Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

An XPath-based OWL storage model for effective ontology management in Semantic Web environment*

Jinhyung KIM[†1], Dongwon JEONG[2], Doo-kwon BAIK[†‡1]

([1]*Department of Computer Science and Engineering, Korea University, Seoul 136-713, Korea*)

([2]*Department of Informatics and Statistics, Kunsan National University, Kunsan 573-701, Korea*)

[†]E-mail: jinhyung98.kim@gmail.com; baikdk@korea.ac.kr

Received May 8, 2008; Revision accepted Oct. 9, 2008; Crosschecked Mar. 27, 2009

Abstract: With the rapid growth of the Web, the volume of information on the Web is increasing exponentially. However, information on the current Web is only understandable to humans, and this makes precise information retrieval difficult. To solve this problem, the Semantic Web was proposed. We must use ontology languages that can assign data the semantics for realizing the Semantic Web. One of the representative ontology languages is the Web ontology language OWL, adopted as a recommendation by the World-Wide Web Consortium (W3C). OWL includes hierarchical structural information between classes or properties. Therefore, an efficient OWL storage model that considers a hierarchical structure for effective information retrieval on the Semantic Web is required. In this paper we suggest an XPath-based OWL storage (XPOS) model, which includes hierarchical information between classes or properties in XPath form, and enables intuitive and effective information retrieval. Also, we show the comparative evaluation results for the performance of the XPOS model, Sesame, and the XML file system-based storage (XFSS) model, in terms of query processing and ontology updating.

Key words: XPath, Web ontology language, Hierarchical structure, Ontology storage, Semantic Web
doi:10.1631/jzus.A0820355 **Document code:** A **CLC number:** TP31

INTRODUCTION

With the rapid growth of the Web, large volumes of Web-based information are being created and propagated. However, precise searching of requested information is becoming more difficult, since there is a considerable increase in the volume of information on the Web. Precise retrieval of the requested information is becoming the principal issue, and this is more important than rapid retrieval of the data. The data on the current Web is designed only for human readability and understandability. At this point, the realization of complete semantic interpretation and understanding in a computing environment is impossible. Therefore, the Semantic Web was proposed to solve these problems (Decker *et al*., 2000; Lee *et al*., 2001; Fahmi *et al*., 2007).

The Semantic Web is an intelligent Web for inter-communication between machines, with data representation in the form of a new language that computers can interpret. The Semantic Web is designed for interpretability and understandability of computer data, to overcome the limitations of current forms of Web data representation that only humans can read and understand. The principle of the Semantic Web is the translation of semantics between information resources in the form of a new language that computers can interpret. Therefore, computers can interpret semantics of information resources and process the information themselves, with inter-communication of information between computers. We must use an ontology language that can describe the semantics of information formally, to enable complete construction of the Semantic Web. Resource description framework (RDF) is the standard adopted by the World-Wide Web Consortium (W3C) for describing

‡ Corresponding author
* Project supported by the Brain Korea 21 Project

metadata (Beckett, 2004; Herman *et al.*, 2004; Stuckenschmidt *et al.*, 2004). RDF can describe metadata with rich expressions as well as possess the advantages of hyper text markup language (HTML) and extensible markup language (XML) (Carroll and Stickler, 2004; Koffina *et al.*, 2005; Li and Wang, 2006). The Web ontology language OWL recently recommended by W3C is designed for applications that need to process the content of the information, instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF-Schema (RDFS), by providing an additional vocabulary along with a formal semantics (Smith *et al.*, 2004).

Therefore, we need an OWL storage model for effective information search. We must consider the hierarchical structure of classes and properties for effective storage and searching. RDB-based storage systems, such as Jena (Carroll and Stickler, 2004; McKenzie *et al.*, 2006; Lausen *et al.*, 2008), Sesame (Broekstra *et al.*, 2002), 3-Store (Riddoch *et al.*, 2002; Harris and Gibbins, 2003), and Hawk (Pan, 2008), analyze the structure of OWL documents and store information in a relational database. In addition, XML file system based storage (XFSS) systems (Min *et al.*, 2003; Park *et al.*, 2007; Woo *et al.*, 2007) are representative systems that consider the hierarchical structure of OWL data and store information in an XML file system. However, Sesame is inefficient, due to the increasing number of iterations in the searching of the hierarchical structure, with an increasing hierarchical level. An XFSS system is not efficient either, because we must access XML storage and relational storage simultaneously, and obtain integrated query results whenever a query is made.

In this paper, we propose an XPath-based OWL storage model (XPOS), which considers hierarchical structures of OWL data, to support more effective extraction of hierarchical information and query processing than Sesame and the XFSS system. XPOS stores all data about OWL documents in a relational database, and hierarchical information in XPath form. Therefore, XPOS can search hierarchical information effectively, without iterative searching, to obtain the relationship between super-class/super-property and sub-classes/sub-properties.

This paper is organized as follows. In Section 2 we describe Sesame and the XFSS system, which are representative OWL storage systems, and related works that consider the hierarchical structure. In Section 3 we present the XPOS model, architecture, and detailed modules in the XPOS system. In addition, we illustrate the translation and storage processes of the XPOS system, intermediate data, and final data stored in the XPOS model, with an OWL example. In Section 4 we make a comparison of the XPOS model, Sesame, and the XFSS XML database (DB) based system via a performance evaluation. Using quantitative experiments, we validate the superiority of the XPOS model over the conventional OWL storage systems. Finally, we conclude this paper with future work in Section 5.

RELATED WORKS

Relational database based OWL storage system

Relational DB based OWL storage systems analyze information about OWL documents and store that information in a relational database with a storage schema. Typical relational DB based OWL storage systems are Jena (Carroll and Stickler, 2004; McKenzie *et al.*, 2006; Lausen *et al.*, 2008), 3-Store (Riddoch *et al.*, 2002; Harris and Gibbins, 2003), Hawk (Pan, 2008), and Sesame (Broekstra *et al.*, 2002).

Jena is a popular Semantic Web toolkit for Java programmers. Studies of Jena commenced in 2000, and Jena2 was released in 2003. The main contribution of Jena is the rich application program interface (API) model for manipulating RDF graphs. Based on this API, Jena provides various tools, including I/O modules for: RDF/XML (Carroll, 2001; Carroll and de Roo, 2004), N3 (Lee, 2000), N-triple (Grant and Beckett, 2004), and RDQL (Miller *et al.*, 2002). Using the API the user can choose to store RDF graphs in memory or in persistent storage. Jena provides an additional API for manipulating DARPA agent markup language+ontology inference layer (DAML+OIL). However, because Jena manages hierarchical structural information about classes and properties in only one table, Jena is inefficient in terms of query processing, due to the many join operations (Jeon *et al.*, 2005).

3-Store efficiently supports RDF and RDFS entailments over relatively large RDF knowledge bases,

using a relational database back-end to perform the queries (Harris and Gibbins, 2003). At present, 3-Store is intended to extend the indexing scheme techniques used for the tables to represent triples in the schema, because the choice of indexes can dramatically affect the query or assertion time.

Hawk is a repository framework and toolkit that supports OWL. It provides APIs as well as implementations for parsing, editing, manipulating, and preserving OWL ontologies (Pan, 2008). It contains the following storage models: SimpleMemory, DLMemory, SimpleDB, and DLDB. Relational DB based storage systems include Parka, Redland, and TAP. However, most systems do not represent a storage schema explicitly. Therefore, we use Sesame, which represents the storage schema explicitly, as a representative relational DB based storage system.

