Journal of Computer Science 8 (3): 316-322, 2012
ISSN 1549-3636
© 2012 Science Publications

Effective Hierarchical Information Management in Mobile Environment

Jinhyung Kim, Myunggwon Hwang and Hanmin Jung
Department of Information Technology Research,
Korea Institute of Science and Technology Information,
52-11, Eoeun dong, Yuseong gu, Daejeon, Republic of Korea

Abstract: Problem statement: As performance of mobile devices is developed highly, several kinds of data is stored in mobile devices. For effective data management and information retrieval, some researches to apply ontology concept to mobile devices are progressed. However, in conventional researches, they apply conventional ontology storage structure used in PC environment to mobile platform. **Conclusion/Recommendations:** Therefore, performance of search about data is low and not effective. Therefore, we suggested new ontology storage schema with ontology path for effective hierarchical information in mobile environment.

Key words: Ontology path, hierarchical information, mobile environment, information retrieval

INTRODUCTION

As a specification (ex. CPU speed, storage capacity, memory capacity) of mobile phone is getting higher revolutionarily as Fig. 1, current mobile phones manage various kinds of data in the phone such as contacts, calendar, email, schedule, multimedia (ex. image, music, video), application lists. In 20 years, mobile processor was progressed from 1MHz to 1GHz and the storage capacity was developed more than 100 times. However, precise searching of requested information is becoming more difficult since there is a considerable increase in the volume of data in mobile phones. Precise information retrieval of the requested information is more important than just fast search of the data (Herman, 2003; Decker *et al*., 2000; Shadbolt *et al*., 2006).

For overcoming above limitations, there are many researches regarding ontology modeling and ontology construction in the mobile platform (D'Aquin *et al*., 2008, Carroll *et al*., 2004; Carroll and Stickler, 2004; Beckett and McBride, 2004). By the ontology modeling and construction in the mobile platform, we can manage mass data effectively and support rich information retrieval for users. However, because mobile devices have lower processing capacity than PC and laptop, effective information retrieval or information recommendation services based on the ontology cannot be provided in performance aspect until now. The most important thing in the information retrieval based on ontology is fast and precise information search about hierarchical structured information.

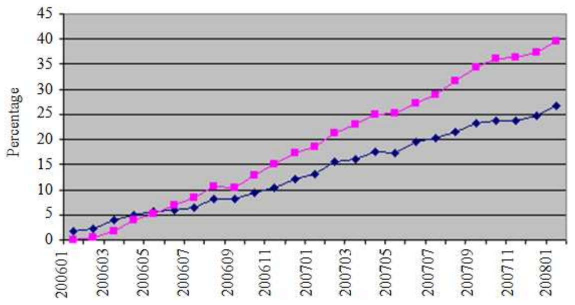

Fig. 1: Mobile multi-core processor progress

Therefore, in this study, we suggest new ontology storage schema for effective information retrieval about ontology data in the mobile platform. Additionally, we perform a comparative evaluation regarding hierarchical information retrieval.

MOntoPath: Basically, the MOntoPath model is based on relational database as ontology storage (Koffina *et al*., 2005; Lausen *et al*., 2008; Zhuan and Yuanzhen, 2006; McKenzie *et al*., 2006; Park *et al*., 2007; Liu and Li, 2008). The MOntoPath model consists of a class table, a property table, a triples table and an instance table. The storage schema of the OM-HI 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

Corresponding Author: Jinhyung Kim, Department of Information Technology Research,
Korea Institute of Science and Technology Information, 52-11, Eoeun Dong,
Yuseong Gu, Daejeon, Republic of Korea Tel: +82-42-869-1699 Fax: +82-42-869-0077

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 path 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 inste_id can be stored. In a predicate attribute, values in prop_id can be stored.

