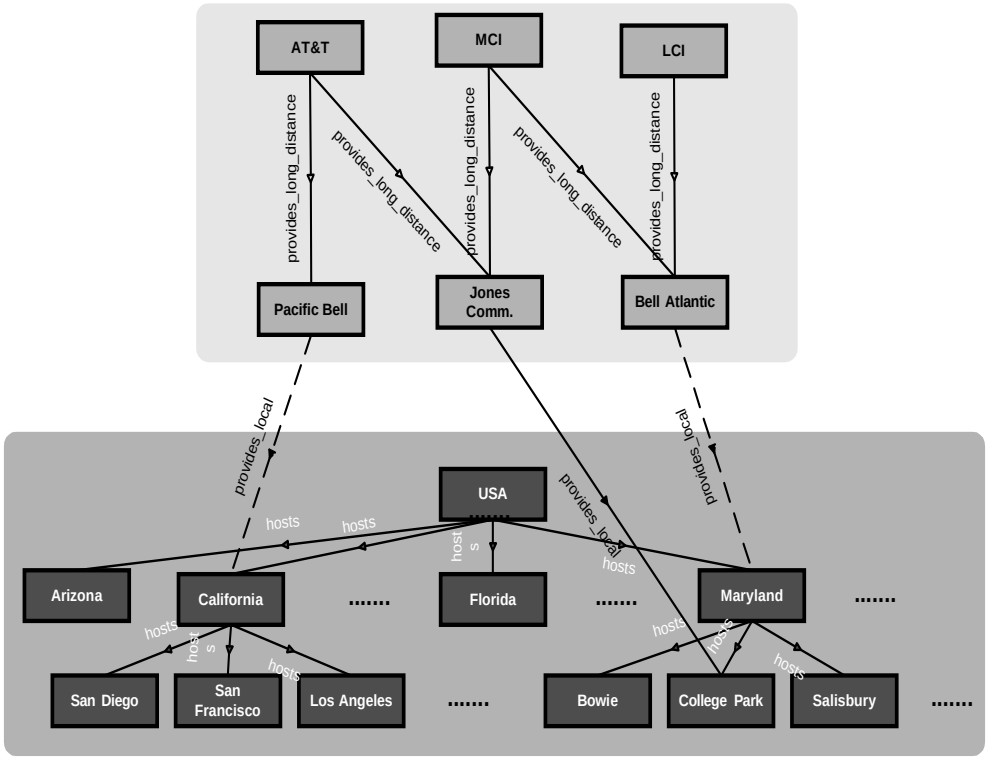

Figure 3: A sample ontology formed by two classes of objects: $location$ (lower box) and $telecom\_provider$ (upper box).

1. Provides support for efficient access, transaction management, and all other functionalities associated to DBMSs.

2. Provides a single, declarative language to serve the roles played by both the data manipulation language and the host language in a DBMS.

One of the formalisms used to describe KBMS capabilities is presented in [14]. The author tackles the general problem of representing knowledge in a database by using a data model[1] named *Datalog*. Key to this formalism is the concept of the extensional database (EDB) which is formed by tuples actually stored into the database, and the concept of the intensional database (IDB) which are basically predicates that embed some kind of knowledge about the world.

For example, considering an ontology describing beverages (like the example in figure 1) and possible relationships among instances of it, one possible powerful query to be issued in an ontology-aware application would be: `SELECT * FROM beverages_instock WHERE is_a(bev_name, 'Alcoholic')`. That is, we would like to retrieve all `beverages_instock` tuples which satisfy the *predicate* `is_a(bev_name,'Alcoholic')`, i.e., all tuples that are alcoholic products. It means that it doesn't matter whether the product is from the category of "wine" or "eggnog", because both are "alcoholic" (see figure 1).

[1] A data model is a mathematical formalism with two parts: a notation for describing data and a set of operators to manipulate that data [13].

www.manaraa.com

As with Datalog, our approach first defines a formalism to hold the domain knowledge, i.e., factual knowledge describing objects, properties, relations, classes, and subclasses, states, process, parts, etc. And, later, we define the operators over the domain knowledge that ultimately will give the DBMS the power of "reasoning" upon the raw data stored in its tables. In some sense, it can be regarded as the IDB in Datalog, whereas the EDB is the database provided by Postgresql in the form of the relational tables.

