RC22744 (W0303-077) March 17, 2003
Computer Science

IBM Research Report

Towards Enterprise-Scale Ontology Management

Juhnyoung Lee, Richard T. Goodwin, Rama K. Akkiraju, Yiming Ye
IBM Research Division
Thomas J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g. , payment of royalties). Copies may be requested from IBM T. J. Watson Research Center ,
P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home .

Towards Enterprise-Scale Ontology Management

Content Areas: Ontology, management system, application programming interface, query languages

Abstract

Increasingly, ontology languages are being used to externalize meta-data about data, software and services in a declarative form that can be used for purposes ranging from search and retrieval to composition and lifecyle management. The use of such externalized descriptions could significantly reduce the costs of deploying, integrating and maintaining enterprise scale systems. The barrier to more wide spread use of ontologies for storing meta-information is the lack of support in the currently available middle-ware stacks used in business applications. In this paper, we describe work on developing an enterprise-scale ontology management system that will provide the reliability, scalability and performance that enterprise uses demand. In addition, to be successful, such a system will need to fit well into the current software development environment and reduce rather than increase the burden on software architects, programmers and administrators. To accomplish this, we are synthesizing concepts familiar to software developers with ideas from the semantic web and ontology communities. In this paper, we describe the design and implementation of the system that programmatically supports the ontology needs of applications in a similar way a database management system supports the data needs of applications. For programmers, the system provides a Java API, Java Ontology Base Connector (JOBC), which is the ontological equivalent of the Java Data Base Connector (JDBC). JOBC provides a simple-to-use but powerful mechanism for application programmers to utilize ontologies without dealing with the details of ontological information. In addition, the ontology management system will support a number of query languages, and currently, we are exploring a variant of DAML Query Language (DQL) as our ontological equivalent of SQL (Structured Query Language). To satisfy the needs for application deployment in a variety of configurations, the ontology management system will support a range of configurations from lightweight thin-client deployments to an enterprise-scale federated server deployments.

1 Introduction

In recent years, there has been a surge of interest in using ontology information for building software applications, especially on the Web as a mechanism for communicating knowledge. An increasing range of applications require holistic control over activities in the spectrum of ontology management, including the creation, storage, retrieval, query, reuse, maintenance, and integration of ontological information. The idea of an ontology management system is that it provides a mechanism for dealing with ontological information at an appropriate level of abstraction. By using programming interfaces and query languages, application programs can manipulate and query ontological information without the need to deal with the details of files and database tables or to re-implement the semantics of standard ontology languages. This is analogous to the way that a database management system allows an application to deal with data as tables and not files and provides a query engine that can understand and optimize SQL queries.

The concept of ontology management system can be explained in the context of the application system structure evolution. The first-generation enterprise application software was large monolithic pieces of code with a complex internal structure, which combined the infrastructure code for manipulating data, and the pieces of code for implementing the actual logic of the application. The second-generation applications were noted by their use of database management systems (DBMS) which was devised to support the definition and concurrent manipulation of data, and consequently, separate data logic from applications. The use DBMS provides data independence to application programs. For example, changes to the data schema and access paths, for example, can be done without impacting the related applications.

In a similar way, a workflow management system (WFMS) separates business logic from applications by supporting the definition business processes. A WFMS externalizes the definition and execution of the control and data flow of processes, the assignment of people to activities, and the invocation of application logic modules. An application becomes a process-aware application consisting of a model of the underlying business process (referred to as a process model or a flow model) and a set of flow-independent application logic modules. In a similar way, ontology management systems will provide application programs with ontology independence.

In this paper, we describe the design and implementation of an ontology management system,

which is being developed at IBM T. J. Watson Research Center. The objective of the project is to develop an industry-strength ontology management system that will provide reliability, scalability, and performance for enterprise uses, and also provide functionality robust and sufficient for different levels of practitioners of ontological information. This system provides a Java API referred to as Java Ontology Base Connector (JOBC), which is the ontological equivalent of the Java Data Base Connector (JDBC). JOBC provides a simple-to-use but powerful mechanism for application programmers to use ontological information for creating applications without dealing with the details of ontologies. In addition, we explain our plan for supporting ontology query languages. Currently, we are exploring a variant of DAML Query Language (DQL) as our ontological equivalent of SQL (Structured Query Language). We describe the schematic architecture and design considerations of the ontology management system. Also, we describe a range of deployments from lightweight thin-client deployments to enterprise-scale systems for implementing increasing functionalities of the system.

