DOI : 10.3745/JIPS.2009.5.1.011

An Efficient Web Ontology Storage Considering Hierarchical Knowledge for Jena-based Applications

Dongwon Jeong*, Heeyoung Shin, Doo-Kwon Baik** and Young-Sik Jeong*****

Abstract: As well as providing various APIs for the development of inference engines and storage models, Jena is widely used in the development of systems or tools related with Web ontology management. However, Jena still has several problems with regard to the development of real applications, one of the most important being that its query processing performance is unacceptable. This paper proposes a storage model to improve the query processing performance of the original Jena storage. The proposed storage model semantically classifies OWL elements, and stores an ontology in separately classified tables according to the classification. In particular, the hierarchical knowledge is managed, which can make the processing performance of inferable queries enhanced and stores information. It enhances the query processing performance by using hierarchical knowledge. For this paper an experimental evaluation was conducted, the results of which showed that the proposed storage model provides a improved performance compared with Jena.

Keywords: *Ontology, Jena, OWL, Ontology, Storage, Hierarchical Structure*

1. Introduction

The Semantic Web has been recognized as the next-generation Web; as such, a considerable volume of research has been conducted with a view to its realization. In particular, various Web Ontology description languages such as RDF (Resource Description Language), RDF-S (RDF Schema), and OWL (Web Ontology Language) have been developed by W3C (World Wide Web Consortium). The Semantic Web, as a data Web, provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries [1].

If the Semantic Web does become reality, it will be better for consumers because they will be able to find products and services more easily using semantics that provide precisely what they need. To realize the Semantic Web, we need many technologies including, above all, technologies for the description and management of Web ontology. The typical description languages include RDF [2], RDF-S [3], and DAML+OIL [4]. DAML and OIL stand for DARPA Agent Markup Language and Ontology Inference Layer or Ontology Interchange Language, respectively. Currently, OWL (Web Ontology Language) is recognized as the most representative language for the description of Web ontology. An OWL Web ontology consists of descriptions of classes, properties, and instances. OWL has been designed for use in applications where the content of information needs to be processed rather than simply presenting the information to users. It facilitates greater machine interpretability of Web content than that supported by XML (Extensible Markup Language), RDF, and RDF Schema (RDF-S) by providing additional vocabularies along with a formal semantics. OWL is based on earlier languages, OIL and DAML+OIL, and is now a W3C recommendation [5]. OWL is also being used in various applications [6,7,8,9].

Jena is a Semantic Web framework developed by Brian Bride at the HP Semantic Web Laboratory. Jena provides a programming environment and a basic RDF parser for RDF, RDF-S, and OWL, and contains an internal reasoning engine based on rules [10]. Jena shows a relatively simple physical schema structure during the automatic creation of a storage model. However, Jena still has several problems with regard to real applications, and from the perspective of storage its query processing performance is unacceptable. In particular, the Jena storage consumes too much time when processing a large volume of web ontology, because Jena stores information in a single table as a non-normalized single structure [12, 13].

In this paper, a new ontology relational database model is proposed to solve the abovementioned limitation. The proposed model is a plug-in storage model designed to accept all the strengths of the Jena framework. The suggested storage model supports more efficient query processing performance. In addition, the results of the experiment are described to show the advantages of the proposed model.

The remainder of this paper is organized as follows. Section 2 introduces the storage model of the Jena framework; Section 3 shows the proposed storage model; Section 4 presents the evaluation methodology, including data sets from the experiment, the system environment, and

Manuscript received March 2, 2009; accepted March 17, 2009.

Corresponding Author: Dongwon Jeong
* Dept. of Informatics & Statistics, Kunsan National University, Gunsan, Jeollabuk-do, Korea (djeong@kunsan.ac.kr)
** Dept. of Computer Engineering Science & Engineering, Korea University, Seoul, Korea (shintul@software.korea.ac.kr, baikdk@korea.ac.kr)
*** Dept. of Computer Engineering, Wonkwang University, Iksan, Jeollabuk-do, Korea, (ysjeong@ wku.ac.kr)

comparative items; and Section 5 presents the conclusion.

2. Related Work

This section describes the overall architecture of the storage structure of the Jena Semantic Web framework and its relational model, which is automatically generated.

2.1. Jena Architecture

Fig. 1 shows the overall architecture of Jena. Jena consists of seven parts: Reader, Writer, Network API, Query, Inference, RDF API, and Store.