This approach is both simple and powerful. It is simple because the integration with an off-the-shelf DBMS could be accomplished without any modification of its engine (though we would like to, due to efficiency reasons), and powerful because it aggregates "reasoning" power to the query answering process.

We shall see in the next subsection the way our own formalism allows Postgresql to support ontology-aware queries turning it into something resembling a KBMS.

2.1 Formal Specification

Ontologies are defined as an abstract data type (ADT) with three data structures:

1. a set of *objects* O. Figure 2 depicts two ontologies A and B and, A, B, C, F, G, and H are examples of objects that pertain to ontology A.

2. a set of *named relationships* (functions) N that labels mappings from one object to another object. In its basic form it is a function defined as follows $f : O \rightarrow_{relationship\_name} O$, therefore named relationships are always functions of arity 2.

   In figure 2, we can see an example of a named relationship connecting objects A and C. In our graph, we call this relationship of $named\_relationship\_2$. Also, we can see a named relationship connecting objects of different ontologies. In this case, we have $named\_relationship\_1$ connecting objects C (from ontology A) and D (from ontology B).

   Also, a more sophisticated kind of named relationship can also be defined. It allows an object o_1 to be connected to another object o_2 and its descendents according to a named relationship that specifies which relation holds this parenthood relationship. The name of the relationship that represents the parenthood is necessary because it can happen that two objects are connected more than one time by different relationships. Formally, it would be defined as $f : O \Rightarrow_{(relationship\_name, parenthood\_relationship\_name)} O$.

   An example of this kind of relationship can be seen in figure 2. Object D is connected to object B, and all descendants of the latter. Semantically, it means that $complex\_relationship\_1$ holds for any pair from the set $\{(D, B), (D, F), (D, G), (D, H)\}$.

3. an *ontology graph* $G(V, E)$, where the vertices V are *objects* ($v_i \in O$, where v_i is one of the vertices in V), and the edges E are relationships (i.e., $e_i \in N$, such that e_i is connecting two vertices v_j and v_k that have a named relationship or $is\_a(v_j)$ and $is\_a(v_k)$ lead to objects o_m and o_n that have a named relationship [2]. In figure 2, we show two subgraphs (one for ontology A and the other for ontology B).

[2] $is\_a(X)$ returns the hierarchy of types for a given object.

www.manaraa.com

The ontology ADT is also formed by two category of methods:

1. the *named relationship* which allows one to evaluate whether $relationship\_name(o_i, o_j)$ or $relationship\_name(is\_a(o_i), is\_a(o_j))$ is valid, basically by looking it up in G, i.e., find an edge that is an instance of the named relationship you are looking for. We can think of named relationships as predicates as defined in the First Order Logic, in the sense that they return either *true* or *false*.

   Figure 2 shows generic examples of four named relationships: $named\_relationship\_1$, $named\_relationship\_2$, $complex\_relationship\_1$, and $complex\_relationship\_2$. Figure 3 depicts a possible real-world example, with named relationships like $hosts$ that interconnects two objects of the category $location$, $provides\_long\_distance$ which connects two objects of the category $telecom\_provider$, and $provides\_local$ which connects pairs of the category $(location, telecom\_provider)$.

2. the *named inference path*, which allows one to evaluate a complex (multiple-edge) relationships, by describing a path (a concatenation of edges) of named relationships to be found in the graph G. In summary, it defines a path description to be found by a graph traversal algorithm. Named inference paths are also regarded as predicates, and therefore their evaluation can be either *true* or *false*.

   Figure 4 shows a possible *inference path* for the knowledge base depicted in figure 3. $provides\_long\_distance$ defines a path of length 2 that connects an object from $telecom\_provider$ to another one from $location$. In order to satisfy this inference path, the following pattern has to be found: $telecom\_provider$ connecting to another $telecom\_provider$ by a relationship $provides\_long\_distance$ which is also connected to an object $location$ by a named relationship $provides\_local$ (if the named relationship is a complex one, the parenthood relationship is $hosts$). Notice that here we are allowing the overloading of the name $provides\_long\_distance$. The name clash is resolved by the type of the parameters (the named relationship $provides\_long\_distance$ connects two objects of the category $telecom\_provider$ and the inference path connects a pair of $(location, telecom\_provider)$.

The approach just described shares some commonalities with the concept of **semantic networks**[3] developed in the context of Artificial Intelligence. Nonetheless, our formalism constraints the way the graph structure can be explored to perform reasoning efficiently, because the DB administrator is supposed to define himself/herself the named inference paths that make sense in his/her application, restricting the way "reasoning" can be done.

