Putting the Semantic Web to Work with DB Technology*

N.Moreno, I. Navas, J.F. Aldana
Higher Technical School of Computer Science Engineering
Computer Languages and Computing Science
Bulevard Louis Pasteur n^o 35. 29071. Málaga, Spain
{vergara, ismael, jfam}@lcc.uma.es

Abstract

We present an architectural design that provides several services in an integrated environment, such as query/inference capabilities over both data and knowledge, dynamic annotation of web pages, maintaining consistency for repositories and annotated pages, crawling services and automatic matching between schemata. This architecture offers a practical and feasible step towards the design, maintenance, integration and development of semantic web applications within concrete domains, making use of and extending standard database technology. For example, in the field of bioinfomatics, non-expert users need tools to assist them to store, query and place the relevant results of their research efforts. Our methodology provides such tools to leverage information gathered from both the "Deep" and "Surface" Web.

1 Introduction

Semantic Web technologies provide a natural and flexible solution for integrating and combining two levels of abstraction, the data level and the knowledge level, which are related by means of metadata. Information is annotated with semantic content/metadata that conform to a domain ontology. For this purpose, mark-up frameworks [1] have been developed to focus on the "*Surface Web*," those static web pages that can be easily found and indexed by traditional web crawlers. However, a large part of the data is stored in on-line databases that comprise the *"Deep Web"* (e.g., dynamic pages built up from web forms). In consequence, current web search engine technology can only be used to query the small fraction of all the data available on the Internet, that is in the "*Surface Web*." The rest can be queried using form-based interfaces that automatically convert user queries into queries over the databases. Although easy to use, these form-based interfaces reduce expressibility of the queries, only allowing conjunctive queries with selection predicates. The user queries are specified by filling in the form fields, imposing strict constraints (mostly equality) on the attribute values stored in the database.

Traditional database techniques do not support effective and efficient query processing in the Semantic Web context because they lack inference capabilities. Methodologies from the field of artificial intelligence, on the other hand, support inference, but can only handle relatively simple models when persisting and manipulating

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work has been supported by the Spanish MCyT Grant (TIC2002-04186-C04-04)

large quantities of data. The latter find answers about the model which had not been explicitly predefined and that involve fewer, but much more complex, data. In this sense, ontologies contribute to the Semantic Web's goal, making explicit a higher level of abstraction and knowledge than semantic data models, and providing new ways for applying efficient reasoning techniques that are not supported by database management systems.

This paper focuses on the use of ontologies to assist database modeling and query processing. For the latter, we explicitly store the relationships that exist between a semantic data model and a related domain ontology. These mappings enhance the semantic knowledge about data and their structure. The preservation of the relationships allows us to use them during query processing or semantic optimization. This paper is organized as follows: section 2 gives an overview of the system in which our proposal is framed; section 3 details those aspects from the Storage Service relevant in making it easy to understand how the query service works; section 4 describes the query service; and finally we conclude with some remarks and a description of future work.

2 System Overview

The emergence of the Semantic Web has lead to the development of distributed systems which share knowledge by making use of conceptualizations of specific domains, generally defined by means of ontologies. Multiple conceptualizations may describe similar domains using different terminologies, or they may overlap. These issues should be, but are not usually, taken into account when a new system is designed, which leads to the increase of development time and effort. In order to tackle these problems, we propose that the design of the application schema should start with the domain ontology (O_D). This allows us to formally define and store the mappings between the O_D and the application schema. The relationships between the data model of a concrete application and the corresponding domain ontology semantically enrich the information in the application domain, capturing it in a more expressive data model and improving several other processes. For instance, this enables enhanced query processing capabilities and semantic query optimisation.

Furthermore, we think that Web applications should be designed taking into account interoperability and integration issues. In this sense, the proposed design methodology based on the existence of a O_D produces the definition and storage of solid mappings, which could be used both to facilitate interoperability between applications and to build an integration scheme. We have therefore developed a prototype that offers functionalities implemented through integrated tools, such as Protégé, ERWin and RACER. This approach is an integrated solution for the realization of the Semantic Web, which consists of four services: a storage service, a crawling service, a semantic page annotator, and a query service. Web developers can first use the storage service to design and implement the stores for their web site, then (by making use of the annotation service) semantically annotate their web contents. They can then take advantage of the crawling service to extract semantics from other web sites, whether or not those other sites were developed with our prototype system. Finally, non-human users (e.g., intelligent agents) can exploit annotations as well as the query service.