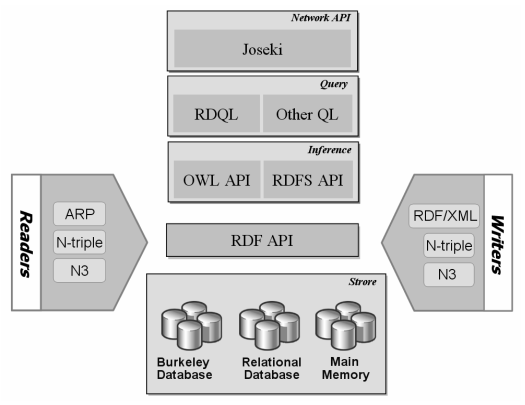


Fig. 1 Jena framework

The part comprising the Reader and Writer has the role of reading and storing RDF in RDF/XML, N3, and N-Triples through the RDF API. The most important part of the Jena framework is the RDF API. The RDF API supports the creation, processing, and querying of RDF graphs, and also supports various storage technologies such as the main memory, relational databases, and Berkeley database. The Inference part is composed of the OWL API and the RDF-S API. Therefore, Jena supports reasoning facilities inherited from both ontology description languages, RDF-S and OWL. The Query part is a set of APIs for the processing of queries in RDQL (RDF Data Query Language) [14] and SPARQL (Simple Protocol and RDF Query Language) [11]. Finally, the Network API enables access (query, correction operation, etc.) to the remote RDF databases.

2.2 Jena Storage Model

Jena provides a Web ontology storage model based on the relational database model. The storage model is automatically created, and Fig. 2 expresses the Jana relational database model, which consists of jena_long_lit, jena_gntn_stmt, jena_long_uri, jena_gntn_reif, jena_prefix, jena_graph, and jena_sys_stmt.

Once an OWL ontology is given, Jena extracts and stores statements (triples) in a table, jena_gntn_stmt. This table is basically composed of three columns (Subject, Property, and Object). If the length of the literals or the URIs is more than 256 bytes, then they are stored in jena_long_lit and jena_long_uri respectively.

In brief, the Jena storage model stores most data in a table (jena_gntn_stmt), which causes many problems. These problems are summarized as the following three key issues:

- Data redundancy
- No consideration of hierarchy between classes or properties
- Low query processing performance

An OWL Web ontology basically consists of classes, properties, relations between classes, and instances. However, in the case of the Jena storage model, most data are managed in a single table, thus causing data redundancy. The redundancy problem naturally affects query processing performance. It means that the Jena storage model causes a reduction in query processing performance. The reduction increases exponentially in the case of join operation queries than simple queries.

For example, let’s assume that the number of all tuples in the table jena_gntn_stmt is *N*. The numbers of classes and instances are *M* and *K* (*N*>*M* and *N*>*K*). When a query to find a specific class is given, an *N*-time comparison is required to retrieve the class. If we manage the classes in a separate table (class_def), the number of tuples of the class_def table will be *M* and will hence require an *M*-time comparison. As for the join operation processing, a query to find a class and its corresponding instances is assumed. Jena requires an *N***N*-time comparison, while the latter requires *M***K* times.

Finally, the Jena storage model has no consideration of hierarchical structures between classes or properties. It also lowers the query processing performance.

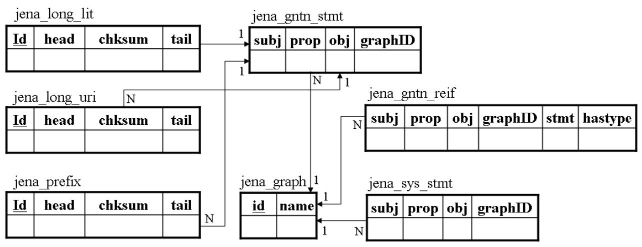


Fig. 2 Jena storage model

3. Proposed Model

This section describes our proposed model, named JeSPi (Jena Storage Plug-in for Enhanced Query Processing).

3.1 Overall Structure

Fig. 3 shows the JeSPi framework. JeSPi has two main components for managing ontology. The first component, Adapter, stores a given OWL ontology in the JeSPi storage model using Jena API; the second component, Converter, translates the JeSPi storage model into the Jena storage model.

Eventually, JeSPi has the advantage of accommodating the relational information in the OWL document to be newly stored and the OWL document already stored. In other words, JeSPi provides improved storage and leads application developers to use Jena's strengths.