Sesame is a system developed as a part of On-To-Knowledge in the information society technologies (IST) project. Sesame can support storage, search, and inference of an ontology, with RDF and RDFS. Key tables in Sesame for storage are class, property, resources, subClassOf, subPropertyOf, and triple tables; additional information about an ontology is stored in domain, range, namespaces, type, labels, comment, and literals tables. A storage schema and a table description of Sesame are shown in Fig.1 and Table 1, respectively. The class table includes information about classes in ontology documents. Names of all classes are stored in the class table, excluding hierarchical structures among classes. The property table contains information about properties in

Table 1 Table description of Sesame

Table name	Contents
Class	Class information in ontology
Property	Property information in ontology
Namespaces	Namespace information including ontology
Resources	Resource information about namespaces and class local names
Range	Value range information about classes and properties
Domain	Domain information about classes and properties
Type	Relationship information between classes and namespaces
Labels	Relationship information between literals and namespaces
Triples	Relationship information among classes, instances, and properties
Comment	Comment information about relationship between literals and namespaces
Literals	Value information included in classes
subClassOf	Hierarchical structural information between classes
subPropertyOf	Hierarchical structural information between properties

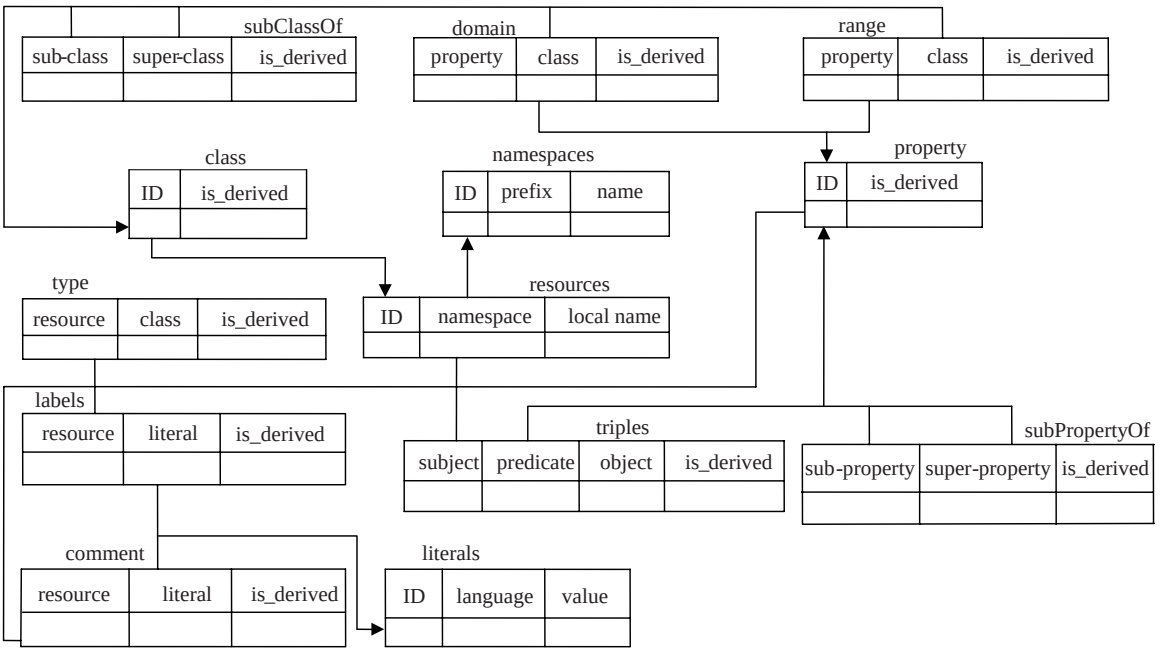


Fig.1 Storage schema of Sesame

ontology documents by storing the names of all properties, excluding hierarchical structures among properties. The literals table includes information about namespaces and instances included in classes. The literals table consists of an ID attribute as an identifier of an instance, and a value attribute storing instance values. The subClassOf table, which is composed of a super-class attribute for parent classes and a sub-class attribute for child classes, contains hierarchical structural information between classes. The subPropertyOf table, which consists of a super-property attribute for parent properties and a sub-property attribute for child properties, includes hierarchical structural information between properties. The subClassOf table and the subPropertyOf table contain only information about adjacent parent/child classes and properties. The triples table includes relationship information among classes, instances, and properties in subject-predicate-object form.

However, as the depth of a hierarchical structure increases and the OWL document grows more complex, operations have to be iterated. To extract hierarchical structures between classes or properties, we search the parent class/property from the super-class attribute or the super-property attribute in the subClassOf table or the subPropertyOf table. Then, we search the child class/property that has a relationship with the searched parent class/property, and search the child class/property that has a relationship with the searched child class/property as the new parent class/property again. This search process must be iterated until there are no more child classes/properties. Table 2 shows the limitations of Sesame in terms of the extraction of hierarchical structural information. In Table 2, we must iterate 10 operations to extract hierarchical structural information between nodes A and K. Sesame is inefficient, due to unnecessary iteration, and if a hierarchical structure is complicated, the time for extraction of hierarchical structural information increases.

Table 2 Hierarchical structure in Sesame

Super-class/ Super-property	Sub-class/ Sub-property	Super-class/ Super-property	Sub-class/ Sub-property
A	B	F	G
B	C	G	H
C	D	H	I
D	E	I	J
E	F	J	K

XML file system based OWL storage (XFSS) system

The XFSS system is a system for efficient storage and searching of hierarchical information in an OWL document, using an XML file system. In the XFSS system, hierarchical structural information is stored in an XML file, and information about classes, properties, and instances is stored in a relational database (Min *et al.*, 2003; Park *et al.*, 2007; Woo *et al.*, 2007). In addition, the XFSS system creates additional XML documents for hierarchical structural information, and stores the XML documents in XML storage. This system creates an XML document, including hierarchical information about classes and properties. Fig.2 illustrates the XFSS system architecture for storing OWL data. If OWL documents are provided, an OWL parser parses the OWL documents and extracts information about classes, properties, instances, hierarchical structure, and constraints. The hierarchical information about classes and properties is translated into an XML document by an XML Manager. The created XML document, including hierarchical structural information, is stored in the XML file system. Other information, such as information about classes, properties, and instances, is stored in a relational database by the Ontology Manager. Fig.3 shows an XML document created for hierarchical structural information about classes and properties.

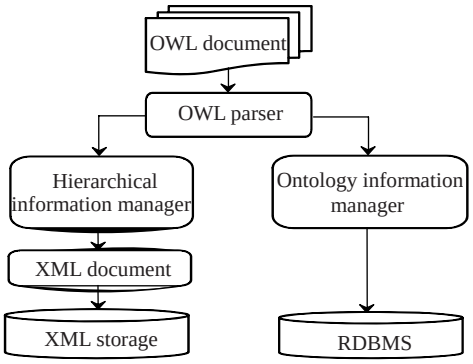


Fig.2 Architecture of an XFSS system

In the case of the XFSS system, whenever a query is processed, we access the XML file system and acquire hierarchical structural information about classes and properties. Then, we access the relational database and obtain final query results with hierarchical structural information from the XML file system.