Storage Service: The main functionality of this service is to support the computer aided design of conceptual schemata and to translate them into physical implementations (for the most popular RDBMS such as Oracle, Sybase, SQL Server, etc.), leveraging the expressiveness and reasoning techniques derived from using the ontology as a conceptual model [2]. This is the core of the system, as the *Execution Plan Generator*, the *Crawler* or the *Annotation Generator*, among other components (see Figure 1), require information and knowledge derived from it (a set of mappings that relates data with stored knowledge).

Annotation Generator Service: We support the automatic annotation of both static and dynamic web documents. We assume the use of XML technology, and introduce an annotation function in the document's stylesheet to generate them automatically. For dynamic documents two steps are followed: the first one is to annotate the queries enveloped in the dynamic document, and the second to annotate the results of these queries. For the former we have described an ontology that defines the general structure of a dynamic document. Its annotation consists of automatically generating a public instance of that ontology. The second step, the

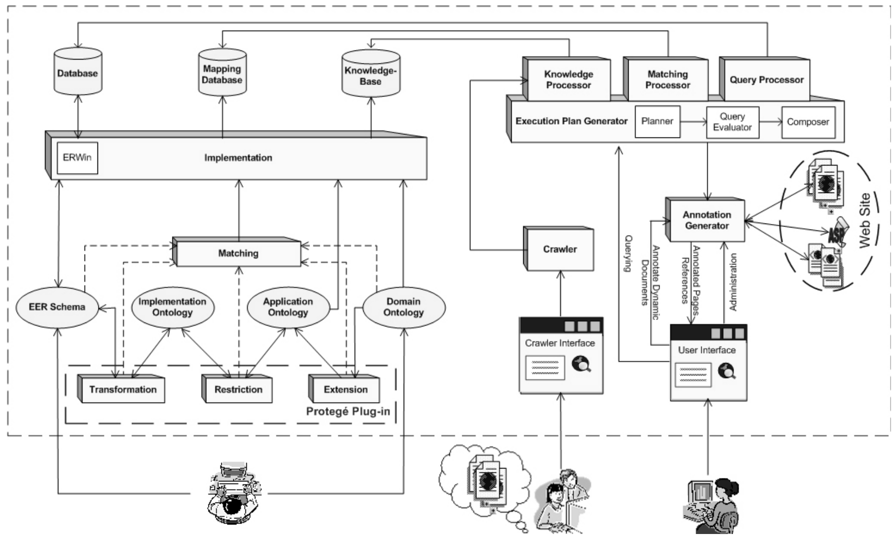

Figure 1: Internal System Architecture

annotation of results of these queries, is addressed applying the same methodology as for static documents.

Crawling Service: This service is an extended semantic crawler that combines [3] and [4] crawling techniques with the semantic annotation of the server-side programs described above to populate the knowledge-base (KB). Static contents are treated like traditional and semantic web crawlers. However, when the crawling process finds a dynamic document, it searches the document with the annotations for queries enveloped in it. The crawler uses this information to prepare an "attack plan" for the dynamic document. As the crawler discovers annotations, they are stored in the KB along with a reference to the web pages in which they were found.

Querying the System: Traditional database systems efficiently resolve a specific kind of query, namely associative queries. Their query capabilities are limited to physical database schemata. In contrast, search engines do not have this restriction, but they cannot derive a concrete answer returning links like query results. Our prototype simulates both behaviors, extending the query capabilities of traditional database systems. Furthermore, it also provides information related to the query results (e.g., links to other web pages). That is, it combines querying and reasoning into a single process.

