RC23730 (W0509-109) September 30, 2005
Computer Science

IBM Research Report

Ontology Management for Large-Scale Enterprise Systems

Juhnyoung Lee, Richard Goodwin
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

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

Ontology Management for Large-Scale Enterprise Systems

Juhnyoung Lee and Richard Goodwin

IBM T. J. Watson Research Center
Hawthorne, NY 10532
U.S.A.
{jyl, rgoodwin}@us.ibm.com

May 30, 2004

Abstract

Semantic markup languages such as RDF (Resource Description Framework) [31] and OWL (Web Ontology Language) [35] are increasingly being used to externalize meta-data or ontologies about data, software and services in a declarative form. Such externalized descriptions in ontological format are used for purposes ranging from search and retrieval to information integration and to service composition [32, 36]. Ontologies could significantly reduce the costs of deploying, integrating and maintaining enterprise systems. The barrier to more wide-spread use of ontologies for such applications is the lack of support in the currently available middleware stacks used in enterprise computing. This paper presents our work on developing an enterprise-scale ontology management system that will provide APIs and query languages, and scalability and performance that enterprise applications demand. We describe the design and implementation of the management system that programmatically supports the ontology needs of enterprise applications in a similar way a database management system supports the data needs of applications. In addition, we present a novel approach to representing ontologies in relational database tables to address the scalability and performance issues. The state of the art ontology management systems are either memory-based or use ad-hoc solutions for persisting data, and so provide limited scalability.

Keywords: Semantic Web, ontology, management system, inference

1. Introduction

In recent years, there has been a surge of interest in using ontological information for communicating knowledge among software systems. Particularly, the effort has been lead by the semantic Web initiative by W3C [33]. As a result, an increasing range of software systems need to engage in a variety of ontology management tasks, including the creation, storage, search, query, reuse, maintenance, and integration of ontological information. Recently, there have been efforts to externalize such ontology management burden from individual software systems and put them together in middleware known as an ontology management system. An ontology management system provides a mechanism to deal with ontological information at an appropriate level of abstraction. By using programming interfaces and query languages the ontology management system provides, application programs can manipulate and query ontologies without the need to know their details or to re-implement the semantics of standard ontology languages. Such a setting is analogous to the way a database management system allows

applications to deal with data as tables and provides a query engine that can understand and optimize SQL queries.

In this paper, we describe the design and implementation of the SnoBase ontology management system [34], which is being developed at IBM T. J. Watson Research Center. While there are a number of projects for building ontology support tools [4, 6, 16, 18, 25, 27, 29], the SnoBase project is different from others in its objective of developing an industry-strength ontology management system. The design of the SnoBase system focuses on providing reliability, scalability, and performance for enterprise computing, and also providing functionality robust and sufficient for different levels of practitioners of ontological information.

To make the system fit well into the current software development environment and reduce rather than increase the burden on software architects, programmers and administrators, we synthesize concepts familiar to software developers with ideas from the semantic Web and ontology communities. The SnoBase system programmatically supports the ontology needs of applications in a similar way a database management system supports the data needs of applications. For programmers, SnoBase provides a Java API referred to as Java Ontology Base Connector (JOBC), which is the ontological equivalent of Java Data Base Connector (JDBC) [24]. 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 SnoBase ontology management system supports a number of query languages. At present, SnoBase supports a variant of OWL Query Language (OWL-QL) [7] and RDF Query Language (RDQL) [30] as ontological equivalents of SQL (Structured Query Language) of relational database systems.

In addition, to address the scalability and performance issues of the state of the art ontology management systems that are either memory-based or use ad-hoc solutions for persisting data, SnoBase provides a novel approach to representing ontologies directly in database tables and providing logical reasoning by using plain SQL triggers. The provision of inference in an ontology management system requires the description and enforcement of semantics of semantic markup language constructs as rules. It turns out that most useful part of the semantics of W3C semantic Web markup languages (such as RDF and OWL) can be effectively expressed as plain SQL triggers. The execution of the triggers automatically enforces the semantics of the ontology markup language constructs. With this database-based reasoning approach, an ontology management system gains the scalability, reliability and query performance of the mature relational database system without extra cost. This paper describes the schematic architecture and design considerations of the ontology management system.

The rest of this paper is structured as follows: In Section 2, we describe several enterprise application contexts in which an ontology management system can be useful. Section 3 provides technical background information on ontology using an example and discusses technical challenges on its management support. In Section 4, we explain how we address the challenges and provide a schematic overview of the SnoBase ontology management system. Additionally, we briefly describe each component of the SnoBase system. Section 5 explains on ontology query languages which the SnoBase system supports. In Section 6, we describe the design of JOBC API with examples. Section 7 presents an approach to representing ontologies directly in database tables and providing logical reasoning capabilities by using plain SQL triggers. Section 8 summarizes the previous work on the ontology management problem, and discusses how the presented work is different from them. In Section 9, conclusions are drawn and future work is outlined.

2. Use Cases in Enterprise Computing

Ontology is similar to a dictionary, taxonomy or glossary, but with structure and formalism that enables computers to process its content. It consists of a set of concepts, axioms, and relations, and represents an area of knowledge. Unlike taxonomy or glossary, ontology allows to model arbitrary relations among concepts, also model logical properties and semantics of the relations such as symmetricity, transitivity and inverse, and logically reason about the relations. Ontology is often specified in a declarative form by using semantic markup languages such as RDF and OWL. It provides a number of potential benefits in processing knowledge, including the separation of domain knowledge from operational knowledge, sharing of common understanding of subjects among human and also among computer programs, and the reuse of domain knowledge.

This section describes several enterprise application contexts in which an ontology management system can be useful. In addition to these examples, ontology management can be beneficial to any system dealing with multiple domain concepts that are interrelated and needs to use the concepts to describe the behavior or capabilities of its programs. Some other examples would include information retrieval and search systems for semantic-based search capabilities, video retrieval systems to annotate media with metadata, and collaboration management systems to provide a common understanding to collaboration contexts and annotate them.