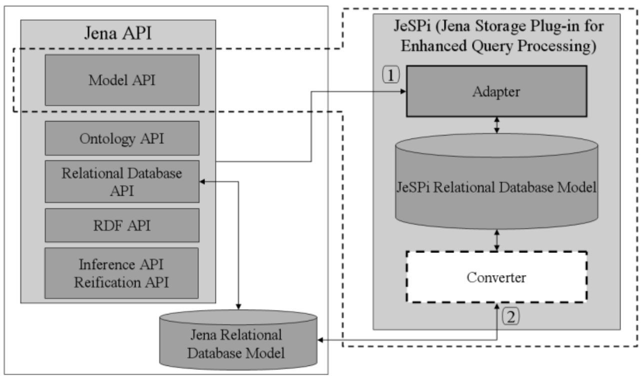
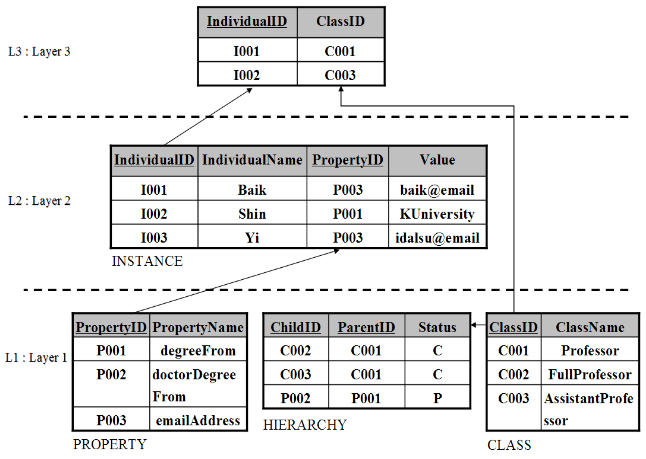

Jena API
Model API
Ontology API
Relational Database API
RDF API
Inference API
Reification API
Jena Relational Database Model
JeSPi (Jena Storage Plug-in for Enhanced Query Processing)
1
Adapter
JeSPi Relational Database Model
Converter
2

Fig. 3 Overall structure of the proposed model

3.2. Proposed Storage Model

This section describes the whole structure of the proposed JeSPi storage model. Fig. 4 shows three layers of the storage model. Layer 1 is composed of three tables, CLASS, PROPERTY, and HIERARCHY. Layer 2 and Layer 3 consist of the table INSTANCE and the table CONCEPT, respectively. And, the JeSPi storage model classifies data into three groups (classes, properties, and instances) and stores each group into a separate table.



L3 : Layer 3

IndividualID	ClassID
I001	C001
I002	C003

L2 : Layer 2

IndividualID	IndividualName	PropertyID	Value
I001	Baik	P003	baik@email
I002	Shin	P001	KUniversity
I003	Yi	P003	idalsu@email

INSTANCE

L1 : Layer 1

PropertyID	PropertyName
P001	degreeFrom
P002	doctorDegree From
P003	emailAddress

PROPERTY

ChildID	ParentID	Status
C002	C001	C
C003	C001	C
P002	P001	P

HIERARCHY

ClassID	ClassName
C001	Professor
C002	FullProfessor
C003	AssistantProfe ssor

CLASS

Fig. 4 Relational model of the proposed storage

In Layer 1, classes and properties are stored in CLASS and PROPERTY, respectively. The table HIERACHY contains the hierarchical knowledge between classes or properties. The field Status in the table is used to identify hierarchical types. C denotes the hierarchical information between classes, and P means the hierarchical relation between properties.

The Jena storage does not manage the hierarchical information as a separate table. Therefore, the operation to find sub-nodes (for example, subclasses of a class or sub-properties of a property) requires many comparisons.

However, JeSPi is able to reduce unnecessary cost by storing only necessary information. Therefore, it enables faster processing than that permitted by Jena.

In addition, OWL has many vocabularies to be managed, such as the hierarchical information. However, this paper deals with subPropertyOf and subClassOf only by focusing on the hierarchical structure, and leaves the others in OWL for future research.

The INSTANCE table of Layer 2 is composed of a table, which is composed of four fields - IndividualID, IndividualName, PropertyID, and Value.

The Layer 3 is composed of a table to map the relationship between the CLASS table and the INSTANCE table. For example, a set of classes (Professor, FullProfessor, and AssistantProfessor) is stored in the CLASS table, and properties such as degreeFrom, doctorDegreeFrom, and emailAddress are stored in the PROPERTY table. The hierarchical relationship between Professor and its sub-classes (FullProfessor and AssistantProfessor) is stored in the HIERARCHICAL table.

C001, a professor's ID, is stored in the CONCEPT table of Layer 3 by referring to an instance (I001) in the INSTANCE table. The proposed storage model can clearly store the relationships between classes and classes, the relationships between properties and properties, and the relationship among classes, properties, and instances. An OWL ontology is stored into each table, whereupon the proposed model shows higher performance than Jena. The performance evaluation is described in Section 4.