MOntoPath model schema:

- T = {Class, Property, Instance, Triples}
- A_{class} = {class_id, class_name, class_path, root_id}
- $A_{property}$ = {property_id, property_name, property_path, root_id}
- $A_{instance}$ = {inst_id, inst_name, class}
- $A_{triples}$ = {subject, predicate, object}
- $R_{(Class, Instnace)}$ = {instantiation}, $R_{(Class, Triples)}$ = {Subject || Object}
- $R_{(Property, Triples)}$ = {Predicate}, $R_{(Instance, Triples)}$ = {Subject || Object}
- $C_{(Class, Triples)}$ = {*,*}, $C_{(Class, Instance)}$ = {1,*},
- $C_{(Property, Triples)}$ = {1.*}, $C_{(Instance, Triples)}$ = {*,*}

Above formal description represents the storage schema for the MOntoPath model. T describes table lists and A illustrates attributes lists in a specific table. R means a relation name between two tables and C represents cardinality information between two tables.

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; Hepp, 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. 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: $N_{(name)}=(N_a, N_u, V_f, C_f, S_f)$, 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_nod e.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. 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.

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.

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.

Performance test:

Ontology modeling for test dataset: The ontology includes relationship and hierarchical information among data sources applications such as contacts, SMS, MMS. email, music. picture, video. The ontology is focusing on the definition about classes in the ontology. The data source class is based on various kinds of applications in the smart mobile platform and can be divided four typical sub data source classes: the key data sources class, the message data sources class, the multimedia data sources class and the web data sources class. The key data sources class includes indispensable instances such as contacts data and application data. The data sources in the key data source class affect many other data sources. The contacts data affects information stored in the SMS, MMS, Email and the application data affects all of application information in the smart mobile platform. If any of data in the key data sources class is changed, the data affected by those data must be modified. As shown in the Fig. 2, the contacts data and the application data included in the key data sources class have many 'DS: Affected By' relationships with many instances included in other classes. The dataset for the experiment is contacts, SMS, MMS, music, image, video, schedule data. We perform experiment from 1,000-10,000 data (Shoaib adn Basharat, 2010; Stuckenschmidt *et al*., 2004; Uanhui *et al*., 2009; Valencia-Garcia and Garcia-Sanchez, 2011).

Test SPARQL queries: We use 3 queries for performance experiment. Every query is related to hierarchical information retrieval. Table 1 represents SPARQL representations used for comparative performance. Query 1 is a query for search all of sub-classes of the MobData class and Query 2 searches all of sub-classes of the Mobdata class and all instances included in each subclasss. Query 3 is a query for search all of instances as object when all instances of the Mobdata and subclasses are subject. As described before, all queries in performance evaluation are related to search about hierarchical structure between classes. Therefore, we execute experiment focusing on comparative evaluation regarding the query processing performance difference among storage systems which have difference storage structure about hierarchical structure information.

Performance evaluation: In this study, we execute comparative performance evaluation with 3 storage structure: Sesame structure (Broekstra and Harmelen, 2001), Dual (Data Info. + Hierarchical Info.) structure (Woo *et al*., 2008) and suggested structure.

Sesame structure stores hierarchical information into relational database as additional table. The table consists of super-class and sub-class attributes. In this case, searching all of hierarchical information needs many repeat operations. Dual structure stores class, instance information and hierarchical information of class and property separately. That is, class and instance information is stored in relational database and the hierarchical information is stored in an additional XML file. In Dual structure, for a query processing, we need access the relational database and XML file. Additionally, user queries must be converted SQL and XQuery format. Results from two databases also have to be merged. However, because the suggested system contains hierarchical information as a Path form, the system can recognize hierarchical location and information of a specific class easily.

Though the size of the dataset increases, the number of classes remains constant. The experimental results of query 1 are unrelated to the size of the test dataset. To process query 1, in Sesame structure we perform an iterative search of sub-classes, to search sub-classes of the MobData class. In the dual structure, we access the XML file acquire hierarchical structural information. Then, we search class information in the relational database. However, in the case of the MOntoPath structure, we can search sub-classes by searching the path attribute in the class table. Therefore, the suggested system shows the best performance in terms of the processing time of query 1, because of searching only one table without any join operation, for performing query 1.