3 Implementation

Traditional relational DBMSs support a data model consisting of a collection of named relations (formed by typed attributes). Postgres is a DBMS that extends this paradigm by allowing the

[3] In AI, a semantic network is a graph structure that encode taxonomic knowledge of objects and properties. In this domain, nodes can be either taxonomic categories, properties, or object constants. The arcs can be either *subset arcs* (denoting *isa* links), or set membership arcs (*instance* links).

www.manaraa.com

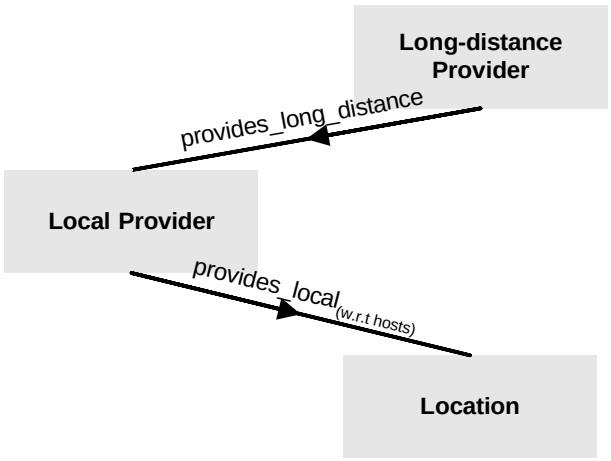

Figure 4: $provides\_long\_distance(location, telecom\_provider)$ is an example of an *named inference path*. An it is evaluated by searching a path that connects an instance of an object $location$ to an instance of an object $telecom\_provider$, by finding the two intermediate named relationships $provides\_long\_distance$ and $provides\_local$. Since $provides\_local$ can also be a complex relationship, we are specifying that the parenthood relationship is $hosts$.

definition of new classes, types, and functions [11]. Its implementation started as a research program in 1986. Currently, Postgres has become an open-source piece of software renamed to Postgresql and supported by a number of developers over the Internet.

Our implementation basically consisted of extensions to Postgresql to provide new functions, and, through these, inference mechanisms. Both are implemented by interactions between the back-end engine with an **ontology server**, also developed in the context of this present work.

In this first approach, we are providing a basic interface of new function calls that can be used in conventional SQL commands. These functions are exactly the access methods of interaction with the ontologies stored in the ontology server: the named relationships and the named inference paths.

These functions (access methods) interact with the ontology server by a mechanism of dynamic linking. The *ontology server* is the autonomous module in charge of storing the data structures mentioned in the previous section and also of implementing the access methods.

Currently the functions are built tailored to the ontologies we have currently stored in the ontology server. But we intend to provided a generic interface so the user can define ontologies using the conventional relational data model and external information. And then, from these two pieces of data, the methods to handle the ontology will be automatically constructed. Therefore, it will be possible to avoid the burden of using a programming language to describe the ontology, and what is even worse, integrated a newly built ontology to a universe of the already existing ontologies.

The access methods are implemented as graph traversal routines. And they can be used either as a the name of the column to be projected or as part of a condition in an arbitrary query. For example:

```
SELECT beverage_name, is_a(beverage_name,'Alcoholic')
FROM beverage_database;
```

or

www.manaraa.com

```
SELECT beverage_name
FROM beverage_databases
WHERE is_a(beverage_name,'Alcoholic');
```

In the above example, `is_a(beverage_name,'Alcoholic')` should be read as $is\_a(beverage\_name) ==' Alcoholic'$. Therefore, in order to evaluate the predicate is_a, the graph depicted in 1 must be traversed until it can be verified whether a path connecting $beverage\_name$ and $Alcoholic$ exists or not. This task is accomplished by the ontology server and a boolean result is returned. It should be noticed that the path can be potentially very long, depending on how deep and complicated the ontology is.

Another possibility is the utilization of the functions to aggregate or order the tuples, like in:

```
SELECT name,is_a(name,'Soda')
FROM instock
ORDER BY is_a(name,'Soda');
```

4 Using the improved Postgresql

In the last section, we have described some utilization of ontologies, although they are rather trivial, they already start to show a whole new set of capabilities in query answering that the users can make use of.

