# SOR: A Practical System for Ontology Storage, Reasoning and Search

Jing Lu [1,2], Li Ma [1], Lei Zhang [1], Jean-Sébastien Brunner [1], Chen Wang [1], Yue Pan [1], Yong Yu[2]
robert_lu@sjtu.edu.cn, malli@cn.ibm.com, lzhangl@cn.ibm.com
brunner@cn.ibm.com, chwang@cn.ibm.com, panyue@cn.ibm.com, yyu@cs.sjtu.edu.cn

[1] IBM China Research Laboratory
ZhongGuanCun Software Park #19
Dong Beiwang road, ShangDi, Beijing 100094, China

[2] Shanghai JiaoTong University
No.1954, Hua Shan Road
Shanghai 200030, China

## ABSTRACT

Ontology, an explicit specification of shared conceptualization, has been increasingly used to define formal data semantics and improve data reusability and interoperability in enterprise information systems. In this paper, we present and demonstrate SOR (Scalable Ontology Repository), a practical system for ontology storage, reasoning, and search. SOR uses Relational DBMS to store ontologies, performs inference over them, and supports SPARQL language for query. Furthermore, a faceted search with relationship navigation is designed and implemented for ontology search. This demonstration shows how to efficiently solve three key problems in practical ontology management in RDBMS, namely storage, reasoning, and search. Moreover, we show how the SOR system is used for semantic master data management.

## 1. INTRODUCTION

The Semantic Web provides a common framework that allows data to be shared and reused across applications, enterprises, and community boundaries [1]. For this purpose, the World Wide Web Consortium developed several recommendations: the Resource Description Framework (RDF), the RDF Schema (RDFS), the Web Ontology Language (OWL), and the SPARQL for RDF query (a candidate recommendation). RDF is a data model for information representation and exchange on the Web. RDFS and OWL are used to publish and share ontologies which are explicit and common descriptions of domain knowledge and provide support for efficient knowledge management. Universal Resources Identifiers (URIs) and ontologies are central to Semantic Web. The former is to uniquely identify resources and the latter is to make the meanings of the data explicit by linking the data to sets of domain concepts and properties and making inference on them. Description Logics (DL) is the logic foundation of the OWL and a DL knowledgebase includes a Terminology Box (TBox) and an Assertion Box (ABox). The TBox introduces the terminology, i.e., the vocabulary of an application domain, while the ABox contains assertions about named individuals (i.e., instances) in terms of this vocabulary. Correspondingly, DL reasoning includes TBox reasoning (i.e., reasoning with concepts) and ABox reasoning (i.e., reasoning with individuals). Ontologies define concepts (classes) and relations (properties) in a domain and sometimes are called the TBox in knowledge representation. From the perspective of data management, ontologies could be considered as a data model (similar to relational schema and XML schema) and the data can be regarded as the instances of ontologies (corresponding to ABox of a DL knowledgebase). There are three key enabling technologies for the use of ontologies in practice.

- How to store ontologies and their instances of large scale (even hundreds of millions of triples) in relational databases. How to support scalable ontology storage by leveraging well-developed optimization features of RDBMSes.
- How to effectively support reasoning in relational databases.
- Besides structured SPARQL queries, how to provide a convenient and friendly search diagram for ontologies.

In this demonstration, we present efficient solutions to the above three problems and show the use of the SOR in an IBM product prototype on semantic master data management.


*VLDB '07*, September 23-28, 2007, Vienna, Austria.