The rest of this paper is structured as follows: In Section 2, we describe user scenarios illustrating how an ontology management system enables applications to manipulate and query ontological information without worrying about the management chores. Section 3 provides a schematic overview of the ontology management system, and also describes its components. Section 4 briefly explains our initial thoughts on ontology query languages. In Section 4, we briefly explain the design of JOBC. In Section 6, we describe a range of deployment configurations for the implementation of the ontology management system. In Section 7, prior work on ontology management systems is evaluated and summarized. In Section 8, conclusions are drawn and future work is outlined.

2 User Scenarios

In this section, we describe the application contexts in which an ontology management system can be useful. We believe that any application dealing with multiple domain concepts that are interrelated and would like to use these concepts to describe the behavior or capabilities of its programs may be benefited by the notion of ontology management. For example, collaboration context management systems, video retrieval systems, text retrieval systems, advanced search engines, business process integration-oriented services and life sciences could all be potential users of an ontology management system.

A collaboration context management system would assert concepts and facts about the users of a particular collaborative process such as Request For Quote (RFQ) processing system. Depending on a particular context, users would interact with other users. Assuming that semantics are used to describe, at a conceptual level, the user job functions and their participation within a general collaborative context, an ontology management system can help a context management system determine, based on the job function, who needs to be contacted for what tasks by inferencing associations between users and their job functions. Specifically, a collaboration context management system can rely on an ontology management system to manage concepts, assert rules and facts, inference relationships, implement the associated ontological working memory and such.

Similarly, in a video retrieval system, the contents of a video could be semantically annotated with concepts from the specific domain which the video is associated with. For example, a television channel would want to retrieve some video footage about a specific company to telecast while presenting that company's quarterly financial results. This concept can be further extended to a business asset library where all assets that a company owns can be annotated and cataloged using semantics for later retrieval.

In a business process integration scenario, application providers can annotate their Web services using semantic markup languages and make these services available for business partners via business service registries such as UDDI. Assume that there are semantic matching services available to help match the requests of service requesters with the services offered by service providers. These matching services would need to reference the ontologies in which the concepts that describe the requirements or capabilities of services are defined, and be able to inference the degree of match between the requested and available services. An ontology management system that can manage the underlying representation models and inferencing can help the matching services significantly in performing the matches. Similar scenarios may be given in other applications such as text retrieval systems, search engines and life sciences areas, where a generic ontology management system could be useful.

3 Schematic Overview

To provide a holistic management support for the entire lifecycle of ontological information, including ontology creation, storage, retrieval, query, reuse, maintenance, and integration, an ontology management system needs to address a wide range of problems. To name a few, it includes ontology models, ontology base design, query languages, programming interfaces, query processing and optimization, especially from federated knowledge sources, various system support (e.g., caching and indexing), transaction support (e.g., ACID property support), distributed system support, and security support. While some of these areas are new challenges for ontology management systems, some are familiar and there has been active research on them, especially in relation to traditional studies on knowledge representation, machine learning and artificial intelligence, or recent studies on semantic Web standards. One of the pragmatic challenges for ontology researchers is how to identify and create missing pieces

in this picture, and to engineer and synthesize them with existing results from prior research work for providing a holistic management system for ontological information.

In our work, we take this challenge with a staged approach to create an industry-strength ontology management system providing reliability, scalability and performance for enterprise scale applications. In this paper, we describe its initial stage where we focus on programming interfaces and query processing. Our system provides a Java API, Java Ontology Base Connector (JOBC), which is the ontological equivalent of the Java Data Base Connector (JDBC). Similarly, the system supports a variant of DAML Query Language (DQL) as our ontological equivalent of SQL (Structured Query Language). We plan to support a few other ontology query languages in the future. Also, we provide the functionality of our system and the deployment options in phases, with the simplest and most lightweight deployments and features offered first.