2.1. Business Process Integration with Web Services

Web services facilitate the creation of business process solutions in an efficient, standard way. However, the process integration with Web services requires the automation of discovery and composition of Web services. Ontologies are applied to resolve two basic issues with the Web service-based business process integration: the discovery of Web services based on the capabilities and properties of published services, and the composition of business processes based on the business requirements of submitted requests. 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. Suppose that there is a semantic matching service available to help match service requests with services offered by providers. Such a matching service would need to refer to ontologies in which domain knowledge describing the requirements or capabilities of services is defined. Then, it would be able to infer degrees of match between the requested and available services. An ontology management system that can manage the underlying ontology representation models and reasoning will enable the matching service to perform such semantic-based matches [1].

2.2. Collaboration Using Corporate Social Networks

In any large organization such as a university, a company and a government, employees have to collaborate with a wide variety of people in order to perform different kinds of tasks. They collaborate with people, not only in their own group or department, but also with people in other departments, particularly, service departments such as Human Resources, Legal, Finance, IT, Purchasing, Facilities, Public Relations, etc. Large organizations often have specific people assigned for these tasks that an employee can contact. However, this contact information is often not available explicitly in the organization database. Therefore, it becomes difficult for a person, especially a new employee, to discover his/her assigned contact for accomplishing a certain kind of task. A possible approach to

alleviating this problem is to use the concept of "social networks" based on the "friend" relation among people [11]. The informal social networks of an organization setting can be further extended with richer ontological information on the types of relations. For example, each employee in the corporate social network has links to various kinds of contacts (e.g., HR, Legal, IT, etc.). The system then explores the corporate social network to find the right contact for a certain task. It was reported that one of the most popular applications of RDF is the Friend of a Friend (FOAF) project, which is about creating a Web of machine-readable homepages describing people, links between them and things they create and do [13, 32].

2.3. Model-Driven Business Transformation

An approach to business transformation is to employ business models such as process-oriented models or business component models to identify opportunities for saving costs or improve business processes [20]. The model-driven approach requires a model representation of business entities such as business processes, components, competencies, activities, resources, metrics, KPIs (Key Performance Indicators), etc. and their relations. Semantic models using ontology markup languages provide useful representation of business models because they are not limited in representing different types of relations among business entities. Also, the automatic reasoning capability of semantic models provides an effective method for analyzing business models for identifying cost-saving or process improvement opportunities. For example, business performance metrics are naturally fit well with business activities and traditionally represented that way. By using this relation between business activities and metrics, and also the relation between business components and business activities represented in a semantic model, a business analyst can infer relations between business components and metrics. This relation can provides business insights into how the corporate can improve its performance metrics by addressing issues with the business components associated with the selected set of metrics. Then, by identifying, again in the semantic model, IT systems (a type of resources) associated with the business components, the analysts may be able to suggest recommendations about IT system management to improve performance metrics.

3. Technical Background

Figure 1 illustrates a simple example of ontology, which represents knowledge of a university domain [11]. The ontology is displayed as a tree, as shown in a popular ontology editor, Protégé [29]. In this simple example, all the relations among nodes are subClassOf. Note that ontology, in general, represents a graph, not a tree, because it allows arbitrary relations among nodes, and also allows a node to inherit more than one parents. In this example, PhDStudent is a subclass of two classes, Researcher and GraduateStudents.

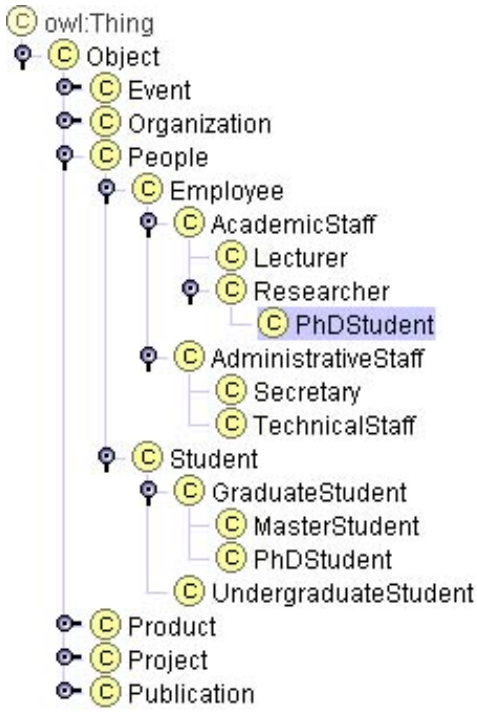

Figure 1 University ontology

As described earlier, an ontology is often specified in a declarative form by using semantic markup languages. The Semantic Web initiative of W3C is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The initiative utilizes XML and XML Schema's ability to define customized tagging schemes and RDF's flexible approach to representing data. RDF provides a simple semantics for data, and the data models can be represented in an XML syntax. In addition, the initiative introduces an ontology language, OWL (Web Ontology Language), which can formally describe the meaning of terms. OWL is part of the growing stack of W3C recommendations related to the Semantic Web. Figure 2 displays a portion of the OWL representation of the university ontology introduced in Figure 1. It shows a number of classes related to people in the domain and their subClassOf relations by using OWL and RDF constructs.