## 2. ONTOLOGY STORAGE AND REASONING IN DATABASES

In the past several years, we have seen the development of ontology repositories for the use in semantic web applications. In general, ontology repositories can be divided into two major categories, i.e., native stores [4] and database based stores [3,5,6,7]. Native stores are directly built on the file system, whereas database based repositories use relational or object relational databases as the backend store. Compared with database based stores, native stores greatly reduce the load and update time. However, database systems provide many query optimization features, thereby contributing positively to query response time. It is reported in [9] that a simple exchange of the order of triples in a query may make the response time of native stores 10 times (or even more) slower. Furthermore, native stores need to re-implement functionalities of a relational database such as transactions processing, query optimization, access control, logging and recovery. One potential advantage of database based stores is that they allow users and applications to access both ontologies and other enterprise data in a more seamless way at the lower level, namely database level. For instance, The Oracle RDF store [6] translates an RDF query into a SQL query which can be embedded into another SQL query retrieving non-RDF data. In this way, query performance could be improved by efficiently joining RDF data and other data using well-optimized database query engines. Efforts are needed to better integrate native ontology stores with RDBMSes. So, here we focus on ontology storage and reasoning in databases.

Generally, an ontology store, such as Jena, Sesame and the Oracle RDF store, mainly uses a relational table of three columns (Subject, Property, Object) to store all triples (hereinafter, we call such a table as the triple table.), in addition to symbol tables for encoding URIs and literals with internal unique IDs. The Oracle RDF store [6] supports so-called rulebases and rule indexes. A rulebase is an object that contains rules which can be applied to draw inferences from RDF data. Two built-in rulebases are provided, namely RDFS and RDF (a subset of RDFS). A rule index is an object containing pre-calculated triples that can be inferred from applying a specified set of rulebases to RDF data. Materializing inferred results by rule indexes would definitely speed up retrieval. Jena2 provides by default several rule sets with different inference capability. These rule sets could be applied on the data stored in the triple table and implemented in memory by forward chaining, backward chaining or a combination of them. Several ontology stores adopt binary tables for storage, instead of the triple table. These stores, such as DLDB-OWL [3] and Sesame on PostgreSQL [5], create a table for each class (resp. each property) in an ontology. A class table stores all instances belonging to the same class and a property table stores all triples which have the same property. An obvious drawback of these stores is the alteration of the schema (e.g., deleting or creating tables) when ontologies change. Also, it is not suitable for very huge ontologies having tens of thousands of classes, such as SnoMed ontology. Too many tables will increase serious overhead to databases.

In summary, there are several features among the above well-known ontology stores on top of RDBMS.

- The triple table is very flexible and thus widely used. Often, multiple self-joins on this table are needed for query answering.
- Rule inference is also often adopted for reasoning and inferred results can be materialized in databases, such as the Oracle RDF store (the rulebase and rule indexes) and Sesame.
- A main-memory Description Logics (DL) reasoner or rule reasoner could be used for reasoning as well, such as Jena.

It is desirable to design schema to effectively support rule reasoning in databases and leverage efficient indexes on the triple table for performance improvement.

## 2.1 Schemes for the Scalability of the SOR

SOR is a significant extension of our previous work on OWL ontology repository [2] by performance improvement and the support of more practical features for enterprise applications (such as concurrency and guided navigation). Figure 1 shows its architecture. The import module consists of an OWL parser and two translators. The parser parses OWL documents into an in-memory EODM model (EMF ontology definition metamodel) [10], and then the DB translator populates all ABox assertions into the backend database. The function of the TBox translator is twofold, one is to populate all asserted TBox axioms into a DL reasoner and the other is to obtain inferred results from the DL reasoner and insert them into the database. Several DL reasoners, such as pellet, are mature enough for TBox reasoning in memory. But scalable ABox reasoning is still challenging and thus the focus of the SOR. Firstly, SOR uses a DL reasoner to infer complete subsumption relationship among classes. Then, the rule engine conducts ABox reasoning based on a set of rules translated from OWL-lite. Currently, the inference rules are implemented using SQL statements. The storage module is intended to store both original triples, as well as inferred assertions by the DL reasoner and the rule inference engine. But, there is a way to distinguish original assertions from inferred assertions via a specific flag. A SPARQL query is translated into a single SQL statement which is directly evaluated by the database. Obviously, this enables the straightforward integration of a SPARQL query with other SQL queries. Furthermore, triples in SOR can be exported and indexed by a full text search engine so that users can navigate ontologies via relationship among instances.