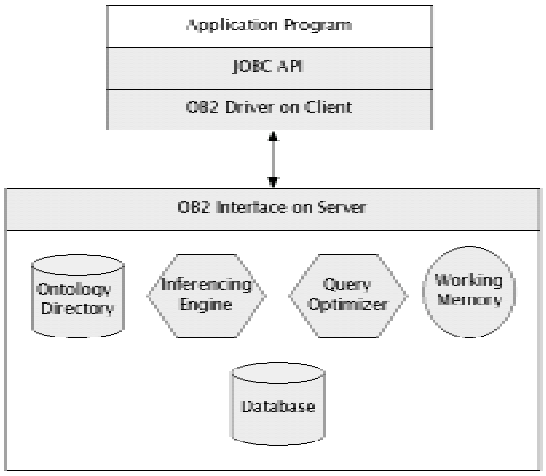

Figure 1. Schematic overview of ontology management

Figure 1 shows the schematic overview of our system in the initial stage. Conceptually, the application programmers interact with the JOBC API that provides high-level access to ontology resources and an ontology engine. The application program interacts with the JOBC API that provides access to an implementation of the API via an ontology base driver. In this case, our driver is OB2. The driver consists of a number of components: a local ontology directory, an inferencing engine, a working memory, a query optimizer and a set of connectors. We explain details on each below.

OB2 Driver: This component is an IBM driver for the JOBC interface that is equivalent to the DB2 driver for JDBC. The OB2 driver consists of Java classes that will provide an implementation of the JOBC API, and contains of a number of components: a local ontology directory, an inferencing engine, a working memory, a query optimizer and a set of connectors, and other infrastructure needed to support ontology management. We explain details on each below.

Ontology Directory: This component provides the meta-level information about which ontologies are available to the JOBC driver. By default, the ontology directory will need to contain the references to the top-level definitions of RDF, RDFSchema, XMLSchema and similar definitions for the set of XML-based ontology languages supported. In addition, the ontology directory may contain deployment information that gives additional sources of ontology information. For each ontology source, the directory will need to store the URI, but may additionally store information about the contents of the ontology source to aid in query optimization.

Inferencing Engine: The inferencing engine provides a mechanism for interpreting the semantics of an ontology language, represented as a set of language specific rules. The rules are used to answer queries, when the requested fact is not immediately available, but must be inferred from available facts. For example, if the application requests the childrenOf an individual, but the working memory only contains parentOf relations, the inferencing engine can use the inverse property statements about childrenOf and parentOf to identify the correct response.

Query Optimizer: For applications that connect to large databases and/or ontologies, it will not be feasible to load the entire set of available information into working memory. Instead, the driver will query the ontology source for appropriate information as it is needed. In addition, the task of the query optimizer is to not only optimize the retrieval of information from ontology sources, but also coordinate queries that span multiple sources.

Ontology Source Connectors: These connectors provide a mechanism for reading, querying, and writing ontology information to persistent storage. The simplest connector is the file connector that is used to persist information to the local file system. In addition, there will be connectors for storing ontological information in remote servers. Also, the connectors are used to implement caching of remote information to improve performance and reliability. For example, it would probably cache the definitions of the top-level ontology definitions RDF, RDFSchema, and XMLSchema to allow the system to work if the W3C website were inaccessible.

4 Query Languages

Currently, we are exploring a variant of DAML Query Language (DQL) as our ontological equivalent of SQL (Structured Query Language) for our ontology management system. DQL is a language and protocol supporting agent-to-agent query-answering dialogues using knowledge represented in DAML+OIL [DAML+OIL, 2001; Fikes *et al.*, 2002]. It precisely specifies the semantic relationships among a query, a query answer, and the ontology base(s) used to produce the answer. It also supports query-answering dialogues in

which the answering agent may use automated reasoning methods to derive answers to queries. To quote from the DQL web site [DQL, 2002], "a DQL query contains a query pattern that is a collection of DAML+OIL sentences in which some literals and/or URIrefs have been replaced by variables. A query answer provides bindings of terms to some of these variables such that the conjunction of the answer sentences - produced by applying the bindings to the query pattern and considering the remaining variables in the query pattern to be existentially quantified - is entailed by a knowledge base (KB) called the answer KB."