OWL Full has a flexible relationship with each element, being able to express the relationship among classes, properties and instances without any sufficient restriction condition. Such a characteristic makes it very difficult to define a storage model, so for this paper this paper handles OWL-DL only.

3.3 Adapter

One of the key components of JeSPi is Adapter, which is composed of three parts. The role of Adapter is to store data in the proposed storage rather than in the Jena storage.

Each part is described as the following three steps.

- Step1: Extracting and classifying the OWL data sets
  This part has the role of extracting and classifying the OWL data by analyzing the meaning of OWL data. OWL data is loaded and then classified according to the meanings. The classification is used to store the OWL data in the proper tables of JeSPi storage.
- Step2: Temporarily storing data in a memory field
  Once OWL data has been extracted from the given OWL documents and classified, the data set is temporarily stored in the memory area.

- Step3: Permanently storing
  The OWL data in the memory is stored into the tables created by JeSPi.

4. Experiment and Evaluation

This section describes the experiment, evaluation and comparison of the Jena storage and the JeSPi storage.

4.1 Experimental Environment

Experiments were conducted under the following physical environment.

- CPU: Pentium 4 (2.81GHz)
- Memory Size: 512MB
- Heap Memory Size: 1024MB
- Hard Disk Size: 100GB
- Operating System: Windows Server 2K OS
- Language: Java
- Java JDK Version: Java JDK 1.6.0
- Database Management System: Oracle 9i

For the experiments, this paper uses a data set created by UBA (University-Bench Artificial Data Generator), an ontology creation tool provided by Lehigh University.

UBA was developed to evaluate the results of the SWAT project [15], and many experiments have used the data sets created using this tool [16,17,18,19]. UBA creates data in LUBM (n, s), where n and s mean the number of universities and the seed value respectively. For example, in the case of creating a data set such as LUBM (1,0), the number of universities is 1. Also, the university is composed of information such as multiple departments, professors, colleges, graduate schools, and so forth.

The reason we use the OWL data set by UBA is that it is possible to conduct experiments by creating a reasonable ontology size. The size of an OWL data set on the web is small, so it is difficult to obtain results that are much more precise. UBA, however, is suitable for evaluating storages' performance with a large volume of ontology.

This paper uses an ontology set, LUBM (1,0) created by UBA. The generated data set basically contains a total of 15 OWL documents. For the experiment, four types of ontology size were used, with each type containing 1, 5, 10 and 15 document files respectively.

Larger ontology sets can be generated and used for the experiment, e.g., LUBM (5,0), LUBM (10,0), or LUBM (20,0). In the experiments conducted for this paper, however, LUBM (1,0) is sufficient to differentiate identify between the performance of the Jena storage and that of the proposed storage.

We herein define five queries for the experiment. The queries are defined considering certain factors, the objects of retrieval and the hierarchical knowledge. In other words, the query patterns are defined according to what the goal, i.e., the query result, of the query is and which hierarchical knowledge is used or not.

- Q-P1: Search all classes
- Q-P2: Search instances of a specific class
- Q-P3: Search all hierarchical classes of a specific class
- Q-P4: Search all hierarchical properties of a specific property
- Q-P5: Search all hierarchical instances of a specific class

We describe each query in SPARQL language [11]. Following that, we describe the characteristics of the query. Let *[RDF-S], [owl],* and *[univ]* denote the following:

[RDF-S] : *http://www.w3.org/2000/01/rdf-schema*
[owl] : *http://www.w3.org/2002/07/owl*
[univ] : *http://www.lehigh.edu/~zhp2/2004/0401/univ- bench.owl*

- Q-P1: Search all classes

```
select ?subj
where  {
  ?subj ?Prop Uv::[owl]#Class
}
```

- Q-P2: Search an email address of AssistantProfessor7

```
select ?Obj
where  {
   ?subj Uv::[univ]# emailAddress ?Obj
   ?subj ?prop
   Uv::http://www.Department0.University0.edu/Assistant
   Professor7
}
```

- Q-P3: Search all professor classes

```
select ?class
where{
   {
     ?class Uv::[RDF-S]#subClassOf ?Obj.
     ?Obj Uv::[RDF-S]#subClassOf Uv::[univ]#Professor
   }
   union (unite?) all
   {
     ?class subClassOf ?Obj.
     ?Obj Uv::[RDF-S]#subClassOf Uv::[univ]#Professor
   }
}
```