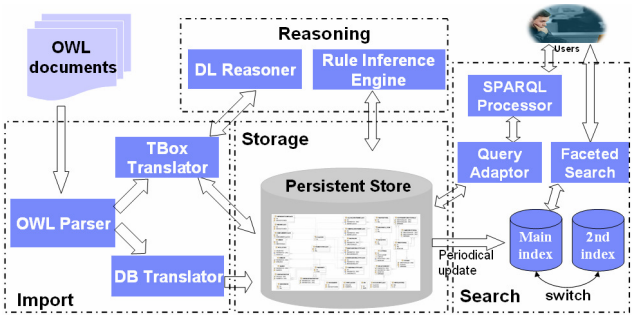


Figure 1. The Component Diagram of the SOR

As introduced in the above, the Oracle RDF store, Jena2 and Sesame mainly use a triple table to store ontology in relational databases. In particular, they persist OWL ontologies as a set of RDF triples and do not consider specific processing for complex class descriptions generated by OWL constructors (intersection, cardinality, someValueFrom restrictions, etc). The highlight of SOR's schema is that all predicates in the ABox inference rules have corresponding tables in the database. Therefore, these rules can be easily translated into sequences of relational algebra operations. For example, Rule *Type(x,C) :- Rel(x,R, y)* $\wedge$ *Type(y,D)* $\wedge$ *SomeValuesFrom(C,R,D)* has four predicates in the head and body, resulting in three tables: *RelationshipInd*, *TypeOf* and *SomeValuesFrom*. It is straightforward to use SQL *select* and *join* operations among these three tables to execute this rule. Leveraging well-optimized database engines for rule inference is expected to significantly improve reasoning efficiency. That is, each predicate of the rules corresponds to a sort of triples which are stored in a separate table, instead of a big triple table. This is more efficient since some self-joins on a big triple table are changed to joins among small-sized tables.

We categorize tables of the database schema of SOR into 4 types: atomic tables, TBox axiom tables, ABox fact tables and class constructor tables, as shown in Figure 2. The atomic tables include: Ontology, PrimitiveClass, Property, Datatype, Individual, Literal and Resource. These tables encode the URI with an integer (the ID column), which reduces the overhead caused by long URIs to a minimum. The hashcode column is used to speed up search on URIs and the ontologyID column denotes which ontology a URI comes from. The Property table stores characteristics (symmetric, transitive, etc.) of properties as well. To leverage built-in value comparison operations of databases, boolean, datetime and numeric literals are separately represented using the corresponding data types provided by databases. There are three important kinds of ABox assertions involved in reasoning: TypeOf triples, object property triples and datatype property triples. They are stored in three different tables, namely tables TypeOf, RelationshipInd and RelationshipLit. A view named *Relationship* is constructed as an entry point to object property triples and datatype property triples. Triples irrelevant to reasoning, such as those with RDFS:comment as the property, are stored in Table Utility. Tables SubClassOf, SubPropertyOf, Domain, Range, DisjointClass, InversePropertyOf

are used to keep TBox axioms. The class constructor tables are used to store class expressions. SOR decomposes a complex class description into instantiations of OWL class constructors, assigns a new ID to each instantiation and stores it in the corresponding class table. Taking an axiom Mother $\equiv$ Woman $\sqcap$ $\exists$ hasChild.Person as an example, we first define S1 for $\exists$ hasChild.Person in Table SomeValuesFrom. Then I1, standing for the intersection of Woman and S1, will be defined in Table IntersectionClass. Finally, {Mother $\sqsubseteq$ I1, I1 $\sqsubseteq$ Mother} will be added to the SubClassOf table. Such a design is motivated by making the semantics of complex class description explicit. In this way, all class nodes in the OWL subsumption tree are materialized in database tables, and rule inference can thus be easier to implement and faster to execute via SQL statements. Also, a view named *Classes* is defined to provide a view of both named and anonymous classes in OWL ontologies.