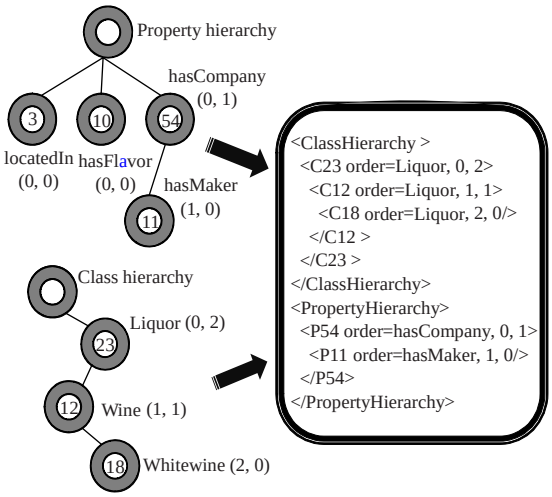


Fig.3 XML document for hierarchical structure

However, we must access both the XML file system and the relational database when we process inputted queries. In addition, we must translate inputted queries into the form of XQuery and SQL, to acquire information from the XML file system and the relational database. Therefore, this system is inefficient in terms of query processing performance. The XFSS system creates tables for each root class and stores instances included in the root class in these tables for efficient searching of instances. By this method we can search instances included in specified classes effectively. However, if queries for searching all instances or instances of many classes are provided, the query processing time increases, due to several join operations between tables. In this case, the XFSS system can be less efficient than Sesame. Table 3 presents the storage schema for the XFSS system.

Table 3 Storage schema of the XFSS system

Classname_Table

preorder	postorder	uid

Propertyname_Table

preorder	postorder	class_uid	value_u	value_s

XPATH-BASED OWL STORAGE (XPOS) MODEL

In this section, we describe an XPath-based OWL storage (XPOS) model for overcoming limitations in terms of storage and query processing of hierarchical structural information of conventional OWL storage systems. One aim of the XPOS model is to support a more effective and intuitive information search of OWL documents than Sesame and the XFSS system. In addition, we define a schema for the XPOS model, and an architecture for the XPOS system. We also present the translation process from OWL documents to the XPOS model with a simple example.

XPOS model definition

The XPOS model includes information about the hierarchical structure between classes or properties. In addition, the XPOS model is designed for effective search of hierarchical structural information. Table 4 presents the storage schema of the XOPS model.

Table 4 Storage schema of the XPOS model

Class_Table

class_id	class_name	class_path	root_id

Instance_Table

inst_id	inst_name	class

Property_Table

prop_id	prop_name	prop_path	root_id

Triples_Table

subject	predicate	object

The XPOS model consists of a class table, a property table, a triples table, and an instance table. The storage schema of the XPOS model is similar to that of Sesame, excluding the class_path and prop_path attributes, which include hierarchical structural information. The class and property tables include an ID attribute for classes/properties identification, and a name attribute for specific names of classes/properties. The class and property tables contain a path attribute and a root_id attribute, for information about hierarchical structure between classes and properties. In the path attribute, information about hierarchical structure is stored in XPath form (e.g., Student/Graduate School Student/Ph.D. Student). The instance table includes an inst_id attribute for instance identification, an inst_name attribute, and a class attribute, in which instances are included. The triples table contains relationships among classes, instances, and properties. In the subject and object attributes, values of class_id and inst_id can be stored. In a predicate attribute, values in prop_id can be stored.

For extraction of information about the hierarchical structure between classes or properties from OWL documents, the following processes are needed. First, we analyze the schema for the OWL document and create a data graph with a hierarchical relationship between classes and properties (Jang *et al.*, 1999; Kobayashi *et al.*, 2005; Zhou *et al.*, 2006). Second, we perform a depth-first search from root classes/properties to leaf classes/properties, based on the created data graph, and create paths for each node. When we create paths for each node, we search from root nodes to leaf nodes. Then, if we arrive at a leaf node, we create a path for the leaf node and the intermediate nodes. However, we create paths for intermediate nodes just once, and thus we can avoid duplicate path creation. Extracted path information is stored in a path attribute in the class and property tables. Table 5 presents descriptions about the symbols and notations used in the path creation algorithm.

Table 5 Description of symbols and notations

Notation	Description
Node.number	Node number assigned by DFS search in the data graph
Node.visiting_flag	Visited nodes have flag set to 1; nodes that have not been visited have flag set to 0
Node.child_flag	Nodes that do not have child nodes have flag set to 0; nodes that have child nodes have flag set to 1
Node.sibling_flag	Nodes that do not have sibling nodes have flag set to 0; nodes that have sibling nodes have flag set to 1
visiting_node[ ]	Array for representing visited node lists
DFS_visit( )	Function for search in the data graph by the DFS search method
Nextnode( )	Function for description of the next ordered node in DFS search
path_temp[ ]	Array for storing created path temporarily before duplication checking
path_storage[ ]	Array for storing created path after duplication checking
Createpath( )	Function for creating XPath from the root node to the current node
Pathcheck( )	Function for checking duplicated creation of node paths
Backtrackingpath( )	Function for execution of path backtracking in the data graph

In the data graph, each node consists of the following elements. Definition 1 presents the node constitution in the data graph.

Definition 1 (Node constitution in the data graph) Each node in the data graph is denoted by a 5-tuple: $N_{(\text{name})}=(N_a, N_u, V_f, C_f, S_f)$, where N_a represents a specific name for each node, N_u represents a specific node number assigned by DFS searching when the data graph is created, V_f represents whether a node has been visited and a path has been created for the node or not, C_f represents whether a node has child nodes or not, and S_f represents whether a node has sibling nodes or not. If a node has been visited and a path has been created for the node, the visiting_flag of this node is 1, otherwise 0; a node that has no child nodes has a flag set to 0, otherwise 1; a node that has no sibling nodes has a flag set to 0, otherwise 1.

Definition 2 (Case definition in the path creation) When we create a path for each node from a data graph, there are two representative cases (1 and 2) and four detailed cases (1, 2-1, 2-2, and 2-3):

Case 1: Node.visiting_flag=0 and Node.child_flag=1.

Case 2: Node.visiting_flag=0 and Node.child_flag=0.

Case 2-1: Node.sibling_flag=1 and Sibling_node.visiting_flag=0.

Case 2-2: Parent_node.sibling_flag=0 and Parent_node.sibling_node.visiting_flag=0.

Case 2-3: Ascendant_node.sibling_flag=0 and Ascendant_node.sibling _node.visiting_flag=0.

Definition 2 defines various cases of path creation. Case 1 represents the case where the current node has not been visited and has child nodes. In case 1, we store the name of the current node in the visiting_node array and search the child node as the next order. Case 2 describes the case where a node has not been visited and does not have child nodes; the current node is a leaf node. In this case, we create a path for the node and the intermediate nodes between the root node and the current node. However, we must check for duplicate path creation in this case, because the intermediate nodes of all sibling nodes are the same. To check for duplication of node paths, we temporarily store the created path in a path_temp array. The path stored in the path_temp array is compared with the path included in a path_storage array. If the created path is not contained in the path_storage array, the path is finally stored in the path_storage array. Cases 2-1, 2-2, and 2-3 are backtracking cases, after we search leaf nodes. Case 2-1

illustrates the case where a leaf node has sibling nodes that have not been visited. In this case, we perform backtracking to a parent node and search sibling nodes in the next ordering. Case 2-2 represents the case where a leaf node does not have sibling nodes that have not been visited. In this case, we perform backtracking to a parent node that has sibling nodes that have not been visited, and search sibling nodes of the parent node. If there are no sibling nodes of the parent node that have not been visited, we perform backtracking to the ascendant node. If the ascendant node has unvisited sibling nodes that have not been visited, we search these nodes, as in case 2-3. This process is iterated until we have searched every node in the data graph, and path creation is completed when there are no nodes in cases 1, 2-1, 2-2, and 2-3. Fig.4 shows the flow of path creation in terms of search cases.