3 Conceptual Database Design for Semantic Web Applications

The main functionality offered by the Storage Service is to support the computer aided design of conceptual schemata, and to translate them into physical implementations. This service (see Figure 1) requires that an expert user provide the system with a domain ontology following the OWL Lite nomenclature. This feature permits efficient reasoning with the RACER tool [5], which allow users to associate logical formulas to instances. Besides, by using its elementary predefined functions for navigating inside the schema, several questions can be solved and applied during the query processing. For example, we can learn which is the most specific concept one instance belongs to or what properties are related to a relationship (transitive, symmetric, inverse).

To obtain the database conceptual schema we can follow two paths: (1) designing an (E)ER schema from scratch, or (2) using a domain ontology as a foundation and extending it by means of a refinement process In the first case, with the ERWin graphical interface we can design the conceptual schema, which is exported to an intermediate representation in XML. The Matching Component matches the conceptual schema with the domain ontology [6, 7], generating a set of rewriting rules about the relationships between both models.

However, the design based on a domain ontology entails several advantages. To assist this process we

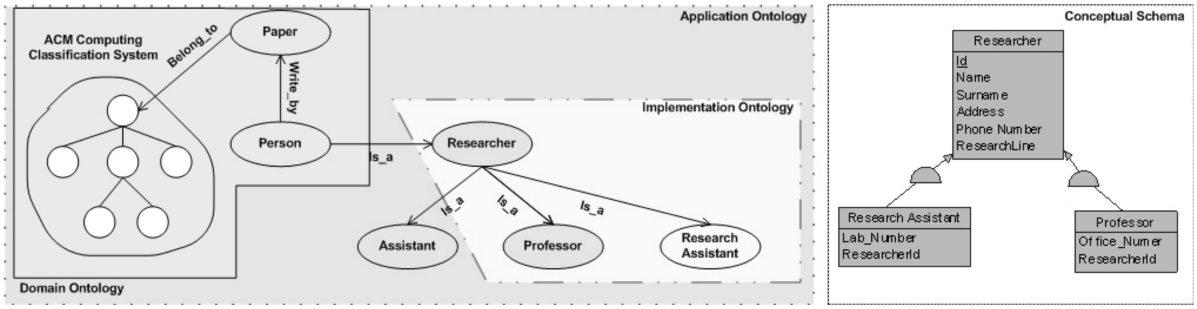

Figure 2: Ontology Creation Process. For simplicity and clarity, slots, rules, constraints, etc. do not appear

have developed a plug-in for the Protégé [8] tool that loads an OWL Lite domain ontology (O_D) and allows the designer to extend it with new relevant concepts, relationships and integrity constraints to the application domain. The result of this extension is an application ontology (O_A) focused on those elements needed for the specific web application under development. Nevertheless, not all the elements represented in O_A are required for a physical implementation (all could be relevant but perhaps persistent instances of all of these concepts are not necessary). Instead, only a selected subset of O_A will be queried by our database applications: O_I makes reference to the ontology obtained in this step, and it verifies $O_I \subseteq O_A$. The selection/renaming of the subset O_I from O_A produces a set of correspondences between these ontologies that is generated by the Restriction Component in order to be used by the Matching Component. Finally, we automatically obtain an enriched ER schema from the implementation ontology [2, 9]. In the process of deriving a final application schema from the domain ontology, we obtain a conceptual model enriched with the expressiveness and the inference capabilities of an ontology. These capabilities can be used and exploited during query processing.

Figure 2 shows an example illustrating the conceptual database schema design for a web application that stores information about authors and their publications. The example extends the *ACM Computing Classification System* domain ontology with classes that provide support for management instances of *Researcher*, *Assistant* and *Professor*, resulting in the O_A. Selecting only those parts that need physical storage from the O_A, we limit the scope of the O_I to the *Researcher* class and its sub-classes, renaming the *Assistant* class as *Research Assistant*. Finally, a translation from the O_I to an (E)ER schema is performed, generating a *Conceptual Schema*.