Currently, most database systems support primary clustering indexes. In this design, an index containing one or more keyparts could be identified as the basis for data clustering. All records are organized on the basis of their attribute values for these index keyparts by which the data is ordered on the disk. More precisely, two records are placed physically close to each other if the attributes defining the clustering index keyparts have similar values. Clustering indexes could be faster than normal indexes since they usually store the actual records within the index structure and the access on the ordered data needs less IO costs. In practice, it is not suitable to create an index on a column with few distinct values because the index does not narrow the search too much. But, a clustering index on such a column is a good choice because similar values are grouped together on the data pages. Considering that real ontologies have limited number of properties, the property column of the triple table, such as the *RelationshipInd* table of SOR, can be a good candidate for clustering. So, it is valuable to use clustering indexes on triple tables for performance purpose.

Similar to unclustered indexes, the clustering index typically contains one entry for each record as well. More recently, Multi-Dimensional Clustering (MDC) [12] is developed to support block indexes which is more efficient than normal clustering indexes. Unlike the primary clustering index, an MDC index (also called MDC table) can include multiple clustering dimensions. Moreover, the MDC supports a new physical layout which mimics a multi-dimensional cube by using a physical region for each unique combination of dimension attribute values. A physical block contains only records which have the same unique values for dimension attributes and could be addressed by block indexes, a higher granularity indexing scheme. Block indexes identify multiple records using one entry and are thus quite compact and efficient. Queries using block indexes could benefit from faster block index scan, optimized prefetching of blocks, as well as lower path length overheads while processing the records. Our evaluation results [11] showed that the MDC indexes could dramatically improve query performance (20 times faster and even more) and the set of indexes P*, (P,O), (S,P,O) on the triple table gives the best result for most queries of the UOBM benchmark [9] over the SOR, where P* means an MDC index, other two represent composite unclustered indexes. Additionally, the MDC index could be built on the table defining *typeOf* information, grouping the records by classes.

In summary, we proposed following schemes to make the SOR be a scalable ontology repository.

- The schema is designed based on inference rules, where each predicate in the rule corresponds to a table in the database. This is greatly different from the Oracle RDF store and Jena2 which use a big triple table to store all triples. Rule inference can be effectively implemented by simple SQL *select* and *join* operations in well-optimized database engines. Also, the SOR can process OWL ontology better than general RDF stores since it considers the support of complex OWL expressions in databases.
- The primary clustering and the MDC indexes are analyzed and proposed to use on the triple table for performance improvement.
- Also, typed literals are categorized into four major types aligned with the datatypes supported by popular RDBMSes and separately stored to support efficient value comparison (thus improve performance for queries involving literal comparison).

## 2.2 Faceted Search with Relationship Query

One of the important functions of an ontology repository is to provide an efficient and effective way for end users to intuitively "view" ontologies. Faceted search is a promising way to search and explore categorized data [8]. From an ontology perspective, faceted search combines conjunctive class query (for example, "Find all products which belong to both **PDAs** and **mobile phones**", where "**PDAs**" and "**mobile phones**" are two classes in TBox.) with full text search in an intuitive way. However, it does not support query involving relations in its current form. Taking the query "Find **computer monitors** *sold in* **stores** that have *a partnership with* ***IBM***" as an example, "**computer monitors"** and **"stores"** are two classes**, "*IBM*"** is an instance in the data, and "*sold in*" and **"***partnership with***"** are two relations in ontology. Obviously, this kind of relationship queries is desirable for end users. In SOR, we define and implement faceted search with relationship navigation. Each instance is considered as a virtual document as shown in Figure 3. Particularly, we index relations among instances by leveraging the index structure of Lucene and develop an efficient algorithm to estimate the count of instances at each navigation step.