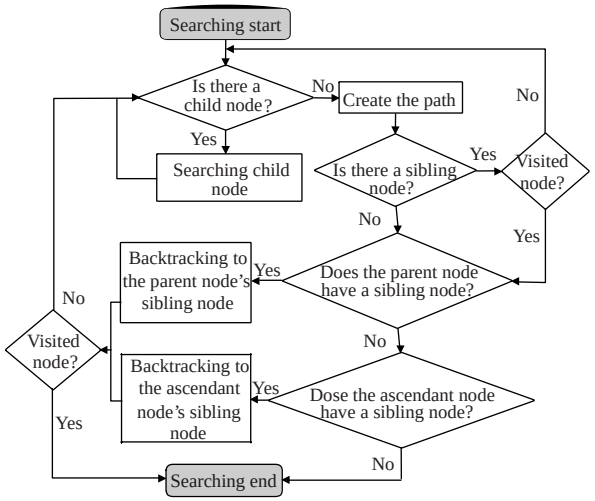


Fig.4 Flow of path creation

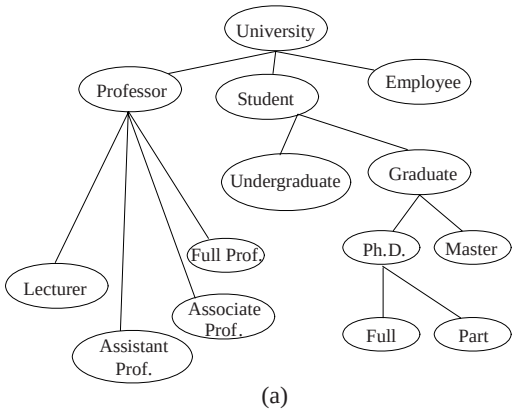
In addition, we store the ID of the root class/property in the root_id attribute in the class and property tables, for efficient access and searching of ontology data. If several ontologies are included in the OWL document, information about the root_id attribute enables us to search and modify the ontology easily. Also, when we modify or reconstruct the ontology, we check the root class/property of the ontology. Then, we just modify or reconstruct classes/properties included in the root class/property. Using the root_id attribute, we can reduce ontology modification and reconstruction time. Table 6 presents the entire path creation algorithm.

Table 6 Path creation algorithm

```
Procedure:
 Initialize i=1, j=0, k=0, m=0;
 Class Node
  Initialize string name=null;
  Initialize int number=0;
  Initialize int visiting_flag=0;
  Initialize child_flag=0;
  Initialize sibling_flag=0;
 For (Root_node to Final_leaf_node)
  Initialize visiting_node[ ];
  DFS_visit(Node);
  if (Node.visiting_flag==0 && Node.child_flag==0)
   visiting_node[k]=Node.name;
   Increase k;
   Node.visiting_flag=1;
   Nextnode(child_node);
  End if
  else if (Node.visiting_flag==0 && Node.child_flag==1)
   visiting_node[k]=Node.name;
   Do while (visiting_node[k]!=null)
    Initialize k=0;
    Initialize path_temp[ ], path_storage[ ];
    path_temp[k]=Createpath(Root_node, visiting_node[k]);
    For j=0 to j_max
     For m=0 to m_max
      Pathcheck(path_temp[j], path_storage[m]);
     End For
     Increase m;
     if (path_temp[j]!=path_storage[m])
      path_storage[m_max+1]=path_temp[j];
     End if
    End For
    Increase j;
   Increase k;
   if (Node.sibling_flag==0 && sibling_node.visiting_
    flag==0)
    Backtrackingpath(sibling_node);
    Nextnode(sibling_node);
   else
    Do while (parent_node!=null)
     Backtrackingpath(parent_node);
     if (parent_node.sibling_flag==0 || parent_node.
      sibling_node.visiting_flag==0)
      Nextnode(parent_node.sibling_node);
     else Loop
  End else if
End Procedure
```

Fig.5 shows the data graph and path attribute information, represented in XPath form, of the hierarchical structure of classes of an OWL document. If information about the hierarchical structure of classes and properties is stored as shown in Fig.5, we can extract hierarchical structural information efficiently from the class and property tables, without iterating or

accessing other storages, such as the XFSS system. We can extract hierarchical structural information about a specified class or property via the path attribute in the class or property tables. In addition, when we extract a sub-class or sub-property from a specified class or property, we can extract a sub-class or sub-property easily by searching the path attribute. In the case where the hierarchical structure of an OWL data is complicated, we can extract information about the hierarchical structure rapidly and effectively by searching the path attribute in the class or property tables.



(a)

class_ID	class_name	path	root_ID
01	University	University	University
02	Professor	University/ Professor	University
…	…	…	…
13	Full	University/Student/Graduate /Ph.D./Full	University
14	Part	University/Student/Graduate /Ph.D./Part	University

(b)

Fig.5 Data graph (a) and path attribute information (b)

However, recently, terabyte or petabyte volumes of data are being stored and managed in a database. Mass data storage can be very complicated and data can have a high level of depth. Therefore, mass data storage can result in a storage capacity problem, in terms of path attributes. Nevertheless, information stored in the path attribute in the class and property tables is simple string data, and the capacity of mass data storage is not based on the complexity of the structure, but the quantity of instance values. Therefore, we will consider the storage capacity problem concerning the path attribute in the class and property tables as a part of future studies.

XPOS system architecture

Fig.6 presents the XPOS system architecture for storing OWL data in a relational database. If OWL documents are provided, the OWL parser parses documents. First, the OWL parser analyzes documents syntactically and semantically. Then, the OWL parser extracts information about classes, properties, instances, hierarchical structure, and constraints from OWL documents. Information about the OWL documents is extracted and categorized into two components by the hierarchical info extractor and the ontology info extractor: one component is the hierarchical information extracted by the hierarchical info extractor; the other is ontology information extracted by the ontology info extractor. Hierarchical structural information is translated into a data graph by the data graph generator, which analyzes the schema of hierarchical information extracted by the hierarchical info extractor, and creates data graphs about the hierarchical structure of classes and properties. The hierarchical info manager performs a depth-first search from root classes/properties to leaf classes/properties, and extracts paths for every node in the data graph. Information about classes, properties, and instances, excluding hierarchical structural information, is analyzed and extracted by the ontology info extractor. The ontology info extractor extracts relationships between classes and instances, or between classes and properties. Finally, hierarchical structural information extracted by the hierarchical info manager, and ontology information extracted by the ontology info manager, are translated by the converter and stored in the XPOS model.