We appreciate this query model for its relative simplicity and expressiveness. To make a query, a program simply describes the concept it is searching for, indicating with variables which aspects of matching concepts it is interested in getting as part of a reply. This is similar to the concept of query by example, but with the advantage that the ontology language allows a richer method for describing the examples.

In our system, we initially support a variant of DQL working with JOBC for understanding the requirements of query processing and optimization for ontology management systems. However, we plan to support not only DAML+OIL, but also RDF and the newly released OWL standard for ontology representation. The advantage of the DQL query approach is that the mechanism is easily adaptable to a variety of ontology representation languages. Refer to the summary and evaluation of related ontology query languages in Section 7.

5 Java Ontology Base Connector

We design the Java API for our system, Java Ontology Base Connector (JOBC), as an ontological equivalent of the Java Data Base Connector (JDBC). JDBC provides a connection-based API for interacting with relational data sources. The API is implemented using the abstract factory class. The DataManager class provides a method that is used to construct a connection, based on the URI used to initiate the connection. There is a mechanism in the DataManager that uses the database type specified in the URI to identify and load the correct driver. This driver is then used to create a connection of the appropriate type. The connection then acts as a factory to produce objects, such as statements. The objects created implement interfaces defined in the JDBC package, but have implementations that are provided by the driver that is loaded.

We intend to follow a similar design pattern when implementing JOBC, with several alterations. We intend to allow connections to be made without reference to a particular base ontology. Such connections would be connections to a default ontology that would include the top-level definitions of OWL, DAML+OIL, RDF, XML and XMLSchema. These top-level definitions are required in order to process any ontological information.

The following code sample illustrates the use of JOBC.

```
Connection connection =
           DriverManager.getConnection();
RDFLiteral john =
           connection.createLiteral("john");
RDFLiteral isA =
           connection.createLiteral("isA");
RDFLiteral teacher =
           connection.createLiteral("teacher");

/* We put a statement in Working Memory:
 * john isA teacher
*/
RDFStatement statement =
 connection.createStatement(john, isA, teacher);
connection.assert(statement);
RDFStatementCollection query =
    connection.createStatementCollection();
query.addStatement(statement);
RDFResultSet resultSet =
    connection.select(query);
```

In this example, the code first gets a connection. This connection is then used to create literals and a statement (John isA teacher). This statement is then asserted into working memory. The code then creates a query for the asserted fact and retrieves it from working memory.

More complex queries can be implemented using variables. For example, the following query requests all teachers.

```
RDFVariable X = connection.createVariable("X");
RDFStatement queryStatement1 =
   connection.createStatement(X, isA, teacher);
```

The results of such a query are a set of tuples that include the binding(s) of the variable(s) in the query.

6 Deployment Configurations

Initially, the OB2 driver only consists of the client-side driver for connecting to existing ontology sources that are served up as files and JDBC connections. However, the same infrastructure will be useful in implementing server-side ontology management. In such a deployment, the server will execute ontology queries against its local data and not require transmission of the data to the inferencing engine on the client to process queries. Server-based ontology query processing also allows for increased performance, since the data on the server can more easily have extensive indexing and preprocessing. This section describes the deployment options and the corresponding functionality of our system, from the most lightweight deployment to complex ones.

Thin-Client: In this configuration, JOBC provides access to ontology information that is available via Web-based protocols. Such a system would be appropriate for use with applications that require to reference standard ontology information available from W3C and other

standards bodies. Figure 2 illustrates the configuration of this deployment.