The utilization of the inference path function calls gives the power of limited "reasoning" over the database. An example of a query using the named inference path in the context of our telecom provider (figure 3) would be:

```
SELECT customer_name, customer_address, telecom_provider_name,
FROM customers, telecom_providers
WHERE customer_name='John Doe' AND
      provides_long_distance(customer_city,telecom_provider_name);
```

In this case something much more complex than the information already in the database is being deduced using a function call that invokes the named inference method function ($provides\_long\_distance$), and tries to find a path in the ontology graph.

A *Cartesian product* is executed using the tables `customers` and `telecom_providers` and then the expression is evaluated to check if the tuples satisfy it. The evaluation of `provides_long_distance(customer_city,telecom_provider_name)` requires that the ontology server find a path in the graph of figure 3 that connects the customer to a long distance carrier provider. For example, let's suppose that John Doe lives in College Park. College Park is connected to Bell Atlantic by a $provides\_local$ named relationship, and Bell Atlantic is connected to MCI by a $provides\_long\_distance$ named relationship. Therefore, MCI is returned as an answer to the query. By the same reasoning, so are LCI and AT&T. This latter one because of the local service to College Park offered by Jones Communications. Thus, the output obtained from Postgresql is:

www.manaraa.com

```
customer_name       telecom_name
--------------------------------
John Doe            MCI
John Doe            LCI
John Doe            AT&T
```

Nevertheless, even more interesting utilization can be implemented using the *libpq* interface[4]. The implementation of complex data mining or hypothesis testing queries can be easily achieved and so we can, for example, implement the algorithms like the ones described in [12] or [9].

Another intriguing possibility is the extension of the work of Meo et alli [7] and implementing the *MINE RULE* operator in an ontology-aware fashion such as the example bellow suggests:

```
MINE RULE SimpleAssociations AS
SELECT DISTINCT 1..n item AS BODY, 1..1 item AS HEAD, SUPPORT, CONFIDENCE
FROM Purchase
WHERE price<=15 AND (is\_a(BODY,'Soda') AND is\_a(HEAD,'Chips')),
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT: 0.1, CONFIDENCE: 0.2
```

The original implementation mentioned in Meo's paper was concerned with retrieving purchases of items that cost more that $15 into a temporary table $SimpleAssociations$. Each tuple corresponds to a discovered rule. The `SELECT` clauses defines the structure of the rule: the body is a set of `items` whose cardinality is any positive integer, and the head is defined as set containing one single item. Our contribution here is to allow rules to be grouped into a higher ontological category, in this case $Soda$ and $Chips$, instead of particular instantiations of these categories.

As just showed here, the same approach could be used for the other categories of data mining summarized in Meo's paper.

4.1 Limitations and Future Work

Our implementation is fully functional and the ontology server was deployed as an external plug-in to Postgresql. Nonetheless, we foresee a design where the ontology itself can be stored into carefully crafted relations, and the methods can be implemented using the *libpq* interface to perform the graph traversal operations over the relations. This implementation would make the whole system integrated. Postgresql currently does not allow us to use this alternative though, because its parser is not reentrant. We are working to overcome this limitation by a couple of modification in its source code.

As already mentioned, one scenario that would benefit from our infrastructure is the hypothesis testing environment. Thus, a very useful improvement would be the deployment of a graphical interface to generate ontology-aware queries. This would make possible to generate high level queries like the ones presented in [12].

One limitation of our approach is due to the fact that currently there is no interaction between the ontology server and Postgresql's optimizer. Therefore, the optimizer is blind when computing the

[4]The libpq [4] interface allows the development of applications that interacts with the DBMS by calling functions that send SQL commands to the back-end, and get the results back.

www.manaraa.com

cost of either named relationship or inference path is concerned. Moreover, because the ontology server assumes no locality in running its methods, a lot of computation can be wasted. To make this example clear, let's analyze the following query:

```
SELECT customer_name, customer_address, telecom_provider_name,
FROM customers, telecom_providers
WHERE provides_long_distance(customer_city,telecom_provider_name);
```

This query executes a *Cartesian product* and, depending on the database contents, the inference path $provides\_long\_distance('CollegePark','BellAtlantic')$ will be computed multiple times. As pointed out, the computation of such a method is actually implemented as a graph traversal and can be pretty expensive. Nevertheless, because this problem can show up frequently, the ontology server could implement a caching mechanism, and hence avoid the re-computation of things already known. In fact, caching can be pretty beneficial due to the fact that the knowledge stored in the ontology server tends to be fairly constant over time.