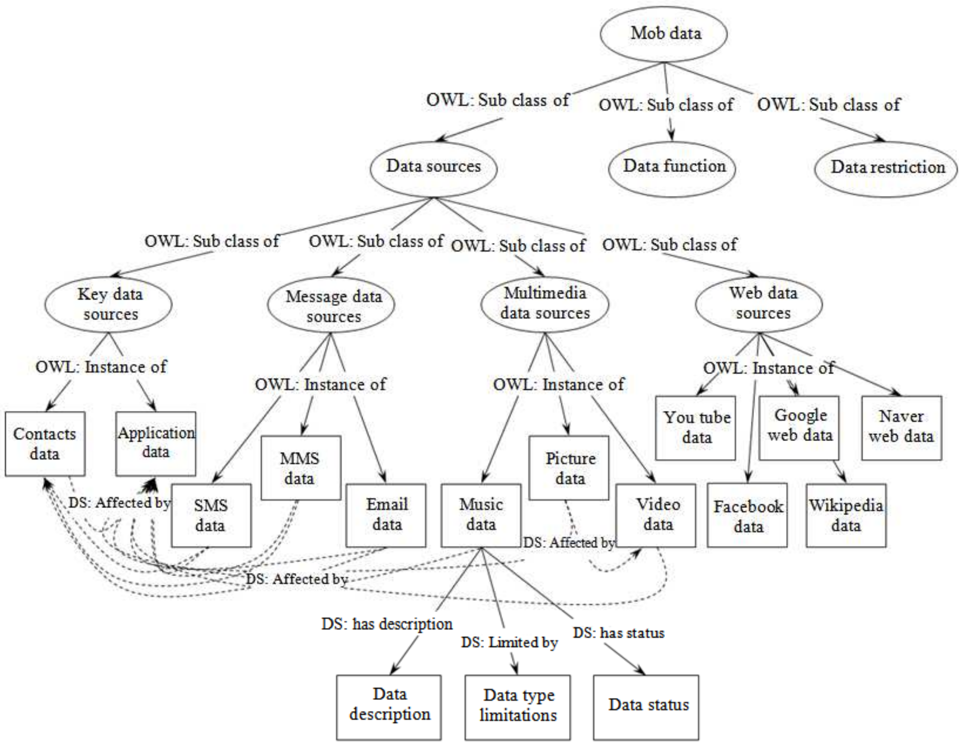

Fig. 2: Test dataset ontology

Table 1: SPARQL queries for experiment

	SPARQL query
Query 1	select ?class where { {?class MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?x. ?x MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?y ?y MO::http://www.w3.org/2000/01/rdf-schema#subClassOf MO::http://software.korea.ac.kr/koolmania/montopath.owl#MobData }union all {?class MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?y ?y MO::http://www.w3.org/2000/01/rdf-schema#subClassOf MO::http://software.korea.ac.kr/koolmania/montopath.owl#MobData }union all {?class MO::http://www.w3.org/2000/01/rdf-schema#subClassOf MO::http://software.korea.ac.kr/koolmania/montopath.owl#MobData } Query for all of sub-classes of the MobData class

Query 2	select ?instance
	where {
	?instance MO::http://www.w3.org/2000/01/rdf-schema#instanceOf ?class {
	{?class MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?x.
	?x MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?y
	?y MO::http://www.w3.org/2000/01/rdf-schema#subClassOf
	MO::http://software.korea.ac.kr/koolmania/montopath.owl#MobData } union all
	{?class MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?y
	?y MO::http://www.w3.org/2000/01/rdf-schema#subClassOf
	MO::http://software.korea.ac.kr/koolmania/montopath.owl#MobData} union all
	{?class MO::http://www.w3.org/2000/01/rdf-schema#subClassOf
	MO::http://software.korea.ac.kr/koolmania/montopath.owl#MobData }
	Query for instances in MobData and sub-classes of MobData

Query 3	select ?x
	where {
	?x ?y ?instance {
	?instance MO::http://www.w3.org/2000/01/rdf-schema#instanceOf ?class {
	{?class MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?x.