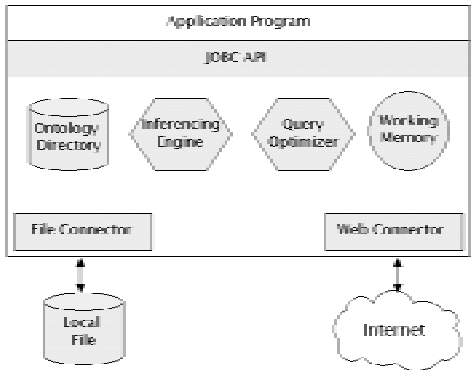

Figure 2. Deployment of the thin-client configuration

Fat-Client: A fat-client provides local persistence for ontology information that the application generates. Depending on the configuration details, information may be stored locally as files or in a database for increased performance and reliability. The local prescient storage can also provide local caching for ontological information that is accessed via the Web. Figure 3 illustrates the configuration of the fat client deployment.

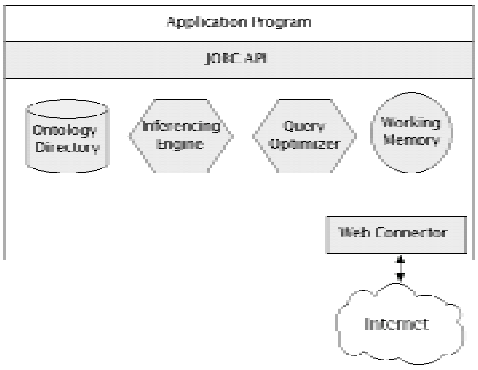

Figure 3. Deployment of the fat-client configuration

Client-Server: This configuration models a full-fledged ontology management system with a complete set of functionalities. To provide ontology sharing and evolution among a large number of clients, the ontology management functions would be moved to an ontology server. Not only does this arrangement provide for easier sharing and updating of ontologies in a distributed environment, it also allows for additional optimization and indexing that might be too costly to perform on a fat client. By linking ontology management to a database, database tables and indexes can be optimized to support query and retrieval. The ontology information can also be preprocessed to speed up the handling of the most common queries. Also, this configuration can be further extended to accommodate federated sources of ontological information.

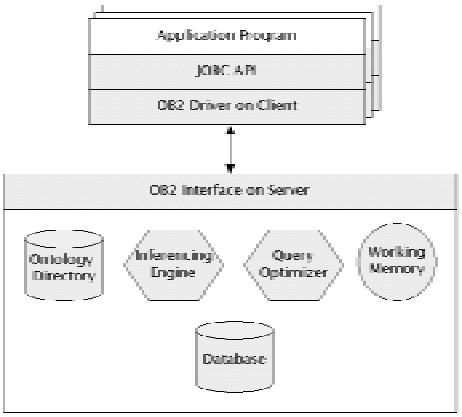

Figure 4. Deployment of the client-server configuration

7 Related Work

One of the notable studies in ontology management is the Open Knowledge Base Connectivity (OKBC) [Chaudhri *et al.*, 1998], which shares somewhat similar objectives with JOBC. It is a protocol for accessing knowledge bases stored in Knowledge Representation Systems (KRSs). The API has been developed to address the issue of knowledge base tool reusability and was implemented in various languages including Java. It provides a set of operations with a generic interface to underlying KRSs, so that it provides applications with independence from the idiosyncrasies of specific KRS and enables the development of generic tools.

One of the challenges faced by ontology management system is merging and diagnosing of existing ontologies. Merging is important when multiple terminologies must be used and viewed as one ontology or when distributed team members need to assimilate two or more ontologies that should work together in an integrated fashion. Diagnosis of ontologies becomes critical when ontologies are obtained from varied sources. One notable system addressing this problem is Chimaera, a merging and diagnostic web-based browser ontology environment [McGuinness *et al.*, 2000]. It is reported that Chimaera accepts over 15 designated input format choices as well as any other OKBC-compliant forms.

The ICS-FORTH RDFSuite [Alexaki *et al.*, 2001] is a suite of tools for RDF metadata management. It consists of tools including a validating RDF parser, the RDF Schema Specific DataBase (RSSDB), and the RDF Query Language (RQL). RSSDB is a persistent tool for loading resource descriptions in an object-relational DBMS (e.g., Postgress) by exploiting the available RDF schema knowledge. Querying of the stored RDF descriptions is

processed by a query module that pushes query evaluation to the underlying DBMS.