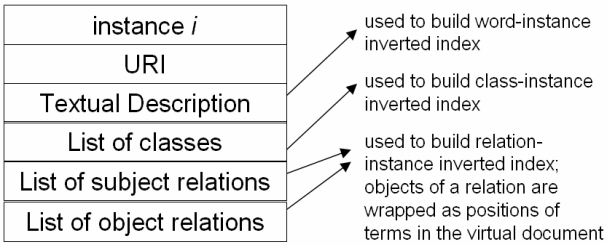


Figure 3. The Index Structure of An Instance

Due to space limitation, figure 4 briefly shows the proposed faceted search paradigm with relationship navigation. The sub-figure above the black bold line is just classic faceted search, whereas the sub-figure in the bottom clearly show all relations associated with instances which satisfy search conditions in the last step. By this kind of relation browsing, users can navigate from one instance to another one. The proposed search diagram allows users to navigate ontologies by walking up relationships (links) among instances.

# 3. DEMONSTRATION

Following demonstrations are conducted to show the scalability and the friendly search diagram of the SOR and its real use cases.

- **The effectiveness of the schemas and the scalability of SOR.** We will demonstrate the effectiveness of the schemas and the scalability of SOR on a real customer data set which include 4,200,000 product items and more than 120M triples. The

expressivity of the ontology is OWL-Lite. We demonstrate the scalability of SOR by observing its performance changing on the UOBM data sets [9] with different sizes. Also, we will show the translated SQL queries from SPARQL queries and explain the translation method and process.

●**Faceted Search combined with relationship navigation.** We will demonstrate the new user interaction paradigm that supports relationship navigation and full text search in ontology hierarchical structure. The user interaction extends the classic faceted search paradigm with relationship query. We will demonstrate how this user interaction for querying, searching and browsing is applied on ontologies to help user navigate large amount of the data easily.

●**Real use scenarios of SOR in enterprise applications.** SOR has been used to build an IBM product prototype on semantic master data management [13]. We would like to demonstrate how real ontologies are exploited to manage master data in terms of modeling, storage and search. Ontology's values for master data management include computable relations representation among master data, formal and expressive definition for categories, dynamic queries on master data based on ontology reasoning, and guided navigation by faceted search with relationship queries. We will demonstrate how SOR is used to realize these attractive features for master data management in detail. We also highlight requirements from real applications for ontology systems and the corresponding solutions in SOR.

## 4. REFERENCES


[1] W3C Semantic Web Activity, http://www.w3.org/2001/sw/.

[2] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: An OWL ontology storage and inference system. In Proc. of Asia Semantic Web Conference, pp.429-443, 2006.

[3] Pan, Z., and Heflin, J., DLDB: Extending relational databases to support semantic web queries. In Proceddings of Workshop on Practical and Scaleable Semantic Web Systems, 2003.

[4] Kiryakov, A., Ognyanov, D., and Manov, D, OWLIM - a pragmatic semantic repository for OWL. In Proceedings of the 2005 International Workshop on Scalable Semantic Web Knowledge Base Systems, 2005.

[5] Broekstra, J., Kampman, A., van Harmelen, F., Sesame: A generic architecture for storing and querying rdf and rdf schema. ISWC 2002, pp.54–68, 2002.

[6] Murray C., Alexander N., Das S., Eadon G., Ravada S., Oracle Spatial Resource Description Framework (RDF), 10g Release 2 (10.2), 2005.

[7] Wilkinson, K., Sayers, C., Kuno, H. A., and Reynolds, D., Efficient RDF storage and retrieval in Jena2. VLDB Workshop on Semantic Web and Databases, pages 131-150, 2003.

[8] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In Proc. of the SIGCHI conference on Human factors in computing systems, pages 401–408, 2003.