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns="http://a.com/ontology#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xml:base="http://a.com/ontology">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="People">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Object"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Employee">
    <rdfs:subClassOf rdf:resource="#People"/>
  </owl:Class>
  <owl:Class rdf:ID="AcademicStaff">
    <rdfs:subClassOf rdf:resource="#Employee"/>
  </owl:Class>
  <owl:Class rdf:ID="Researcher">
    <rdfs:subClassOf rdf:resource="#AcademicStaff"/>
  </owl:Class>
  <owl:Class rdf:ID="Lecturer">
    <rdfs:subClassOf rdf:resource="#AcademicStaff"/>
  </owl:Class>
  <owl:Class rdf:ID="PhDStudent">
    <rdfs:subClassOf rdf:resource="#GraduateStudent"/>
    <rdfs:subClassOf rdf:resource="#Researcher"/>
  </owl:Class>

  ...
```

Figure 2 University ontology representation in OWL

An ontology is expected to allow machines to perform useful reasoning tasks on documents and system it annotates with its concepts and relations. An inference engine provides this reasoning functionality. It extracts all the facts from the given ontology, and asserts them into its working memory. Also, it is equipped with a knowledge base which stores a set of action rules for interpreting constructs of RDF and OWL.

Before asserting the facts initially specified in OWL into the working memory, the inference engine often utilizes an OWL parser and translates them into a language, Notation 3 or N3 [28], which is simplified, but basically equivalent to RDF and OWL in computational power. In RDF and OWL, information is simply a collection of statements, each with a subject, verb and object, and nothing else. In N3, facts are written as triples, verb (subject, object). Figure 3 shows a portion of facts extracted from the university ontology, translated into N3, and numbered. They will be asserted into the working memory of inference engine for reasoning. Note that this set of facts is basically the same set shown in Figure 2, only in a different notation, i.e., N3.

```
1    http://www.w3.org/1999/02/22-rdf-syntax-ns#type(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#Object,
     http://www.w3.org/2002/07/owl#Class)

2    http://www.w3.org/1999/02/22-rdf-syntax-ns#type(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#People,
     http://www.w3.org/2002/07/owl#Class)

3    http://www.w3.org/2000/01/rdf-schema#subClassOf(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#People,
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#Object)

4    http://www.w3.org/1999/02/22-rdf-syntax-ns#type(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#Employee,
     http://www.w3.org/2002/07/owl#Class)

5    http://www.w3.org/2000/01/rdf-schema#subClassOf(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#Employee,
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#People)

6    http://www.w3.org/1999/02/22-rdf-syntax-ns#type(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#AcademicStaff,
     http://www.w3.org/2002/07/owl#Class)

7    http://www.w3.org/2000/01/rdf-schema#subClassOf(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#AcademicStaff,
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#Employee)

8    http://www.w3.org/1999/02/22-rdf-syntax-ns#type(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#Researcher,
     http://www.w3.org/2002/07/owl#Class)

9    http://www.w3.org/2000/01/rdf-schema#subClassOf(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#Researcher,
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#AcademicStaff)

10   http://www.w3.org/1999/02/22-rdf-syntax-ns#type(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#PhDStudent,
     http://www.w3.org/2002/07/owl#Class)

11   http://www.w3.org/2000/01/rdf-schema#subClassOf(
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#PhDStudent,
     http://rgoodwin.watson.ibm.com/snoscape/university.owl#Researcher)