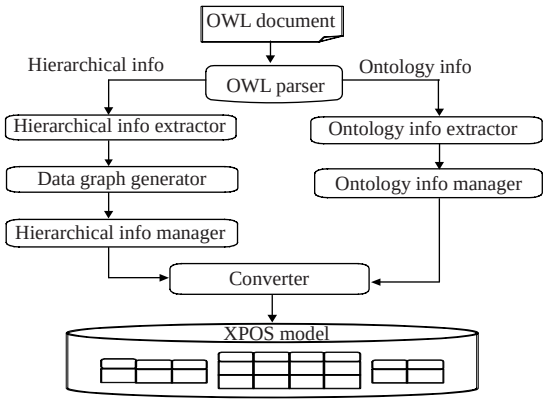


Fig.6 XPOS system architecture

Translation and storage processes of the XPOS system

In this subsection, we describe the translation and storage processes of the XPOS system, with sample OWL data (University0_0.owl) created by the Univ-Bench Artificial (UBA) data generator.

The sample OWL data includes information about university, department, and activity in university. This document is used for describing the translation and storage processes of the XPOS system as a simple example. If the University0_0.owl document is inputted into the XPOS system, the OWL parser parses the document, and checks for syntactical or grammatical errors. If there is a parsing problem, the conversion and storage process for the OWL document by the XPOS system is terminated at this step. The OWL document is analyzed by the hierarchical info extractor and the ontology info extractor and classified into a hierarchical structural information component and an ontology information component, including classes, properties, instances, and triples information. Then, the data graph generator creates a data graph based on hierarchical structural information. Fig.7 shows data graphs of the hierarchical structure of classes and properties. After the data graph is created, the hierarchical info manager performs a depth-first search from the root node to the leaf nodes. The hierarchical info manager creates a node path when it arrives at leaf nodes. Then, node paths for intermediate nodes are also created. When it searches sibling nodes of leaf nodes, node paths for intermediate nodes are not created. Via this method, we can reduce duplicate path creation. Table 7 presents the search ordering, search path, and created path of the class hierarchy data graph and property hierarchy data graph.

If the node search and path creation are completed, hierarchical structural information is stored in the path attribute in the class and property tables. In addition, names of classes, names of properties, values of instances, and values of triples are stored in the class, property, instance, and triples tables, respectively. The IDs in the class, property, and instance tables are sequentially assigned, and hierarchical structural information created by the hierarchical info manager is stored in the path attribute in XPath form. In the root_id attribute, we store the root nodes of the class and the property data graphs, for efficient

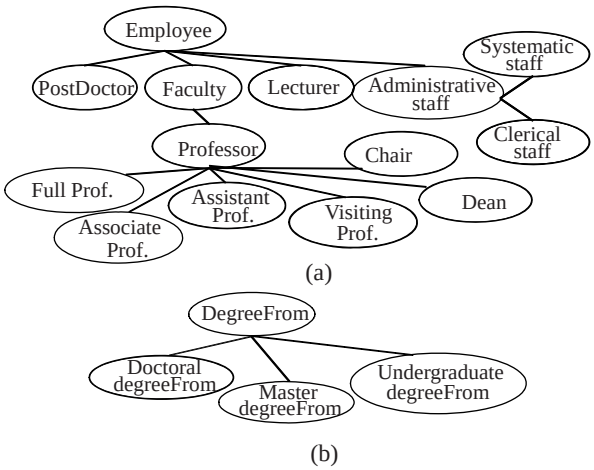


Fig.7 (a) Class hierarchy data graph; (b) Property hierarchy data graph

Table 7 Search ordering, search path, and created path of the class/property data graph

Order	Data graph	Search path	Created path
1	Class	Employee->PostDoctor	Employee/PostDoctor
2		Employee->Faculty->Professor->Full Professor	Employee/Faculty Employee/Faculty/Professor Employee/Faculty/Professor/Full Professor
3		Employee->Faculty->Professor->Associate Professor	Employee/Faculty/Professor/Associate Professor
4		Employee->Faculty->Professor->Assistant Professor	Employee/Faculty/Professor/Assistant Professor
5		Employee->Faculty->Professor->Visiting Professor	Employee/Faculty/Professor/Visiting Professor
6		Employee-> Faculty->Professor->Dean	Employee/Faculty/Professor/Dean
7		Employee->Faculty->Professor->Chair	Employee/Faculty/Professor/Chair
8		Employee->Lecturer	Employee/Lecturer
9		Employee->Administrative Staff->Systematic Staff	Employee/Administrative Staff Employee/Administrative Staff/Systematic Staff
10		Employee->Administrative Staff->Clerical Staff	Employee/Administrative Staff/Clerical Staff
11	Property	degreeFrom->Doctoral degreeFrom	degreeFrom/Doctoral degreeFrom
12		degreeFrom->Master degreeFrom	degreeFrom/Master degreeFrom
13		degreeFrom->Undergraduate degreeFrom	degreeFrom/Undergraduate degreeFrom

ontology modification and reconstruction. We store the names of instances and related classes in the instance table. The triples table includes triples information as classes-properties-instances or classes-properties-classes and each triple is represented as a uniform resource identifier (URI). Table 8 presents the XPOS model translated by the XPOS system, with the simple OWL document.

Table 8 XPOS model

Class_Table

class_id	class_name	class_path	root_id
c01	Employee	Employee	Employee
c02	PostDoctor	Employee/PostDoctor	Employee
...	...	...	...

Property_Table

prop_id	prop_name	prop_path	root_id
p01	degreeFrom	degreeFrom	degreeFrom
p02	Doctoral degreeFrom	degreeFrom/Doctoral degreeFrom	degreeFrom
...	...	...	...

Instance_Table

inst_id	inst_name	class
i01	http://www.Department0.University0.edu/AssistantProfessor0	Assistant Professor
i02	http://www.Department0.University0.edu/FullProfessor2	Full Professor
...	...	...

Triples_Table

subject	property	object
Http://www.Department5.University0.edu/AssistantProfessor4	Http://www.lehigh.edu/~zhp2/2004/0401univ-bench.owl#doctoraldegreeFrom	Http://www.University932.edu
Http://www.Department5.University0.edu/AssistantProfessor4	Http://www.lehigh.edu/~zhp2/2004/0401univ-bench.owl#masterdegreeFrom	Http://www.University264.edu
...	...	...

PERFORMANCE EVALUATION

In this section, we describe the performance evaluation for query processing among the XPOS system, Sesame, and the XFSS system.

Experimental environment and data

For the comparative experiment, we used a Pentium Dual CPU 2.66 GHz system with 1 GB memory. In addition, we used Oracle 9i as the DBMS, and Java as the implementation language.

The data used in the experiment was the dataset created by UBA (Guo *et al.*, 2005), which is an ontology creation tool developed by Lehigh University, able to create ontologies of various sizes. The created ontology data consisted of contents about university, department, and university activity, and included 43 classes and 32 properties. In the comparative experiment, we created LUBM(1, 0), LUBM(5, 0), and LUBM(10, 0) OWL ontologies, which contained OWL files for 1, 5, and 10 universities, respectively. LUBM(*N*, *S*) means that the dataset contains *N* universities, beginning with university 0, which were generated using a seed value of *S*. OWL ontologies were translated by the XPOS system, Sesame, and the XFSS system, and stored in each storage model. Then, we evaluated the performance in terms of the query processing time and the ontology updating time. At present, systems storing ontology information in a relational database are Sesame, 3-Store, DLDB, Hawk, the XFSS system, etc. However, many systems do not present an explicit storage schema structure. Therefore, we used Sesame and the XFSS system, which have released explicit storage schemas, for an accurate experiment. Fig.8 shows data graphs of classes and properties for the experimental data, the LUBM ontology data.