- Q-P4: Search all properties included in DegreeFrom

```
select ?
where{
   {
     ?subj [RDF-S]#subPropertyOf ?Obj.
     ?Obj [RDF-S]#subPropertyOf
     Uv::[univ]#degreeFrom
   }
   union all
```

```
    {
      ?subj Uv::[RDF-S]#subPropertyOf
      Uv::[univ]#degreeFrom
    }
  }
```

- Q-P5: Search all professor instances

```
  select ?ins
  where{
    ?ins rdf:type ?subj
    {
      ?subj Uv::[RDF-S]#subClassOf ?Obj.
      ?Obj Uv::[RDF-S]#subClassOf Uv::[univ]
      #Professor
    }
    union all
    {
      ?subj Uv::[RDF-S]#subClassOf
      Uv::[univ]#Professor
    }
  }
```

4.2 Results of the Experiment

This section describes the results of the experiment conducted on the following comparative items: query response time, accuracy, and completeness.

4.2.1 Results of the experiment on query response time

As mentioned above, this paper used four data sets, LUBM (1,0,1), LUBM (1,0,5), LUBM (1,0,10) and LUBM (1,0,15), for the experiment. Table 1 and Table 2 show the entire results of the experiments with four data sets and five queries.

Table 1 Results of the experiment on query response time.

Models / Data sets	Jena				
	Q-P1	Q-P2	Q-P3	Q-P4	Q-P5
LUBM(1,0,1)	506.4	1417.6	515.8	516.0	1118.4
LUBM(1,0,5)	678.2	2096.8	516.0	524.6	1443.8
LUBM(1,0,10)	681.4	3406.4	516.0	596.0	9481.0
LUBM(1,0,15)	825.0	4603.0	516.0	624.2	11796.6
Models / Data sets	**JeSPi**				
	Q-P1	Q-P2	Q-P3	Q-P4	Q-P5
LUBM(1,0,1)	500.0	891.0	500.0	500.0	503.0
LUBM(1,0,5)	500.0	1153.2	500.0	500.0	512.8
LUBM(1,0,10)	500.0	1818.8	500.0	500.0	2971.8
LUBM(1,0,15)	500.0	2034.4	500.0	500.0	3772.6

Fig. 5 shows the results of the experiment in Table 1. Fig. 5(a) and Fig. 5(d) show the results with each query to respectively find a class and a property. In Fig. 5(a) and 5(d), the larger the ontology, the more query response time Jena consumes. On the other hand, JeSPi's response time is uniform. Regardless of the data sets such as LUBM (1,0,1) and LUBM (1,0,15), the number of classes or properties is constant. In other words, as the size of the ontology size, only instance increases in size. Therefore, all the data sets have the same class or property set. As mentioned above, Jena stores all data (classes, properties, and instances) in a table, and thus compares unnecessary data (instances and properties) as well as classes to search for a given class. However, JeSPi manages data in separate tables and accesses classes to do it. As a result, JeSPi requires the same query response time to search for a class or property; whereas Jena's response time increases exponentially according to the size of the table, i.e., instances.

Fig. 5(b) shows the results of the experiment with the query (Q-P2) to find all the instances of a given class. Therefore, this query requires the equi-join operation to get the query result. To find its query result, two operations are required: The first consists in finding information related with the given class, the result of which is a tuple containing information such as class ID, class name, and so on; the other consists in finding corresponding instances to the given class. To do this, an equi-join operation between instances and classes should be accomplished. In a word, the query processing performance is affected by comparison cost of both operations.

In Fig. 5(b), Jena consumes much more time compared to JeSPi. We simply assume that the numbers of classes, properties, and instances are 10, 5, and 100. For the first operation, Jena compares all of the instances and properties including classes, whereas JeSPi only compares classes. Therefore, Jena's comparison time is 115 while JeSPi's is 10. For the second operation, Jena requires 1*115-time comparison and JeSPi's comparison time is 1*100. As a result, Jena and JeSPi consume 115*115 and 10*100 respectively.

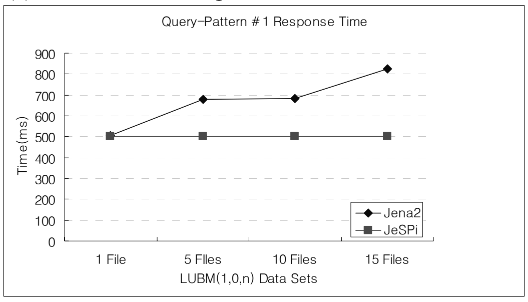
Fig. 5(e) depicts the results of the experiment with the query (Q-P5) to find all instances of the sub-classes of a given class. In this figure, we can understand that JeSPi is more efficient than Jena for the same reason discussed in relation to Fig. 5(b).