...
```

Figure 3 Asserted facts of the university ontology in N3 notation (part)

An essential component of ontology management systems is the inference engine, which 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 inference engine can use the inverse property statements about childrenOf and parentOf to identify the correct response. For the inference engine component, most ontology management systems depend on pattern

matching algorithms such as Rete [8] originated from earlier work on the rule-based systems [2]. These systems comprises a working memory comprised of a set of facts representing the current status of the system, a knowledge base which stores a set of condition action rules, and a rule interpreter which applies the rules to the working memory. The pattern matching algorithms find all rules that are eligible to be fired by matching antecedent of rules to facts in working memory. If rules have variables, matching requires unification. The Rete algorithm does it efficiently because it avoids searching all rules every cycle by storing changes.

The inference engines of most ontology management systems currently available are based on pattern matching algorithms developed for expert systems in the 1970's and 1980's. Their model for working memory and knowledge base is memory-based or uses ad hoc solutions for managing data. While this model is adequate for dealing with class hierarchies in small to medium scales, it does not scale for applications that involve large-scale ontologies and large amounts of instance data. We will explain how we address this scalability problem with the inference engines of ontology management systems in Section 6. First, we will describe the architecture and APIs of the SnoBase ontology management system.

4. SnoBase Ontology Management System

Ontologies are becoming increasingly prevalent and important in a wide range of enterprise applications. They are used to support parametric searches, enhanced navigation and browsing, to integrate heterogeneous data sources and applications, to configure software, products and services, and to qualitatively analyze business processes and components. In addition, applications such as information and (Web) service discovery and composition, and autonomous agents that are built on top of the Semantic Web require extensive use of ontologies.

One of challenges in the design of an industry-strength ontology management system is the versatility of Application Programming Interfaces and query languages for supporting such as diverse applications. This objective requires a careful design to simultaneously satisfy seemingly conflicting objectives such as being simple, easy-to-use for the users, and easy-to-adopt for the developers. Another challenge in ontology management is to provide ontology-enhanced industrial applications with a system that is scalable (supporting thousands of simultaneous distributed users), available (running 365x24x7), fast, and reliable. These non-functional features are essential not only for the initial development and maintenance of ontologies, but also during their deployment.

To provide a holistic management support for the entire lifecycle of ontological information, including ontology creation, storage, search, query, reuse, maintenance and integration, an ontology management system needs to address a wide range of problems; ontology models, ontology base design, query languages, programming interfaces, query processing and optimization, federation of knowledge sources, caching and indexing, transaction support, distributed system support, and security support, to name a few. While some of these areas are new challenges for ontology management systems, some are familiar and there have been active studies, particularly in relation to traditional studies on knowledge representation, or recent studies on semantic Web standards. Our approach to the ontology management support is a pragmatic one, that is, we identify missing pieces in this picture, and engineer and synthesize them with prior work for providing a holistic management system for ontological information.

Figure 4 shows a schematic overview of the SnoBase ontology management system. Conceptually, the application programs interact with the JOBC API that provides high-level access to ontology resources and the ontology engine. The application program interacts with the JOBC API that provides an access to an implementation of the API via an ontology base driver. In this case, our driver is the SnoBase driver. In this section, we will describe the each component of the SnoBase ontology management system.

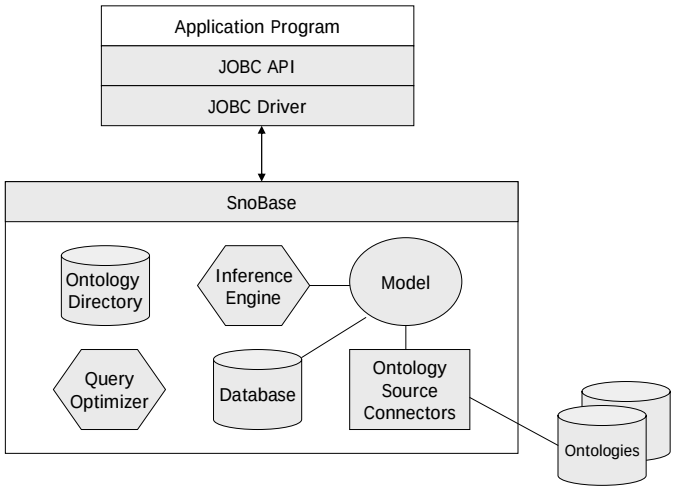

Figure 4 SnoBase ontology management system architecture

4.1. JOBC API

The SnoBase 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). The JOBC API follows the design patterns of JDBC, with several alterations. Just like JDBC, JOBC provides a connection-based interaction between applications and ontology sources. Also, JOBC provides JDBC-style, cursor-based result sets for representing query results. The similarity of JOBC to JDBC was a design decision to help application developers of SnoBase can quickly learn the programming style of JOBC from their previous experience of the popular JDBC protocol. One difference between JOBC and JDBC is that JOBC allows connections to be made without reference to a particular base ontology. Such connections provide an access to default ontologies of the top-level definitions of XML-based ontology languages such as OWL, RDF, RDF Schema and XML Schema. These definitions are required in order to process any ontological information.

4.2. SnoBase Driver

This component is an IBM driver for the JOBC interface that is equivalent to the IBM DB2 driver for JDBC. The SnoBase 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 inference engine, a working memory, a query optimizer and a set of connectors, and other infrastructure needed to support ontology management.

4.3. Ontology Directory

This component provides the meta-level information about ontologies that are available to the SnoBase driver. By default, the ontology directory contains the references to the top-level definitions of OWL, RDF, RDF Schema, XML Schema, and similar definitions for the set of XML-based ontology languages supported. In addition, the ontology directory provides metadata such as deployment information and 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.

4.4. Inference Engine

This component 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 inference engine can use the inverse property statements about childrenOf and parentOf to identify the correct response. The details of this component, different approaches to implementing this component and issues of the scalability and performance will be discussed in Section 6.

4.5. 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.

4.6. 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 store 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 cache the definitions of the top-level ontology definitions OWL, RDF, RDF Schema, and XML Schema to allow the system to work if the W3C Web site were inaccessible.

5. Query Languages

Currently, the SnoBase system supports a variant of OWL Query Language (OWL-QL) as an ontological equivalent of SQL (Structured Query Language). OWL-QL is a language and protocol supporting agent-to-agent query-answering dialogues using knowledge represented in OWL. It precisely specifies the semantic relations 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. An OWL-QL query contains a query pattern that is a collection of OWL sentences in which some literals and/or URIs 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.

This design provides a simple but expressive query model. 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 receiving as part of a reply. This query model 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. Another advantage of the OWL-QL query approach is that the mechanism is easily adaptable to a variety of ontology representation languages. In addition to OWL-QL, SnoBase also supports another ontology query language, RDQL, whose specification was submitted to W3C for a possible recommendation. RDQL is similar to OWL-QL in its underlying query mechanism.

6. Java Ontology Base Connector

As described earlier, we designed the JOBC API for SnoBase as an ontological equivalent of the Java Data Base Connector (JDBC). The API is implemented using the abstract factory pattern [10]. An abstract factory class defines methods to create an instance of each abstract class that represents a user interface widget. Concrete factories are concrete subclasses of an abstract factory that implements its methods to create instances of concrete widget classes for the same platform. 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 follow a similar design pattern in the implementation of JOBC, with several alterations, as described above. The following code sample illustrates the use of JOBC.

```
/* We connect to an ontology resource. */
Connection connection = DriverManager.getConnection();
RDFResource john = connection.createRDFResource("John");
RDFProperty isA = connection.createRDFProperty("isA");
RDFClass researcher = connection.createRDFClass(“researcher");

/* We assert a statement in inference engine: John isA researcher. */
Statement statement = connection.createStatement(John, isA, researcher);
connection.assert(statement);

/* We create a simple query. */
StatementCollection query = connection.createStatementCollection();
query.addStatement(statement);
ResultSet resultSet = connection.select(query);
```

Figure 5 A JOBC Statement

In this example, the code first gets a connection. This connection is then used to create resources and a statement (john isA researcher). This statement is then asserted into the inference engine. The code then creates a simple query for the asserted fact and retrieves it from the working memory of the inference engine. More complex queries can be implemented using variables. For example, the following query requests all who are researchers.