4 Querying the Surface/Deep Web

The Execution Plan Generator enables users to query annotated Web documents. This component (see Figure 1) analyzes conceptual queries, written in terms of the $O_D \cup O_A$, and can return document links, data and knowledge. From a conceptual point of view and following our design methodology, the data stored in the database can be seen as instances of the knowledge-base due to the fact that O_I (the ontology with which data committed) is contained in $(O_D \cup O_A)$. This particular feature allows us to evaluate queries like an inference process. Users formulate conjunctive queries in terms of $O_D \cup O_A$, which are more expressive than those offered by traditional web forms. Unlike traditional database systems, this extension adds support for reasoning to the system's initial query capabilities. Thus, when the Execute Plan Generator receives a query, the Planner analyzes dependencies between variables to generate the corresponding dependency graph, and locates where each sub-goal can be solved. This last task is carried out applying structural reasoning (e.g., class/sub-class subsumption) against the knowledge-base schema ($O_D \cup O_A$) and the database schema (S_I).

Making use of the information derived from the dependency graph and the location for evaluating each sub-objective, the Planner builds a query graph that is part of the evaluation plan. The latter includes a query graph with a list of sub-goals, which are the leaf nodes, and how they must be combined. Each leaf node includes

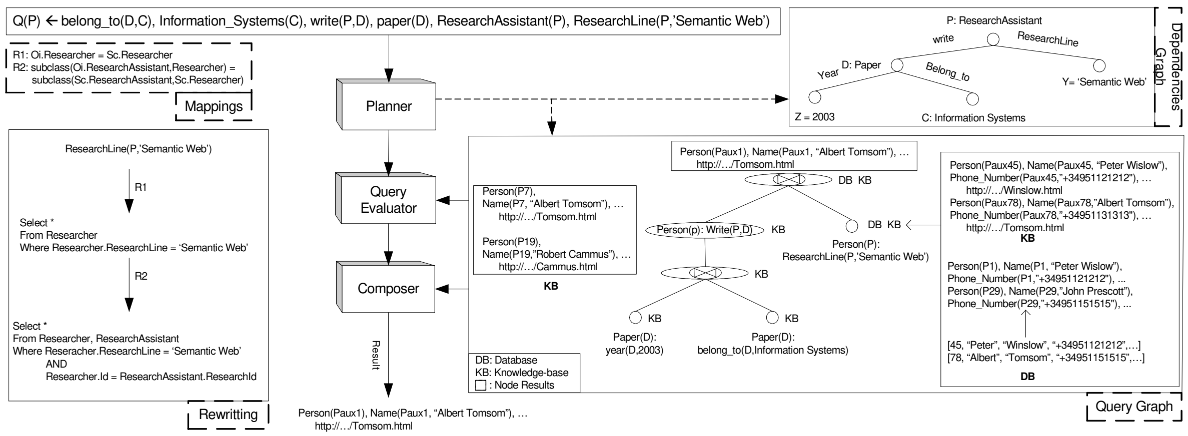

Figure 3: Example for querying the system

information about possible locations where it can be solved. This information is propagated to ancestor nodes of the graph in order to be used in generating the plan. For example, when the Planner needs to build a join node, its behavior depends on the provenance of the descendant nodes. If the nodes retrieve information from the same source, the intermediate node is generated based on a natural composition over unique key identifiers. Otherwise, the join is based on the similarity between data results. The graph structure defines for itself several alternatives to be evaluated, generating different execution plans. The Planner realizes a cost analysis of these plans and determines which has the lowest cost value based on heuristics, information about connection speeds, etc. The others are stored according to their cost value, in case execution errors occur.

Once the Planner ends, it sends the evaluation plans to the Query Evaluator component. Its main functionality is to rewrite and evaluate each sub-goal in terms of the corresponding sources in which it will be evaluated. For the rewriting phase this component takes advantage of the stored mappings looking for a sequence of correspondence rules that relates the initial sub-goal with an element of the data sources, in other words a class, a slot, a table, a relationship, etc. As users wait for responses in terms of $O_D \cup O_A$, the Query Evaluator must try to generate the inverse sequence too. If it is not possible to rewrite a sub-goal i.e. the direct or inverse mapping sequence is not found, then the execution is aborted and an empty result is returned. Using the inverse rules mentioned, partial results are translated to the initial nomenclature.