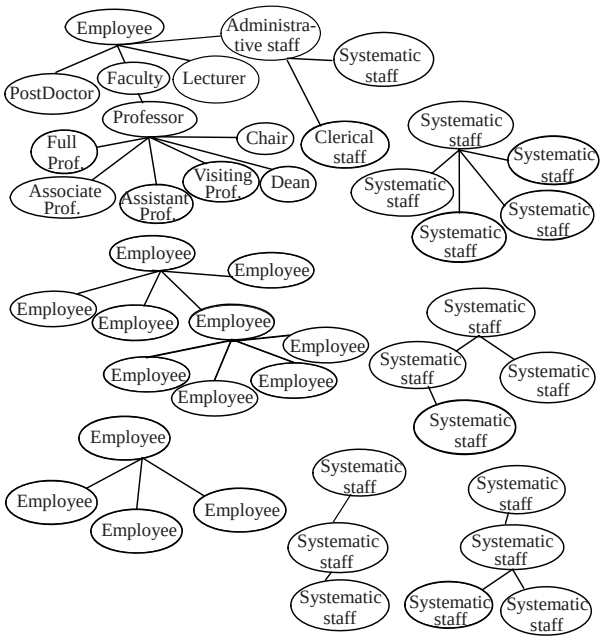


Fig.8 Hierarchical structure of Univ-Bench Artificial (UBA) ontology data

Query processing time

We used six experimental queries for a comparative evaluation in terms of the query processing time. Query 1 was a query for searching all

sub-classes of a specified class. Query 2 searched all sub-classes of a specified class, then all instances included in the sub-classes. Query 3 was a query for searching all sub-properties of a specified property. Query 4 searched all sub-properties of a specified property, and all objects related to sub-properties. Query 5 searched all sub-properties of a specified property, and all subjects related to sub-properties. Query 6 searched all sub-properties of a specified property and all objects related to sub-properties, and the classes related to the objects.

All queries in the performance evaluation were related to searches of the hierarchical structure between classes or properties. Therefore, we performed an experiment focusing on a comparative evaluation among storage systems, in terms of the query processing performance with respect to the storage structure of hierarchical structural information. In other words, we focused on a performance comparison of the extraction of hierarchical structure between classes and properties.

We measured the average time by repeating the experiment 100 times, for a more precise experiment. Fig.9a presents the processing results for the three systems, for query 1. Though the size of the OWL file increases, the number of classes remains constant—the experimental results of query 1 are unrelated to the size of the OWL file. To process query 1, Sesame performs an iterative search of sub-classes, to search sub-classes of the employee class. The XFSS system accesses the XML storage and checks the root class of the employee class based on the XML file, including hierarchical structural information. Then, the XFSS system searches sub-classes in the employee table. However, in the case of the XPOS system, we can search sub-classes by searching the path attribute in the class table. Therefore, the XPOS system shows the best performance in terms of the processing time of query 1, because the XPOS system searches only one table without any join operation, for performing query 1.

Fig.9b shows the processing results for the three systems for query 2. As the size of the OWL file increases, the number of instances also increases; if the size of the OWL file increases, the query response time for query 2 increases. Sesame and the XPOS system retrieve instances by searching the instanceOf/ instance table, based on the results of query 1. Sesame requires more iterations for searching sub-classes than the XPOS system, for the processing of query 1. Therefore, query processing performance for query 2 of the XPOS system is better than that of Sesame. However, the XFSS system requires many more join operations, because instances are stored in many classs and property tables. As a result, the XFSS system involves higher computational costs for query 2 than the XPOS system and Sesame.

Fig.9c illustrates the processing results for the three systems for query 3. As the size of the OWL file increases, the number of properties remains constant—the size of the OWL file does not affect the processing time of query 3. Sesame needs more iterations for searching sub-properties of the memberOf property in the subPropertyOf table. The XFSS system accesses the XML storage and checks the root property of the memberOf property. Then, it searches all sub-properties by searching the memberOf table with hierarchical structural information. In the XFSS system, instances are stored in each property table. Though the number of properties is not great, we must search as many property tables as the number of stored instances. However, the XPOS system can search sub-properties only by searching the path attribute in the property table. Therefore, in the case of query 3, the XPOS system shows the best performance in terms of the query response time, because the XPOS system performs an efficient search of the hierarchical structure, in only one class/property table.

Figs.9d and 9e show the processing results of the three systems for queries 4 and 5, respectively. As the size of the OWL file increases, the numbers of instances and objects in the triple also increase; the size of the OWL file is directly proportional to the processing time of query 4. However, as the size of the OWL file increases, the number of subjects in the triple structure remains constant. In other words, there is no relationship between the size of the OWL file and the response time of query 5. Queries 4 and 5 are similar queries, and search objects or subjects in the triple properties retrieved as the results of query 3 are included in a predicate in the triple. Sesame and the XPOS system retrieve objects and subjects by searching the triples table based on the results of query 3. In terms of the query processing performance of queries 4 and 5, the XPOS system shows better performance than Sesame, due to the difference in