```
/* We form a query: show me all who isA researcher. */
Variable X = connection.createVariable("?X");
Statement queryStatement = connection.createStatement(X, isA, researcher);

/* We create a simple query. */
query.addStatement(queryStatement);
resultSet = connection.select(query);
```

Figure 6 A JOBC query

The results of such a query are a set of triples that include the binding(s) of the variable(s) in the query. In this case, the variable X is bound to john. Using these basic APIs, SnoBase programmers can build more complicated queries. For example, a query may contain multiple variables and multiple (query) statements. Also, note that SnoBase does not simply retrieve information previously stored for queries. Instead, by using an inference engine, it infers for answering facts that are not immediately available.

7. Direct Representation of Ontologies in a Database

State of the art inference engines for ontology management systems are either memory-based or use ad-hoc solutions for persisting data (for a survey of ontology storage systems, see [14]). While this is adequate for dealing with the class hierarchies in small to medium-size ontologies, it does not scale for applications that involve large amounts of instance data. This is due to the emphasis that is placed on

the metadata (hierarchy of classes or concepts) as first-class citizen as opposed to the data (in-stances of classes).

However, many new application domains of enterprise computing such as e-commerce and social networks, and bioinformatics deal with large amounts of pre-existing data that needs to be linked to an ontology. Current solutions recommend migration of the existing data into the ontology data structures. However, if other applications still use that data, this approach requires constant replication to keep the two versions in sync. Moreover, typical ad-hoc storage solutions do not provide the same level of support for data integrity, concurrent access, and recovery as a mature database management system. We believe that approaches that create custom storage systems may suffer from severe limitations in the long run, such as problems arising from the need to integrate different ontologies, or ontologies with existing data in terms of scalability and flexibility.

To overcome these limitations, we propose exploring database-centric architectures for storing and manipulating ontological data. Such solutions will take advantage of existing standards for data management and the DBMS features that have been optimized over the years (robustness, concurrency control, recovery, scalability). This is, however, just the first step towards leveraging relational database systems for large-scale ontology data management (ODM). Due to the inherent complexity of ontological queries, a straightforward database implementation of an ontology system may not perform optimally. The reason is that database systems are not optimized for this type of application. We identify several promising directions of future research with the hope of stimulating the interest of the database community in supporting efficient management of ontological data.

In this section, we introduce a solution to the lack of scalability and performance problem of the traditional approach to reasoning in ontology management systems. This solution improves reasoning for ontologies by directly representing facts in relational database tables and representing action rules in SQL triggers. This solution pro-vides a single table called Fact Table, which stores facts asserted by ontology. The table is designed to store facts expressed in N3 notation. Also, this solution provides a set of triggers which are fired when a statement is inserted, updated, or deleted from the Fact Table. Facts added by the triggers are called derived facts, as opposed to asserted facts which are originated from the ontology. The Fact Table stores these derived facts along with the keys (IDs) of statements that caused their existence. Those keys are called justifications. Note that the justification fields of asserted facts are null. The justification fields are important because they are used to delete de-rived statements when their justification statements are deleted from the Fact Table. The deletion of derived facts is also automatically managed by using triggers.

Figure 7 displays the data schema of the Fact Table, which shows the fields for three parts of statements, subject, verb and object, justification fields, along with the key field. Note that the fields for subject, verb and object are foreign keys to an ancillary table which stores actual strings. The model field represents the scope of the given statement, but it is not directly related to this discussion.

```
// Create fact_table

CREATE TABLE FACT_TABLE
(
        FACT_ID BIGINT DEFAULT AUTOINCREMENT INITIAL 1 INCREMENT 1 NOT NULL,
        SUBJECT BIGINT NOT NULL,
        VERB BIGINT NOT NULL,
        OBJECT BIGINT NOT NULL,
        MODEL VARCHAR(256) NOT NULL,
        JUSTIFICATION_1 BIGINT,
        JUSTIFICATION_2 BIGINT,
        JUSTIFICATION_3 BIGINT
)
```

Figure 7 Data schema for Fact Table

Figure 8 displays a couple of triggers that enforce the semantic of subClassOf relations, namely, the subClassOf relation is transitive. If a statement inserted into the Fact Table completes the rule's antecedent, then the trigger adds a derived fact as a consequent. There can be many rules represented in plain SQL triggers in the system to implement the meaning of constructs of RDF and OWL.

```
// (A subClassOf B) and (B subClassOf C) => (A subClassOf C)

CREATE TRIGGER SUBCLASSCLASS1
        AFTER INSERT
        ON FACT_TABLE
        REFERENCING NEW ROW AS inserted_fact
        FOR EACH ROW
        INSERT INTO fact_table (subject, verb, object, model, just_1, just_2)
        SELECT DISTINCT inserted_fact.subject, verb, object, model, inserted_fact.fact_id, fact_id
        FROM fact_table
        WHERE verb = 2
                AND verb = inserted_fact.verb
                AND subject = inserted_fact.object
                AND model = inserted_fact.model

CREATE TRIGGER SUBCLASSCLASS2
        AFTER INSERT
        ON FACT_TABLE
        REFERENCING NEW ROW AS inserted_fact
        FOR EACH ROW
        INSERT INTO fact_table (subject, verb, object, model, just_1, just_2)
        SELECT DISTINCT subject, verb, inserted_fact.object, model, fact_id, inserted_fact.fact_id
        FROM fact_table
        WHERE verb = 2
                AND verb = inserted_fact.verb
                AND object = inserted_fact.subject
                AND model = inserted_fact.model
```

Figure 8 Triggers for deriving implied facts from asserted facts (part)

Figure 9 displays a few facts derived by the triggers shown in Figure 8. In this simple example, it is relatively straightforward to see the justifications of each derived statement, as shown in Figure 10.

Note that firing of a rule sometimes can cause chains of trigger firing. Also, note that a statement can be derived by multiple combinations of justification statements. Therefore, the solution should provide a mechanism that prevents a statement from being added multiple times to the Fact Table.

101 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Employee,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Object)

102 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#AcademicStaff,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#People)

103 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#AcademicStaff,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Object)

104 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Researcher,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Employee)

105 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Researcher,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#People)

106 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Researcher,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Object)

107 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#PhDStudent,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#AcademicStaff)

108 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#PhDStudent,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Employee)

109 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#PhDStudent,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#People)

110 http://www.w3.org/2000/01/rdf-schema#subClassOf(
http://rgoodwin.watson.ibm.com/snoscape/university.owl#PhDStudent,
http://rgoodwin.watson.ibm.com/snoscape/university.owl#Object)

...

Figure 9 Derived facts of the university ontology (part)