The Composer receives the execution plans and the results of evaluations from the Query Evaluator component. The reader should note that a leaf node can be resolved in several resources, and in this case it is necessary to combine all results providing a unique solution. Although this approach reduces efficiency as regards response time, we have opted for a more complete solution. Combining results entails having to face traditional issues of heterogeneous database integration, such as replication or inconsistency.

Figure 3 shows an example of query processing in which we solve the query: *Find Research Assistants whose research line is focused on Semantic Web and have a paper in the category Information Systems*. The query is shown in predicate logic terms, due to space limitations. Once the query reaches the Planner, it builds a dependency graph that represents the evaluation precedence of each variable in the query. Starting from this graph the planner generates the query graph, which in this context involves a parallel evaluation of leaf nodes and the composition of intermediate nodes when results are available. Note that each node is annotated with its source location. This annotation allows the Query Evaluator to decide how to rewrite each sub-goal in order to solve it. We have included in this illustration an example of a rewriting process based on stored mapping rules for the evaluation of the database. The annotations of the intermediate nodes provide the Composer with knowledge to decide which type of join must be performed. That is, if a join node only receives information from the knowledge-base, the join is done using the instance identifiers. However, when the information is provided from the database and the knowledge-base, the Composer combines the results making use of semantic distance among the instances. For example, it can use in the root join a similarity measurement based on the name of the people retrieved from the descendant nodes. In the illustration, the data provided from the knowledge-base and database will be combined deleting the redundancies. In this case, the instances about "Peter Wislow" and "Peter Winslow" are considered redundant and the data from the database are considered the valid ones, but the link to his personal web page is maintained on the combined instance. On the other hand, the root node applies a join based on the content of descendant node instances, due to data provided from the database and the knowledge-base. Note that the result contains information as well as links to web documents.

5 Conclusions and Future Work

In this paper we have presented a system and design methodology for Semantic Web applications, which allow us to translate query processing on databases to an instance retrieval process on inference engines. We enable such applications to combine traditional database systems' ability to manage large amounts of data with the expressiveness and the semantic enrichment of a conceptual model, bringing new query/inference capabilities, such as structural inference. Ontology users (or agents) will first query the ontology schema to identify what relevant information exists, and then proceed to query the data to extract the desired information from the underlying databases. Besides, we provide a way to automatically annotate both static and dynamic documents. Thus, we achieve the annotation of the Surface Web as well as the Deep Web, allowing to improve web crawling. In addition, the domain ontology used as the starting point in database schema design can act as the mediation schema for the integration of those databases that commit with it. This feature makes it possible to easily translate queries from one data source to another. Thus, we plan to apply this methodology to mediation systems.

References

[1] Semantic Web - Annotation and Authoring. http://annotation.semanticweb.org/tools

[2] A. Borgida, M. Lenzerini, R. Rosati. Description Logics for Data Bases. The Description Logic Handbook: Theory, Implementation and Applications. 2003.

[3] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In VLDB 2001: 27th International Conference on Very Large Data Bases, September 2001.

[4] D. Zeinalipour-Yazti, M. Dikaiakos. Design and implementation of a distributed crawler and filtering processor. In Proc. of NGITS 2002.

[5] V. Haarslev, R. Möller. RACER system description. In Proc. Of the Int. Joint Conf. On Automated Reasoning (IJCAR 2001), volume 2083 of LNAI, pages 701-705. Springer, 2001.

[6] G. Wang, J. Goguen, Y. Nam, K. Lin. Interactive Schema Matching With Semantic Functions. In Semantic Integration Workshop, Sanibel Island FL, October 2003

[7] E. Rahm, P. A. Bernstein. A survey of approaches to automatic schema matching. In The VLDB Journal: Volume 10 Issue , (2001), pages 334-350, 2001.

[8] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, M. A. Musen. Creating Semantic Web Contents with Protege-2000. IEEE Intelligent Systems 16(2):60-71, 2001.

[9] U. Sattler, D. Calvanese, R. Molitor. Relationship with other Formalisms. The Description Logic Handbook: Theory, Implementation and Applications. 2003.