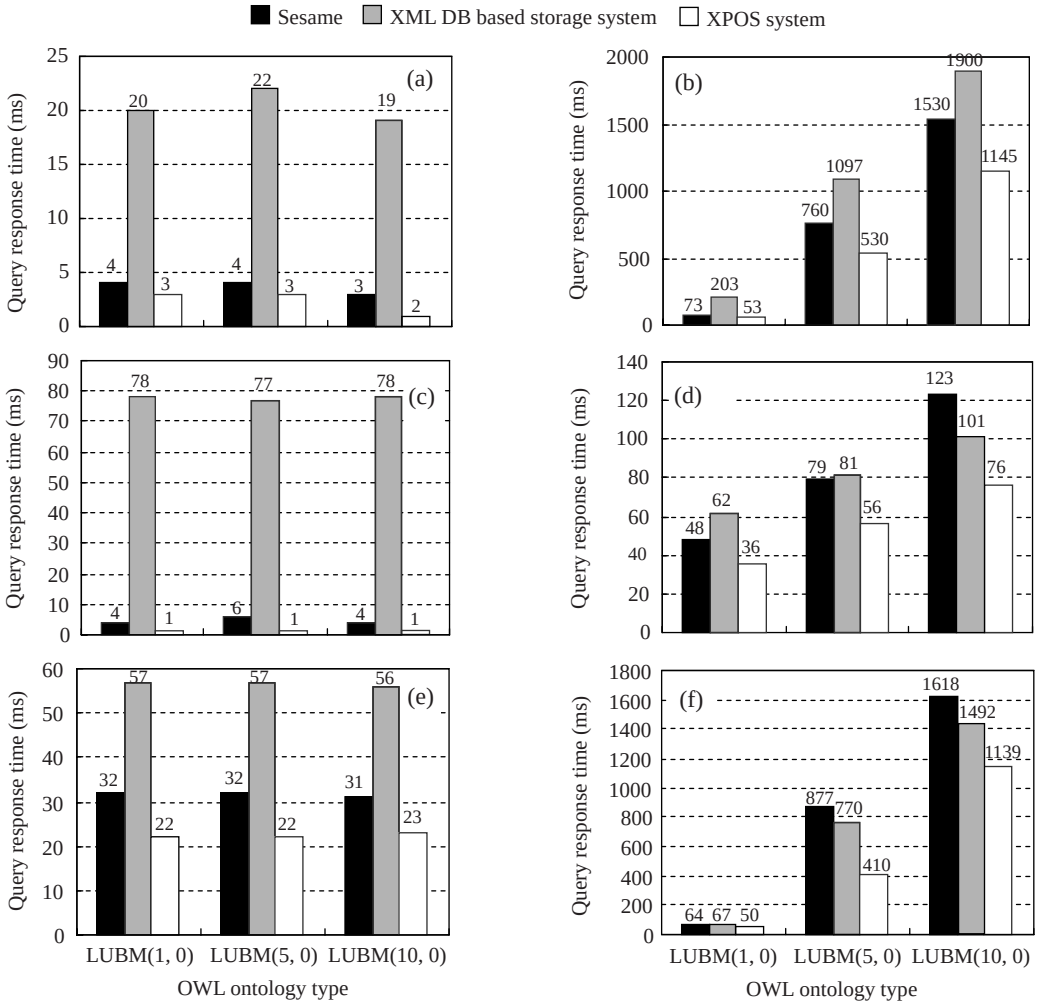


Fig.9 Query processing time for the three systems—Sesame, the XPOS system, and the XML DB based storage system—of six queries of LUBM(1, 0), LUBM(5, 0) and LUBM(10, 0)
(a) Query 1; (b) Query 2; (c) Query 3; (d) Query 4; (e) Query 5; (f) Query 6

the search processing time of sub-properties of the specified property. The XFSS system searches objects and subjects in one table, and shows the best performance in terms of searching objects and subjects related to the specific property. However, the computational cost for searching sub-properties is greater than that of Sesame and XPOS.

Fig.9f presents the processing results of the three systems for query 6. As the size of the OWL file increases, the numbers of instances in classes and properties also increase. In other words, the size of the OWL file is directly proportional to the query processing time for query 6. Sesame and the XPOS system search classes in the instance table based on the results of query 4. However, query processing results for query 6 of XPOS show better performance compared with those of Sesame, due to the difference in the time of searching sub-properties. The XML DB based storage model has a short search time of classes, because it searches classes in one table, based on the results of query 4; however, the time of searching sub-properties is much longer than that of the XPOS system. Therefore, the XPOS system shows the best performance, in terms of query processing for query 6.

The results of the aforementioned experiments for queries 1~6 prove that the performance of the XPOS model for searching hierarchical structural information between classes or properties is superior to that of Sesame and the XML DB based storage system. Sesame must perform an iterative search of sub-classes in the subClassOf table and subPropertyOf table. The XML DB based storage system

accesses the XML storage and always performs an XPath query for extracting hierarchical structural information. In addition, the XML DB based storage system has to access RDBMS and search ontology information based on the extracted hierarchical structural information. However, the XPOS system searches only values of the path attribute in the class and property tables to extract hierarchical structural information. Therefore, in terms of query processing performance with respect to hierarchical structure, the XPOS system always shows better performance than the other two systems.

Ontology updating time

Ontology data in a Web environment can be frequently modified and removed. Hierarchical structural information or instance values in the ontology can be modified. If instance values are changed, modification of the ontology is very simple. We can find the specified instance, and change the value in the storage. However, if the hierarchical structure is changed, ontology reconstruction and updating require much more time. If the hierarchical structure of the ontology is changed, as shown in Fig.10, each system must reconstruct hierarchical structural information in storage. For a performance evaluation of the ontology updating time, we assumed the ontology updating scenario shown in Fig.10.

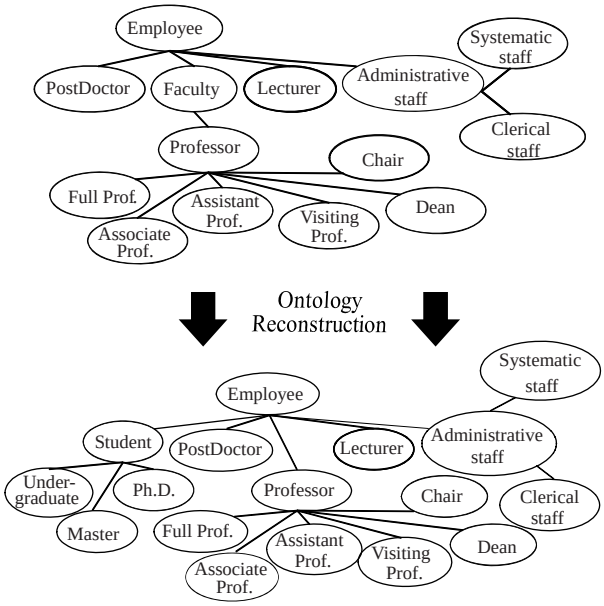


Fig.10 Ontology updating scenario

As in Fig.8, there are seven ontologies in the UBA ontology data. However, part of the ontology is modified, as shown in Fig.11. In this case, we do not need to completely reconstruct all the ontologies, and reload all the data; we do not need to reconstruct the remaining six ontologies. The performance evaluation of the ontology updating time is performed with the UBA ontology updating shown in Fig.8 and the ontology updating scenario shown in Fig.10. Fig.11 illustrates the ontology updating time of three OWL storage systems with the aforementioned ontology data and updating scenario.

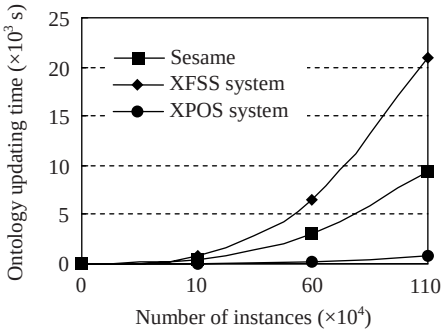


Fig.11 Ontology updating time

The ontology updating time means the ontology reconstruction time and storage time for each storage model. In Sesame, the ontology updating time is the same as the ontology loading time of the entire ontology. Sesame cannot partially reconstruct the ontology; it loads all ontology data and re-constructs the hierarchical structure. The XFSS system modifies the XML file system with the modified hierarchical structure and reconstructs the relational storage with changed values. The reconstruction time of the hierarchical structural information in the XFSS system is also the same as the initial construction time of the hierarchical structural information. However, the XFSS system can rapidly present modified values easily, because the XFSS system stores values about classes and properties in each ontology table. The XPOS system can update and partially reconstruct the ontology, because the XPOS system manages the root node of every node, classes and properties. The class and property tables in the XPOS model include the root_id attribute for managing the root nodes in each ontology. By managing the root nodes, the XPOS system can rapidly update and reconstruct the ontology

in a short time. As shown in Fig.11, the XPOS system shows 90%~92% better performance than Sesame, and 95%~97% better performance than the XFSS system, in terms of ontology updating time.

Except for query processing and ontology updating, the loading time of the ontology is also an important factor for evaluating the translation system. Obviously, an XPOS system consumes more loading time of ontology data than conventional systems, because it includes a path creation step additionally for more efficient query processing and ontology management. However, the major contribution of this study is to present a method that can provide more precise and effective query processing and ontology management than the methods proposed before. In addition, effective query processing and management of stored ontology data is more important than loading speed in the Semantic Web environment. Therefore, we do not focus on loading time complexity of an XPOS system in this work.