```
101:    (3, 5)
102:    (5, 7)
103:    (7, 101), or (3, 102)
104:    (7, 8)
105:    (8, 102), or (5, 104)
106:    (8, 103), (101, 104) or (3, 105)
107:    (8, 10)
108:    (10, 104), or (7, 107)
109:    (10, 105), (102, 107), or (5, 108)
110:    (10, 106), (103, 107), (101, 108), or (3, 109)
```

Figure 10 Justifications of derived facts

With statements both asserted and derived in relational database tables, queries to the ontology management system become SQL queries to the relational database tables. Most ontology management systems available today uses certain query languages which express queries with a collection of N3 statements in which some parts are replaced by variables. A query answer provides bindings of terms to some of these variables. As explained earlier, the query language of the IBM's SnoBase system which is loosely based on OWL-QL follows this query pattern. The solution presented in this section provides a query processor which transforms ontology queries into SQL queries against the Fact Table.

Provision of an inference mechanism which is purely based on mature relational database technology and SQL would resolve the problems with scalability and performance of the traditional approach based on rule-based expert systems. In addition, query optimization techniques known to relational databases such as indexing, schema optimization, and performance tuning can be applied to this proposed approach and further improve the performance of ontology query execution.

8. Model-Driven Approach to a Semantic Toolkit

Until now, we have focused on the description of the application programming interfaces, query languages, and inference engines of the SnoBase Ontology Management System. In this section, we will describe its environment for application and model development and transformation, which is equally important in the adoption of the semantic technology in the industry. The work presented in this section is a result from our collaboration with researchers at IBM's China Research Lab.

Participating in a number of real-world applications by using the SnoBase Ontology Management System, we have learned that it is critical to provide a comprehensive development environment including supporting tools for the application developers. A pick-and-choose approach to the best of the breed tools from different environments does not always work well for the majority of the developers and often results in a longer learning curve for the developers. A comprehensive ontology development environment often means a tight integration of various tools for application development, ontology development, model import and transformation, among others. Semantic markup languages such as W3C's RDF and OWL are based on the work in the logic and AI (Artificial Intelligence) communities, such as Description Logic and Knowledge Representation. The syntax of these languages is less intuitive to those trained for object-oriented programming and simple XML-based languages. This deficiency makes the job of subject matter experts and application developers difficult, and often

affects negatively to the adoption of the semantic technology in the industry. An effective ontology application development environment should bridge this gap between the semantic markup languages and the object-oriented programmers by providing a tight and seamless integration.

Another consideration for the industry adoption of the semantic technology is the interoperability of the semantic markup languages with the well-established and widely-accepted industry standard modeling languages such as Entity-Relation (ER) modeling, XML Schema, and Unified Modeling Language (UML). The fact is that enterprises developed models in these languages for the past few decades and invested significantly to build systems around them. Despite all the advantages the semantic technology brings in, it is highly unlikely that the enterprises abandon the legacy systems and develop new systems around the semantic technology only. Rather, the users of the semantic technology in the industry would be interested in the interoperability of the modeling languages, and the reuse of the existing models and data with the semantic technology.

To address these practical requirements of the industry, we took an approach based on the Model Driven Architecture (MDA), which enables developers and users to design, build, integrate and manage applications throughout their lifecycle, while separating technology and business concerns [26]. The Object Management Group's MDA specification provides means to organize and manage enterprise architectures supported by automated tools and services for both defining the models and facilitating transformations between different model types. It also provides an open, vendor-neutral approach against the challenge of interoperability. It facilitates efficient use of models in the software development process and reuse of best practices when creating families of systems.

For implementation, we utilized the Eclipse Modeling Framework (EMF), which is IBM's open source MDA infrastructure for integration of modeling tools [23]. A model specification described in various modeling languages including XML Metadata Interchange (XMI) language, XML Schema, and annotated Java source can be imported into EMF. Then EMF produces a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. In its current implementation, EMF does not provide formal semantics definitions, inference and the related model specifications. We are adding this capability to EMF for the comprehensive ontology application development environment and the dynamic application integration.

For adding the semantic model transformation capability to EMF, we utilized the OMG's specification of Ontology Definition Metamodel (ODM) [3], which provides metamodels of W3C's RDF and OWL in UML. By using EMF and ODM, we generated a foundational memory model, i.e., Java classes, for the constructs of RDF and OWL. This foundational memory model is referred to as EODM (Eclipse Ontology Definition Metamodel). By adding several necessary helper classes and methods to EODM, we can use it to create, edit, and navigate any models in RDF and OWL.

We also added an RDF/OWL parser to EODM, which can load RDF/OWL files into EODM and generate RDF/OWL files from EODM, i.e., serialize EODM models to standard XML RDF/OWL files. The parser utilizes an XMI adaptor which enables the transformation between the RDF/OWL models and EMF Core (Ecore) models [23]. The transformation is made possible by defining a mapping between RDF/OWL and the Ecore metamodel. The transformation opens a way to interoperability between RDF/OWL models and other EMF supported models, which currently include ones defined in XML Schema, UML and annotated Java classes. The support of other models such as Entity Relationship models in EMF will be provided in the near future. By leveraging the

RDF/OWL parser and the bi-directional transformation between the RDF/OWL models and the Ecore models, ontology application developers can develop ontologies using their favorite model building tools, import them into EMF, transform their models into OWL ontologies, enrich them with semantics, leverage their inference capability, and utilize the comprehensive development facility of Eclipse and EMF.