There is a difference in the query response time between Jena and JeSPi in Fig. 5(c) because JeSPi stores only the information necessary for the hierarchical relationship between classes or properties. However, the relational database storage model in Jena stores additional information (#type, #first, #rest) together. The number of properties, however, is limitedly equal as data sets increase as mentioned above, so both the Jena model and the JeSPi model show no change in query response time.

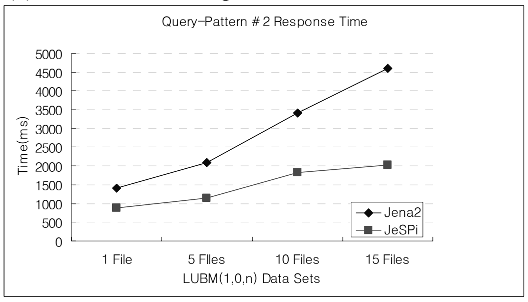
The reasons for the greater improvement in JeSPi's performance compared with Jena can be summarized as follows:

- First, when we search instances of classes, a join operation is executed. In this process, Jena needs to execute the join operation with all tuples in the single table. JeSPi, however, can directly find instances of classes using the table that contains information on pairs, class ID, and instance ID. Therefore, the query response time of JeSPi is faster than that of Jena.

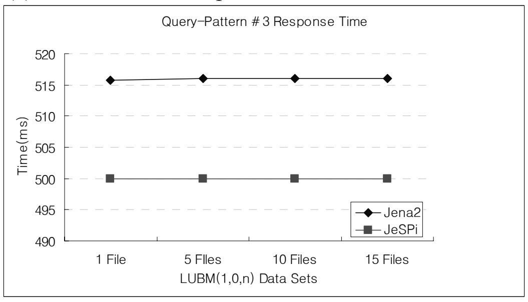
(a) Result of Q-P1 Experiment



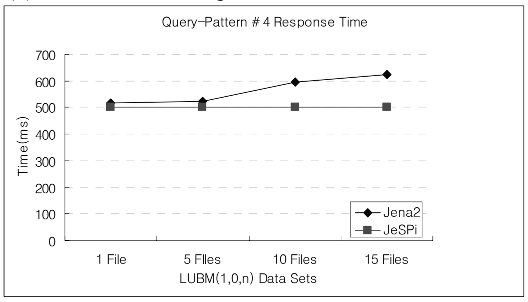
(b) Result of Q-P2 EXperiment



(c) Result of Q-P3 Experiment



(d) Result of Q-P4 Experiment



(e) Result of Q-P5 Experiment

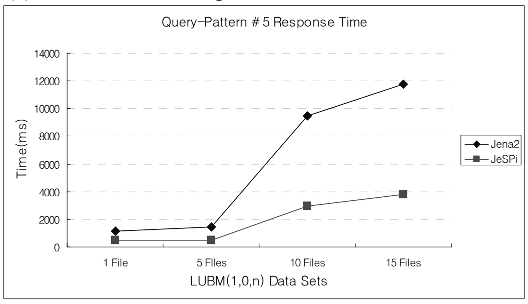


Fig. 5 Expression of the experiment results in graphs

- To find the sub-nodes, i.e., sub-properties or sub-classes, JeSPi uses the table storing hierarchical knowledge only. It can remove unnecessary comparisons, and thus the proposed model shows a faster query processing performance than Jena.

4.2.2 Results of the experiment in terms of the number of tuples, completeness, and accuracy

Table 2 shows the results of the experiment in terms of the number of tuples, completeness, and accuracy of the query results for the Jena and JeSPi storage models. In this paper, completeness means that we measure the degree of completeness of each query answer as the percentage of the entailed answers that are returned by the system. Accuracy is measured as the number of true answers/the number all answers.

For example, given that the required result set is {a, b, d} and the query processing result set is {a, b, c, d}, then completeness becomes 100%. Only three of the results, however, are true, and thus accuracy is 3/4 * 100 = 75%.

Table 2 shows that both completeness and accuracy are 100% for all items. In other words, both storages can return users all exact answers, meaning that both storages return the same answers for all queries, from Q-P1 to Q-P5.

With the experiments shown in Fig. 5 and Table 2, even though both storages provide the same query result, the proposed model supports a more improved query processing performance.

4.2.3 Qualitative evaluation

Table 3 shows the qualitative evaluation between the Jena storage model and the proposed storage model.

The proposed model consumes less query response time than the Jena storage, because the latter stores all data in a non-normalized table structure.

The Jena storage model has high redundancy owing to an inherent structural problem. In Jena, all data are stored as triples in a single table. Hence, if a class has many instances, then the class information is duplicated. Such redundancy also lowers its query processing performance as well as requiring high memory occupation.