Table 1: Continuous

	?x MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?y ?y MO::http://www.w3.org/2000/01/rdf-schema#subClassOf MO::http://software.korea.ac.kr/koolmania/montopath.owl#MobData} union all {?class MO::http://www.w3.org/2000/01/rdf-schema#subClassOf ?y ?y MO::http://www.w3.org/2000/01/rdf-schema#subClassOf MO::http://software.korea.ac.kr/koolmania/montopath.owl#MobData } Query for instance as a subject when instances in MobData and sub-classes of MobData is object

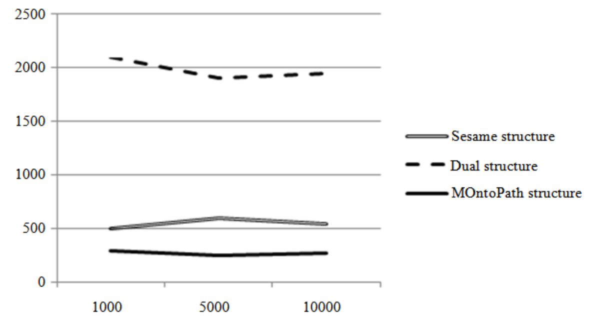

Fig. 3: Test result of query 1

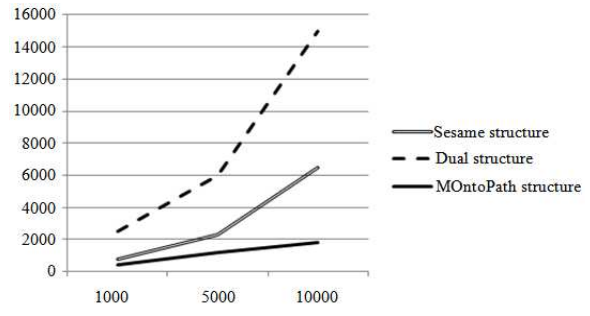

Fig. 4: Test result of query 2

As the size of the dataset increases, the number of instances also increases; if the size of the dataset increases, the query response time for query 2 also increases. In the Sesame structure and the MOntoPath structure, we retrieve instances by searching the instanceOf/instance table, based on the results of query 1. However, as shown Fig. 3, the system requires more iteration for searching sub-classes in Sesame structure than the MOntoPath structure. Therefore, query processing performance for query 2 of the MOntoPath structure is better than that of Sesame structure as Fig. 4. However, in the dual structure, the system requires many more join operations because instances are stored in many classes and property tables.

In case of query 3, we need execute query 2 and additional operation for search triple data. Triple information in stored separately in all of structure. Therefore, result of query 3 is almost similar to that of query 2. Fig. 5 shows the result of query 3.

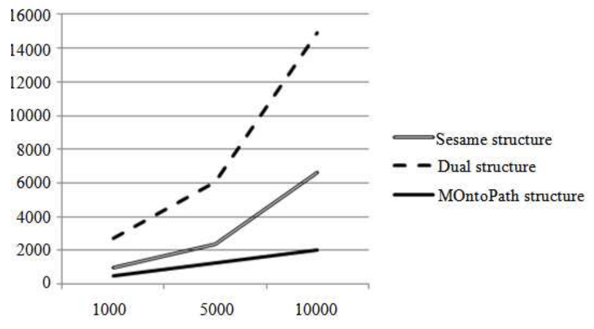

Fig. 5: Test result of query 3

The results of the aforementioned experiments for queries 1~3 prove that the performance of the MOntoPath model for searching hierarchical structural information is superior to that of Sesame and the dual structure. Sesame structure must perform an iterative search of sub-classes in the subClassOf table. The dual structure accesses the XML file and always performs an XPath query for extracting hierarchical structural information. In addition, the dual structure has to access RDBMS and search ontology information based on the extracted hierarchical structural information. However, in the MOntoPath structure 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 MOntoPath model always shows better performance than the other two systems.

CONCLUSION

In this study, we suggested the new ontology storage model for effective hierarchical information retrieval in mobile platform. In mobile platform, because fast and precise information retrieval is the most important factor, the suggested model is focusing on those points. Additionally, we performed the comparative performance evaluation among diverse ontology storage structures such as Sesame structure and dual structure. As shown in the performance test, the suggested MOntoPath model shows the best performance regarding hierarchical information processing.

For 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 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. and B. McBride, 2004. RDF/XML syntax specification. World Wide Web Consortium.