[9] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete owl ontology benchmark. In Proc. of European Semantic Web Conference, pp.125–139, 2006.

[10] IODT, IBM's Integrate Ontology Development Toolkit, http://www.alphaworks.ibm.com/tech/semanticstk, 2005

[11] Brunner, J., Ma, L., Wang, C., Zhang, L., Wolfson, D. C.., Explorations in the Use of Semantic Web Technologies for Product Information Management. WWW 2007. pp.747-756.

[12] Bhattacharjee, B., Padmanabhan, S., and Malkemus, T., Efficient Query Processing for Multi-Dimensionally Clustered Tables in DB2, In Proceedings of the 29th Conference on Very Large Data Bases, pages 963–974, 2003.

[13] Morris, H.D. and Vesset, D. Managing Master Data for Business Performance Management: The Issues and Hyperion's Solution, IDC white paper, 2005.


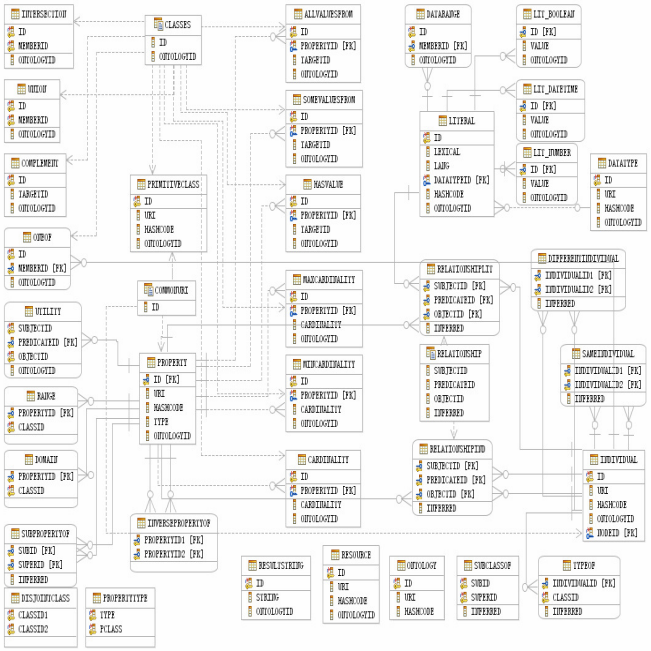

Figure 2. The Schema of the SOR

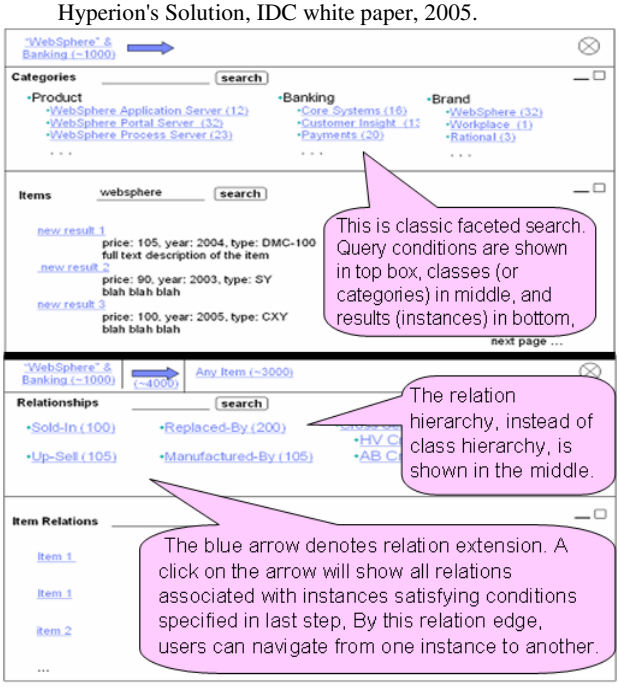


Figure 4. Faceted Search with Relationship Navigation