As for ease of data comprehension, the relational database model in Jena is stored in a single table, so it is not easy for users to understand the meanings of data. The proposed model, however, is structured conveniently to enable understanding of meanings. It provides support for users to intuitively understand the meanings of data, for example, which are classes or which are their instances. Besides, in the proposed model, ontology data are classified and stored in separate tables, and it also helps users' to understand data.

As for the query modeling cost, JeSPi facilitates the abstraction and comprehension of relationships with specific data because the OWL ontology data are semantically classified into classes, properties, and instances. However, in the Jena storage model, we can

Table 2. Experiment results on the number of tuples, completeness, and accuracy of query results.

Queries	Data sets / Compared items	LUBM(1,0,1)		LUBM(1,0,5)		LUBM(1,0,10)		LUBM(1,0,15)	
		Jena	JeSPi	Jena	JeSPi	Jena	JeSPi	Jena	JeSPi
Q-P1 (C)	The number of query results	43	43	43	43	43	43	43	43
	Completeness (%)	100	100	100	100	100	100	100	100
	Accuracy (%)	100	100	100	100	100	100	100	100
Q-P2 (CI)	The number of query results	10	10	46	46	94	94	146	146
	Completeness (%)	100	100	100	100	100	100	100	100
	Accuracy (%)	100	100	100	100	100	100	100	100
Q-P3 (CH)	The number of query results	6	6	6	6	6	6	6	6
	Completeness (%)	100	100	100	100	100	100	100	100
	Accuracy (%)	100	100	100	100	100	100	100	100
Q-P4 (PH)	The number of query results	3	3	3	3	3	3	3	3
	Completeness (%)	100	100	100	100	100	100	100	100
	Accuracy (%)	100	100	100	100	100	100	100	100
Q-P5 (CHI)	The number of query results	34	34	147	147	294	294	447	447
	Completeness (%)	100	100	100	100	100	100	100	100
	Accuracy (%)	100	100	100	100	100	100	100	100

identify only subjects, predicates or objects. In addition, it makes the management of data easy, because managers can identify semantically and systematically ontology and storage structure.

Table 3 Qualitative evaluation

Items	Jena	JeSPi
Query processing performance	*Low*	*High*
Duplication of data	*Duplicated*	*Normalized*
Ease of data comprehension	*Difficult*	*Easy*
Query modeling cost	*Low*	*High*
Ease of model conversion	*Low*	*High*

5. Conclusion and Future Work

This paper proposes a new storage model aimed at enhancing the efficiency of the Jena storage model. The proposed storage model has been developed for use as a Jena plug-in. For the purposes of this paper, the overall architecture of the proposed storage model was shown, and an experimental evaluation was conducted.

The results of the experiment and the evaluation show that the proposed storage model provides a more improved performance than the Jena storage model. Most significantly, the difference in performance between the two models increases exponentially as the size of the ontology becomes larger.

In the future, the conversion module should be implemented; and research on comparative evaluations with various systems such as Sesame and DLDB will be necessary.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, The Semantic Web, Scientific American Magazine (2001). Available at: http://www.sciam.com/ (accessed 23 January 2008).

[2] D. Beckett, RDF/XML Syntax Specification, 2004.

[3] D. Brickley and R.V. Guha (eds.), RDF Vocabulary Description Language 1.0: RDF Schema, 2004.

[4] D. Connolly, F.V. Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein, DAML+OIL Reference Description W3C Note, 2001. Available at: http://www.w3.org/TR/daml+oil-reference (accessed 23 January 2008).

[5] M. Dean and G. Schreiber (eds.), OWL Web Ontology Language Reference (2004).

[6] I.C. Hsu and S.J. Kao, An OWL-based extensible transcoding system for mobile multi-devices, Journal of Information Science, 31(3) 6 (2005), pp.178 - 195.

[7] L. Liao, K. Xu, and S. Liao, Constructing intelligent and open mobile commerce using a semantic web approach, Journal of Information Science, 31(5) 10 (2005), pp. 407 - 419.

[8] H. Wang and C. Wang, Ontologies for universal information systems, Journal of Information Science 21(3) 1 (1995), pp. 232 - 239.

[9] L. Han, G. Chen, and L. Xie, AASA: A Method of Automatically Acquiring Semantic Annotations, Journal of Information Science, Vol. 8 (2007), pp. 435 - 450.

[10] HP Labs, Jena – A Semantic Web Framework for Java.

[11] E. Prud'hommeaux and A. Seaborne (eds.), SPARQL Query Language for RDF, 2008.