Another improvement to be implemented is the development of indexing methods to improve the performance of the graph traversal routines, by borrowing the paradigm followed by the Generalized Search Tree [6].

Finally, we can state that our ultimate goal is the deployment of the ontological knowledge as the data model for the data management system. In some sense, a given data model in the relational model is already a subset of an ontology for some domain. In effect, Chandrasekaran points out in [3] that databases built using the simple ontology cannot make simple inferences that one would expect to be able to make given a knowledge base and our proposal will overcome this barrier in the same way our current work does, i. e., making the ontology the formalism for describing the data model would give all the power that our solution provides, plus the benefits of being completely integrated into the database engine, and therefore allowing a perfect interaction with the indexing and optimizing subsystems.

5 Conclusion

The greatest advantage of our approach is the utilization of conventional off-the-shelf DBMS. And hence the possibility that it can be used with legacy systems without introducing the burden of the implementation of new architecture with a whole new design-implement-test cycle.

In effect, the knowledge dimension can be aggregated only to portions of the database that will benefit from it. For example, in a supermarket management database infrastructure, an ontology tailored to the sale transactions could help identify high level rules like the ones described in section 1 without disrupting the database that deals with stock management. Likewise in a medical center, only the databases of the information systems related to managing clinical data would be modified to help a pathologist to find a set of rules that defines the pattern of efficiency of a given antibiotic against a family of bacteria.

Despite the gains of our approach, we still can have higher improvements if we extend the DBMS to have the ontology as its language to describe the data model in the way described in the last section. And then obtaining a true, full-fledged KBMS with the needed efficiency and power of expression.

www.manaraa.com

References

[1] R. Agrawal, T. Imielinsk, and A. Swami. Mining association rules between set of items in large databases. In *Proceeding of the ACM SIGMOD Conference on Management of Data*, pages 207–216. ACM, 1993.

[2] T. J. M. Bench-Capon and P. R. S. Visser. Ontologies in legal information systems: The need for explicit specifications of domain conceptualisations. In *Proceedings of the 6th International Conference on Artificial Intelligence and Law*, pages 132–141. ACM, 1997.

[3] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins. The ontology of tasks and methods. In *11th Workshop on Knowledge Acquisition, Modeling and Management*, 1998.

[4] The PostgreSQL Development Team edited by T. Lockhart. *PostgreSQL Programmer's Guide*. www.postgresql.org, 1998.

[5] J. H. Gennari, D. E. Oliver, W. Pratt, J. Rice, and M. A. Musen. A web-based architecture for a medical vocabulary server. Technical Report Knowledge Systems Laboratory, Medical Computer Science, KSL-95-41, Knowledge Systems Laboratory, 1995.

[6] gist.cs.berkeley.edu. Gist: A generalized search tree for database systems. gist.cs.berkeley.edu.

[7] R. Meo, G. Psaila, and S. Ceri. A new sql-like operator for mining association rules. In *Proceedings of the 22nd VLDB Conference*, 1996.

[8] D. E. O'Leary. Using ai in knowledge management: Knowledge bases and ontologies. *IEEE Intelligent Systems*, pages 34–39, 1998.

[9] K. Stoffel, J. D. Davis, G. Rottman, J. Saltz, J. Dick, W. Merz, and R. Miller. A graphical tool for ad hoc query generation. In *AMIA'98*. American Medical Informatics Association, 1998.

[10] K. Stoffel, J. Saltz, J. Hendler, J. Dick, W. Merz, and R.Miller. Semantic indexing for complex patient grouping. In *AMIA'97*. American Medical Informatics Association, 1997.

[11] M. Stonebraker and F. Kemnitz. The postgres next generation database management system. *CACM*, 34(10):78–92, 1991.

[12] M. G. Taylor, K. Stoffel, and J. A. Hendler. Ontology-based induction of high level classification rules. *SIGMOD Data Mining and Knowledge Discovery workshop Proceedings*, 1997.

[13] J. D. Ullman. *Database and Knowledge-Base Systems*. Computer Science Press, Rockville, MD, 1988.

[14] J. D. Ullman. The database approach to knowledge representation. In *Proceeding of the 13th National Conference of the AAAI*. American Association for Artificial Intelligence, 1996.

[15] P. C. Weinstein. Ontology-based metadata: Transforming the marc legacy. In *Proceedings of the 1998 ACM 7th International Conference on Information and K nowledge Management*, pages 52–59, 1998.

المنارة للاستشارات

www.manaraa.com