Jena [McBride, 2001] is a collection of RDF tools written in Java that includes a Java model/graph API, an RDF parser, a query system, support classes for DAML+OIL ontologies, and persistent and in-memory storage. Jena provides statement-centric methods for manipulating an RDF model as a set of RDF triples and resource-centric methods for manipulating an RDF model as a set of resources with properties.

There has been active work done in the areas of ontology query. In addition to DQL described in Section 4, there are a few other ontology query languages of interest. TRIPLE is an RDF query, inference, and transformation language for the semantic Web [Sintek *et al*., 2001]. It allows the semantics of languages on top of RDF to be defined with rules. It also allows reasoning and transformation under several different semantics, which is necessary if you need to access multiple ontology sources in an application.

RQL (RDF Query Language) is a typed, declarative query language for querying RDF description bases [Karvounarakis *et al*., 2002]. It is defined by a set of basic queries and iterators, which can be used to build new ones through functional decomposition. The novelty of RQL lies in its ability to combine ontology and data querying while exploiting the taxonomies of labels and multiple classifications of resources. That is, users are able to query resources described by the preferred ontology, while discovering in the sequel how the same resources are described in a different ontology.

RDFQL is an SQL-style, statement-based query language that supports the RDF model and syntax. It provides facilities for querying RDF structured data as well as the ability to infer new statements from existing ones. As an SQL-style language, it also uses commands such as INSERT, DELETE and SELECT to perform query and inference operations on RDF triples, as well as data definition commands like CREATE TABLE or CREATE VIEW.

8 Concluding Remarks

An increasing range of applications require an ontology management system that helps externalize ontological information for a variety of purposes in a declarative way. The primary objective of ontology management systems is to provide holistic control over management activities for ontological information by externalizing them from application programs. Ontology management systems provide ontology independence to applications in a similar way that database management systems provide data independence. One of the pragmatic challenges for ontology management system research is how to create missing component technology pieces, and to engineer them with existing results from prior research work for providing a holistic management system.

In this paper, we described our project for developing an industry-strength ontology management system that will provide reliability, scalability, and performance for enterprise uses, and also provide functionality robust and sufficient for different levels of practitioners of ontological information. We described the design and implementation of an ontology management system, which is under development at IBM T. J. Watson Research Center. For this work, we take this challenge with a staged approach. We described its initial stage where we focus on programming interfaces and query processing. Our system provides a Java API, Java Ontology Base Connector (JOBC), which is the ontological equivalent of the Java Data Base Connector (JDBC). Similarly, this system supports a variant of DAML Query Language (DQL) as our ontological equivalent of SQL (Structured Query Language), and will support a few other ontology query languages in the future. Also, we described a range of deployments from lightweight thin-client deployments to enterprise-scale systems with increased functionality of the system.

References

[Alexaki *et al*., 2001] S. Alexaki *et al*. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. *Proceedings of the 2nd International Workshop on the Semantic Web*, 2001.

[Chaudhri *et al*., 1998] V. Chaudhri *et al*. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. *Proceedings of AAAI-98*.

[DAML+OIL, 2001] DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.

[DQL, 2002] DQL. http://www.daml.org/2002/08/dql/dql.

[Fikes *et al*., 2002] R. Fikes, P. Hayes, and I. Horrocks. DQL - A Query Language for the Semantic Web. Stanford University 2002.

[Karvounarakis *et al*., 2002] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. Proceedings of WWW2002, May 7-11, 2002.,

[McBride, 2001] B. McBride. Jena: Implementing the RDF Model and Syntax Specification. *Proceedings of the Second International Workshop on the Semantic Web*, May 2001.

[McGuinness *et al*., 2000] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. *Proceedings of AAAI 2000*.

[Sintek *et al*., 2001] M. Sintek, and S. Decker. TRIPLE - An RDF Query, Inference, and Transformation Language for the Semantic Web. *Proceedings of the International Semantic Web Conference*, 2001.