CONCLUSION AND FUTURE WORK

As the volume of Web information increases rapidly, extraction of precise information becomes the principal issue. In such an environment, the Semantic Web emerged, for assigning semantics to information and defining Web data formally. In addition, ontology description languages such as RDF, RDFS, and OWL were developed and utilized. In an ontology, hierarchical structural information between classes and properties is a critical factor. Therefore, we need an effective storage method for OWL data, which considers hierarchical structural information for precise extraction of information in the Semantic Web.

In this paper we described an XPOS model that considers hierarchical structure for effective and accurate extraction of information. In addition, we illustrated the structure of the XPOS system for the translation and storage of OWL data. An XPOS system analyzes a data schema of inputted OWL data and creates a data graph with hierarchical structural information between classes and properties. Also, an XPOS system extracts paths from the root class/property to all classes/properties via a depth-first search method. Extracted hierarchical structural information is stored in a path attribute in the class and property tables of the XPOS model.

Therefore, we can overcome the limitations of Sesame and the XML file system based storage system using the XPOS model proposed in this paper. Sesame and the XML file system based storage system are inefficient and ineffective in terms of query processing. Sesame requires unnecessary iterations for extraction of hierarchical structural information. In the case of the XML file system based storage system, it needs twice the number of accesses and two kinds of queries for RDB and the XML file system in every query processing. However, the XPOS model shows effective query processing performance, and enables intuitive and fast information extraction via XPath-based storage of a hierarchical structure.

In future studies, we need to consider the trade-off between storage efficiency and query processing time for hierarchical structural information. In general, if the storage efficiency is good, the query processing time is long, because it does not consider hierarchical structure in detail. Conversely, if the query processing time is short, with detailed consideration about hierarchical structure of an OWL document, the loading time is long, because this system has complex pre-processing steps. Therefore, we must research both of these cases, with consideration of the trade-off between storage efficiency and query processing time.

References

Beckett, D., 2004. RDF/XML Syntax Specification. W3C Recommendation. Available from http://www.w3.org/TR/rdf-syntax-grammar/

Broekstra, J., Kampman, A., van Harmelen, F., 2002. Sesame: an architecture for storing and querying RDF data and schema information. *LNCS*, **2342**:54-68.

Carroll, J.J., 2001. CoParsing of RDF & XML. HP Labs Technical Report, HPL-2001-292.

Carroll, J.J., de Roo, J., 2004. OWL Test Cases. W3C. Http://www.w3.org/TR/owl-test/

Carroll, J.J., Stickler, P., 2004. RDF Triples in XML. HP Labs Technical Reports, HPL-2003-268.

Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K., 2004. Jena: Implementing the Semantic Web Recommendations. Int. World Wide Web Conf., p.74-83.

Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Erdmann, M., Horrocks, I., 2000. The Semantic Web: the roles of XML and RDF. *IEEE Internet Comput.*, **4**(5):63-73. [doi:10.1109/4236.877487]

Fahmi, I., Zhang, J., Ellermann, H., Bouma, G., 2007. SWHi system description: a case study in information retrieval,

inference, and visualization in the Semantic Web. *LNCS*, **4519**:769-778.

Grant, J., Beckett, D., 2004. RDF Test Cases. W3C. Http://www.w3.org/TR/rdf-testcases/

Guo, Y.B., Pan, Z.X., Heflin, J., 2005. LUBM: a benchmark for OWL knowledge base systems. *J. Web Semant.*, **3**(2):158-182.

Harris, S., Gibbins, N., 2003. 3store: Efficient Bulk RDF Storage. PSSS, p.1-15.

Herman, I., Swick, R., Brickley, D., 2004. Resource Description Framework (RDF). W3C. Http://www.w3.org/RDF/

Jang, H., Kim, Y., Shin, D., 1999. An Effective Mechanism for Index Update in Structured Documents. Proc. 8th Int. Conf. on Information and Knowledge Management, p.383-390. [doi:10.1145/319950.320031]

Jeon, H., Kim, J., Jun, J., Kim, J., Im, D., Kim, H.J., 2005. RDF and OWL storage and query processing based on relational database. *KIISE J. Comput. Pract.*, **11**(5):451-457.

Kobayashi, K., Liang, W.X., Kobayashi, D., Watanabe, A., Yokota, H., 2005. VLEI Code: An Efficient Labeling Method for Handling XML Documents in an RDB. ICDE, p.386-387.

Koffina, I., Serfiotis, G., Christophides, V., Tanen, V., Deutsch, A., 2005. Integrating XML Data Sources Using RDF/S Schemas: The ICS-FORTH Semantic Web Integration Middleware (SWIM). Deutsch Dagstuhl Seminar: Semantic Interoperability and Integration, p.1-6.

Lausen, G., Meier, I., Schmidt, M., 2008. SPARQLing Constraints for RDF. Proc. 11th Int. Conf. on Extending Database Technology Advances in Database Technology, p.499-509. [doi:10.1145/1352431.1352492]

Lee, T.B., 2000. Primer: Getting into RDF & Semantic Web Using N3. W3C. Http://www.w3.org/2000/10/swap/Primer.html

Lee, T.B., Hendler, J., Lassila, O., 2001. The Semantic Web. Scientific American, New York.

Li, Z., Wang, Y.Z., 2006. An approach for XML inference control based on RDF. *LNCS*, **4080**:338-347.

McKenzie, C., Preece, A., Gray, P., 2006. Implementing a Semantic Web blackboard system using Jena. *LNCS*, **4187**:204-218.

Miller, L., Seaborne, A., Reggiori, A., 2002. Three implementations of SquishQL, a simple RDF query language. *LNCS*, **2342**:423-435.

Min, J.K., Ahn, J.Y., Chung, C.W., 2003. Efficient extraction of schemas for XML documents. *Inf. Processing Lett.*, **85**(1):7-12. [doi:10.1016/S0020-0190(02)00345-9]

Pan, Z.X., 2008. HAWK: OWL Repository and Toolkit. Lehigh University, Bethlehem. Available from http://swat.cse.lehigh.edu/downloads/index.html#hawk

Park, M.J., Lee, J., Lee, C.H., Lin, J.X., Serres, O., Chung, C.W., 2007. An Efficient and Scalable Management of Ontology. DASFFA, p.975-980.

Riddoch, A., Gibbis, N., Harris, S., 2002. 3Store.SourceForge.NET. Http://sourceforge.net/projects/threestore

Smith, M., Welty, C., McGuinness, D., 2004. OWL Web Ontology Language Guide. W3C Recommendation. Http://www.w3c.org/Tr/2004/REC-owl-guide-20040210/

Stuckenschmidt, H., van Harmelen, F., de Waard, A., Scerri, T., Bhoal, R., van Buel, J., Fluit, C., Kampman, A., Broekstra, J., van Mulligen, E., 2004. Exploring large document repositories with RDF technology: the DOPE project. *IEEE Intell. Syst.*, **19**(3):34-40. [doi:10.1109/MIS.2004.9]

Woo, E.M., Park, M.J., Chu, C.W., 2007. An efficient storage schema construction and retrieval technique for querying OWL data. *KIISE J. Database*, **34**(3):206-216.

Zhou, J.T., Wang, M.W., Zhang, S.S., Sun, H.W., 2006. Semi-structure Data Management by Bi-directional Integration between XML and RDB. CSCWD, p.1077-1081.