To be more specific, the EODM Ecore model is the core model that represents ontologies in memory. It is the intermediate model for imported and transformed legacy models, as well as the generated ontology, Java code, Java editor and Java edit. The development environment allows its users to manipulate EODM Ecore models, enrich it with semantic specification, and generate Java code. A default set of mappings between metamodels of legacy models and OWL are developed in EMF. Eclipse plug-in developers can extend the mappings to handle other types of legacy models, or other elements in legacy models specifying semantics. In the generated Java code, a small foot-print inference engine is shipped with the code and can be invoked by applications. The generated Java editor and Java edit provide ready-to-use visual tools to populate or manipulated instances of OWL models. The visual tools are actually copies of the standard methods of supporting application development in EMF.

9. Related Work

One of the notable studies in ontology management is the Open Knowledge Base Connectivity (OKBC) [4], which shares somewhat similar objectives with JOBC. It is a protocol for accessing knowledge bases stored in Knowledge Representation (KR) Systems. 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 KR systems, so that it provides applications with independence from the idiosyncrasies of specific KR system and enables the development of generic tools.

Other work on ontology management includes KAON [18], SymOntoX [16], pOWL [27], and Jena [25]. KAON is a tool suite for ontology management, providing an editor, a Web interface, an API and an inference engine based on a deductive database approach. SymOntoX is a Web-based ontology editing environment which supports collaborative and distributed ontology authoring activities. Similarly, pOWL also provides a Web-based ontology editing environment. Jena 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 OWL 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.

Additionally, there is active work for ontology integration, which enables to detect and resolve ontological differences and mismatches among existing models and ontologies. Systems such as PROMPT [17] and Chimera [15] belong to this category. A more comprehensive survey on ontology tools can be found in [6]. This survey focuses on the ontology editors and reports 94 existing tools. Additionally, the study reports on management functions of the existing systems such as reasoning and problem solving facilities, standard ontology language support, version control, visual navigation support, ontology alignment, natural language support, collaborative development support, etc.

There are many activities going on in the areas of ontology query. In addition to OWL-QL described in Section 4, there are a few other ontology query languages of interest. RDQL (RDF Query Language) is

a typed, declarative query language for querying RDF description bases. TRIPLE is an ontology query, inference, and transformation language for the semantic Web [19]. It allows the semantics of languages on top of RDF to be defined with rules.

The database-centric architecture for storing and managing ontological data presented in Section 6 is related to deductive databases [9, 21, 22]. The concept of the deductive database was implemented in several systems including SABRE [9], CORAL Deductive Database [21], and Datalog [22]. They often provided a logic-based, declarative query language for the relational model, and imperative constructs such as update, insert and delete rules. In addition, CORAL provided C++ API to combine declarative and imperative programming.

One aspect that separates our approach from the deductive databases is that it teaches rules are written in the standard SQL language as SQL triggers, while the prior art (CORAL, SABRE, Datalog) teaches that rules are written in a logic-based, declarative query language, which is not standard SQL. In addition, unlike prior art, this approach takes advantage of the mature standard SQL technology for managing rules (for update, assert and retract) instead some other means, which is not standard SQL.

Another database-centric approach related to ontology support was discussed in [5]. This approach expanded SQL by adding new operations for supporting ontology-based semantic matching in SQL queries. It allows users to write a SQL query by using the new ontology operators that returns not only syntactically matching rows of data, but also rows of data semantically related as defined in the ontology also specified in the query. It stores ontologies in system-defined tables instead of user tables, and so, they are invisible to the users. Also, inference for queries is hidden in the implementation of the ontology operators. Therefore, it is invisible to the users. The approach additionally discussed a special indexing scheme for speeding up the ontology-based semantic matching queries. This work focused on extending SQL to support ontology-based semantic matching. They placed the actual implementation of the inference and ontology storage mechanisms in a lower level of the system. In contrast, the database-centric approach presented in this paper focuses on the implementation of the inference and ontology storage mechanisms by using plain SQL constructs (triggers) and user tables. The user queries ontologies through OWL-QL which SnoBase supports and the OWL-QL queries are automatically translated into SQL queries.

10. 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 presented a 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 the SnoBase ontology management system, which is under

development at IBM T. J. Watson Research Center. The programming interface of the SnoBase system provides a Java API, Java Ontology Base Connector, which is the ontological equivalent of the Java Data Base Connector. Similarly, this system supports a variant of OWL Query Language (OWL-QL) as our ontological equivalent of SQL (Structured Query Language), and will support a few other ontology query languages in the future.

As ontologies are becoming central to an increasing number of enterprise applications such as business process and information integration, and information search and navigation which require scalability and performance, efficient storage and manipulation of large scale ontological data is going to be essential. In this paper, we introduced an architecture that leverages relational database systems for storing ontologies. We described a direct representation of ontologies in a relational database using a vertical table for storing ontology triples and triggers for automatic maintenance of inferred facts. This solution is primarily concerned with the storage of classes and their relations and less with instance data. In order to accommodate large volumes of data, we are working on an architecture that leaves the data in place, and provides a virtualization layer that effectively links the data triples to ontology classes. The database-centric approach to ontology management support is still in its infancy. For this approach to scale, a number of technical challenges need to be addressed. Some directions for further investigation include:

- Improving trigger evaluation by exploiting the fact that many inference rule-specific triggers share considerable portions of their conditions.
- Exploring query rewriting possibilities by analyzing the characteristics of vertical virtual views used in the database-centric inference scheme.
- A specialized metadata, ontology indexes to guarantee short response times when supporting an increasing number of enterprise applications with large volumes of instance data.
- Extensions to SQL for ontological queries over classes, relations and instances. For example, it might be useful to allow path patterns over the labeled graph of classes and relations.
- Exploring hybrid approaches to storage ranging from fully materialized vertical tables to fully virtual views over existing tables. Cost measures can take into account query work-load and physical distribution of data.
- Reasoning about the cardinalities of the relations between classes and tables. For example, one table corresponds to one and only class, one table stores instances of many classes, a single class can span multiple tables, and finally many-to-many class-table relations.
- Lazy vs. Eager computation of derived facts: some inference rules involve transitive closure and are harder to evaluate at query time; for others, simpler rules (non-recursive) can be added to a user query automatically and evaluated at query time.

Finally, we presented an ontology application development environment which allows model transformation. We took an approach based on the Model Driven Architecture, which supports automated tools and services for both defining the models and facilitating transformations between different model types. Our implementation utilized the Eclipse Modeling Framework (EMF) which is IBM's open source MDA infrastructure. By leveraging the bi-directional transformation between the RDF/OWL models and the EMF models, our system enables ontology application developers to develop ontologies using their favorite model building tools, import them into EMF, transform their models into OWL ontologies, enrich them with semantics, leverage their inference capability, and utilize the comprehensive development facility of Eclipse and EMF.

Acknowledgements

The author thanks Guo Tong Xie, Yue Pan, Li Ma, Yang Yang, ZhaoMing Qiu and Rama Akkiraju for their constructive discussion and support for this work.

References

[1] R. Akkiraju, K. Verma, R. Goodwin, P. Doshi and J. Lee, "Executing Abstract Web Process Flows," The 14th International Conference on Automated Planning and Scheduling (ICAPS 2004), Whistler, British Columbia, Canada, June 3-7 2004.

[2] L. Brownston, R. Farrell, and E. Kant, Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming, The Addison-Wesley series in Artificial Intelligence, 1985.

[3] Daniel T. Chang and Elisa Kendall, "Metamodels for RDF Schema and OWL," http://www.sandsoft.com/edoc2004/ChangRDFS&OWLMDSW.pdf.

[4] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice, "OKBC: A Programmatic Foundation for Knowledge Base Interoperability;" AAAI-98.

[5] S. Das, E. I. Chong, G. Eadon, J. Srinivasan, "Supporting Ontology-based Semantic Matching in RDBMS," The 30th International Conference on Very Large Data Bases, Toronto, Canada, 29 August - 3 September 2004.

[6] M. Denny, "Ontology Tools Survey, Revisited," http://www.xml.com/pub/a/2004/07/14/onto.html, July 14, 2004.

[7] R. Fikes, P. Hayes and I. Horrocks, "OWL-QL - A Language for Deductive Query Answering on the Semantic Web," Knowledge Systems Laboratory, Stanford University, Stanford, CA, 2003.

[8] C. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence, 19, pp 17-37, 1982.

[9] G. Gardarin, and F. Pasquer, "Design and Implementation of SABRE : A Deductive and Parallel Database Machine" in Database Machines, NATO, Pergamon Press, 1985.

[10] M. Grand, Patterns in Java, Volume 1, A Catalog of Reusable Design Patterns Illustrated with UML, John Wiley & Sons, 1998.

[11] I. Horrocks, University Ontology, http://protege.stanford.edu/plugins/owl/owl-library/ka.owl.

[12] V. Krebs, An Introduction to Social Network Analysis, http://www.orgnet.com/sna.html.

[13] J. Lee and R. Goodwin, "The Semantic Webscape: a View of the Semantic Web," The International World Wide Web Conference, Chiba, Japan, May 10-14, 2005.

[14] A. Magkanaraki, G. Karvounarakis, T. T. Anh, V. Christophides, D. Plexousakis, "Ontology Storage and Querying," ICS-FORTH Technical Report, No 308, April 2002.

[15] D. L. McGuiness, "Conceptual Modeling for Distributed Ontology Environments," In Proceedings of the 8th International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues (ICCS 2000), Darmstadt, Germany, August 14-18, 2000.

[16] M. Missikoff, and F. Taglino, "SymOntoX: A Web-Ontology Tool for eBusiness Domains ," The 4th International Conference on Web Information Systems Engineering (WISE'03), Rome, Italy, December 10 - 12, 2003.

[17] N. F. Noy, and M. A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," In 17th National Conference on Artificial Intelligence (AAAI-2000).

[18] D. Oberle, R. Volz, B. Motik, and S. Staab, "An Extensible Ontology Software Environment," In Steffen Staab and Rudi Studer, Handbook on Ontologies, chapter III, pp. 311-333. Springer, 2004.
[19] M. Sintek, and S. Decker, "TRIPLE – An RDF Query, Inference, and Transformation Language for the Semantic Web," International Semantic Web Conference, 2001.
[20] J. Zhu, Z. Tian, T. Li, W. Sun, S. Ye, W. Ding, C. C. Wang, G. Wu, L. Weng, S. Huang, B. Liu, and D. Chou, "Model-Driven Business Process Integration and Management: a Case Study with the Bank SinoPac Regional Service Platform," http://www.research.ibm.com/journal/rd/485/zhu.html.
[21] Coral Database Project, http://www.cs.wisc.edu/coral/.
[22] Deductive Databases -- Datalog Approach, http://goanna.cs.rmit.edu.au/~zahirt/Teaching/subj-datalog.html.
[23] EMF: Eclipse Modeling Franework, http://www.eclipse.org/emf/.
[24] J2EE JDBC Technology, http://java.sun.com/products/jdbc/.
[25] Jena: A Semantic Web Framework, http://www.hpl.hp.com/semweb/jena.htm.
[26] OMG Model-Driven Architecture, http://www.omg.org/mda/.
[27] pOWL: Semantic Web Development Platform, http://powl.sourceforge.net/.
[28] Primer: Getting into RDF & Semantic Web using N3, http://www.w3.org/2000/10/swap/Primer.html.
[29] Protege Ontology Editor, http://protege.stanford.edu/.
[30] RDF Data Query Language (RDQL), "RDQL - A Query Language for RDF," http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.
[31] Resource Description Framework (RDF), http://www.w3.org/RDF/.
[32] Resource Description Framework (RDF): Projects and Applications, http://w3c.org/RDF/#projects.
[33] Semantic Web, http://www.w3.org/2001/sw/.
[34] SnoBase: IBM Ontology Management System, http://alphaworks.ibm.com/tech/snobase.
[35] Web Ontology Language (OWL), http://www.w3.org/2004/OWL/.
[36] Web Ontology Language (OWL): Tools, Projects and Applications, http://w3c.org/2004/OWL/#projects.