[12] H.S. Kim, H.S. Cha, J.S. Kim, and J.H. Son, Development of an Efficient OWL Document Management System for Embedded Applications, Springer-Verlag, Lecture Notes in Computer Science 3597, (2005), pp. 75-84.

[13] M.J. Park and C.W. Chung, Property-Based OWL Storage Schema in Relational Databases (Unpublished Technical Report, Div. of Computer Science, KAIST, Dec. 2005).

[14] Seaborne, RDQL - A Query Language for RDF, 2004.

[15] Y. Guo, SWAT Projects - the Lehigh University Benchmark (LUBM).

[16] Y. Guo, Z. Pan, and J. Heflin, An Evaluation of Knowledge Base Systems for Large OWL Datasets, Springer-Verlag, Lecture Notes in Computer Science, 3298 (2004) pp. 274-288.
[17] Y. Guo, Z. Pan, and J. Heflin, LUBM: A Benchmark for OWL Knowledge Base Systems, Journal of Web Semantics, 3(2) (2005), pp. 158-182.
[18] D.W. Jeong, M.H. Choi, Y.S. Jeon, Y.H. Han, Y.S. Jeong, and S.K. Han, A Novel Memory-Oriented OWL Storage System, Springer-Verlag, Lecture Notes in Computer Science, 4331 (2006), pp. 542-549.
[19] D.W. Jeong, M.H. Choi, Y.S. Jeon, Y.H. Han, L.T. Yang, Y.S. Jeong, and S.K. Han, Persistent Storage System for Efficient Management of OWL Web Ontology, Springer-Verlag, Lecture Notes in Computer Science, 4611 (2007), pp. 1089-1097.

Dongwon Jeong
He received his Ph.D. degree in Computer Science from Korea University, Seoul, Korea, in 2004. He worked as a full-time instructor from 1999 to 2000 in the Advanced Institute of Information Technology, Korea. He was a Senior Researcher with the Jigunet Corporation from 2000 to 2001. He was a Research Assistant Professor at Korea University from 2004-2005. He was a Visiting Research Scholar (PostDoc.), School of Information Sciences & Technology, Pennsylvania State University, USA, in 2005. Since 2005, he has been a Professor in the Dept. of Informatics and Statistics, Kunsan National University, Korea. He was a committee member of the Data Study Group (SG08.02), Telecommunications Technology Association (TTA), Korea, from 2002 to 2004. Since 2004 he has been a committee member of the Metadata Project Group (PG 606), Telecommunications Technology Association (TTA) of Korea. He has been a committee member of the Data Management Service (ISO/IEC JTC 1/SC 32 Mirror Committee) and the Geographic Information (ISO/TC211) Mirror Committee of the Korean Agency for Technology and Standards (KATS) since 2006 and 2008 respectively. He is and has been a PC member or Program/Publicity/General Chair of many conferences and workshops. His research interests include Data Integration, Semantic Web, Semantic Sensor Network, Semantic GIS, Semantic Grid, and Security.

Heeyoung Shin
He received a Master's degree in Computer Science from Korea University, Seoul, Korea, in 2008. He has been working for Hana Financial Group, Seoul, Korea since 2008. His research interests include Data Integration, Semantic Web, and Metadata Management.

Doo-Kwon Baik
He received his Ph.D. degree in Computer Science at Wayne State University, USA, in 1986. He was the Dean of the College of Information and Communications from 2002 to 2006, and has been a director of the Research Institute of Computer, Information and Communication, Korea University since 2006. He has also been a Korean Head of Delegates (HOD), ISO/IEC JTC 1/SC 32 since 1993. He is and has been a member of many societies, conferences, and workshops. His research interests include Data Engineering, Software Engineering, and Modeling and Simulation.

Young-Sik Jeong
He received a Ph.D. degree in Computer Science from Korea University, Seoul, Korea, in 1993. He was a visiting scholar at the Dept. of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA, in 1997, as well as at the Dept. of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202, USA, in 2004. He has been a director of the Culture Contents Technology Society since 2006; a member of the Appraisal Committee of the University Industrial Technology Force (UNITEF) for Patents since 2003; a chairman of the IEC/TC 108 Korean Agency for Technology and Standards since 2003; a member of the Committee of the IEC/TC 100 Korean Agency for Technology and Standards since 2002; and a member of the Korean Committee, ISO/IEC JTC 1/SC 25 since 2002. He has also been a PC Member or Program/Publicity/General Chair of many conferences and workshops. He has been a Professor in the Department of Computer Engineering, Wonkwang University, Iksan, Korea since 1993. His interests include Grid Computing, Semantic Grid, Mobile Computing, and e-Learning.