Broekstra, J. and F.V. Harmelen, 2001. Sesame: An architecture for storing and querying RDF data and schema information. Comput. Inform. Sci., 2342: 1-16.

Carroll, J.J. and P. Stickler, 2004. RDF triples in XML. Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters, May 17-22, ACM, New York, USA., pp: 412-413. DOI: 10.1145/1013367.1013501

Carroll, J.J., I. Dickinson, C. Dollin, D. Reynolds and A. Seaborne *et al*., 2004. Jena: Implementing the semantic web recommendations. Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters, May 17-22, ACM, New York, USA., pp: 74-83. DOI: 10.1145/1013367.1013381

D'Aquin, M., E. Motta, M. Sabou, S. Angeletou and L. Gridinoc *et al*., 2008. Toward a new generation of semantic web applications. IEEE Intell. Syst., 23: 20-28. DOI: 10.1109/MIS.2008.54

Decker, S., S. Melnik, F.V. Hermelen, D. Fensel and M. Klein *et al*., 2000. The semantic web: The Roles of XML and RDF. IEEE Internet Comput., 4: 63-73. DOI: 10.1109/4236.877487

Hepp, M., 2006. Semantic web and semantic web services: Father and son or indivisible twins? IEEE Internet Comput., 10: 85-88. DOI: 10.1109/MIC.2006.42

Herman, I., 2003. Introduction to the semantic web. W3C.

Jang, H., Y. Kim and D. Shin, 1999. An effective mechanism for index update in structured documents. Proceedings of the 8th International Conference on Information and Knowledge Management, Nov. 02-06, ACM, USA., pp: 383-390. DOI: 10.1145/319950.320031

Kobayashi, K., W. Wenxin, D. Kobayashi, A. Watanabe and H. Yokota, 2005. VLEI code: An efficient labeling method for handling XML documents in an RDB. Proceedings of the 21st International Conference on Data Engineering, Apr. 05-08, IEEE Xplore Press, pp: 386-387. DOI: 10.1109/ICDE.2005.153

Koffina, I., G. Serfiotis, V. Christophides, V. Tannen and A. Deutsch, 2005. Integrating XML data sources using RDF/S schemas: The ICS-FORTH Semantic Web Integration Middleware (SWIM). Dagstuhl Seminar Semantic Interop. Integr.

Lausen, G., M. Meier and M. Schmidt, 2008. SPARQLing constraints for RDF. Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology, Mar. 25-30, ACM, Nantes, France, pp: 499-509. DOI: 10.1145/1353343.1353404

McKenzie, C., A. Preece and P. Gray, 2006. Implementing a semantic web blackboard system using jena. University of Aberdeen.

Park, M.J., J. Lee, C.H. Lee, J. Lin and O. Serres *et al*., 2007. An efficient and scalable management of ontology. Adv. Databas.: Concepts, Syst. Appli., 4443: 975-980. DOI: 10.1007/978-3-540-71703-4_88

Shadbolt, N., W. Hall and T. Berners-Lee, 2006. The semantic web revisited. IEEE Intell. Syst., 21: 96-101. DOI: 10.1109/MIS.2006.62

Shoaib, M. and A. Basharat, 2010. ERMOS: An efficient relational mapping for ontology storage. Proceedings of the IEEE International Conference on Advanced Management Science, Jul. 9-11, IEEE Xplore Press, Chengdu, pp: 399-403. DOI: 10.1109/ICAMS.2010.5553141

Stuckenschmidt, H., F.V. Harmelen, A.D. Waard, T. Scerri and R. Bhogal *et al*., 2004. Exploring large document repositories with RDF technology: The DOPE project. IEEE Intell. Syst., 19: 34-40. DOI: 10.1109/MIS.2004.9

Uanhui, L., Z.M. Ma and X. Zhang, 2009. Fuzzy ontology storage in fuzzy relational database. Proceedings of the 6th International Conference on Fuzzy Systems and Knowledge Discovery, Aug. 14-16, IEEE Xplore Press, Tianjin, pp: 242-246. DOI: 10.1109/FSKD.2009.701

Valencia-Garcia, R., F. Garcia-Sanchez, D. Castellanos-Nieves and J.T. Fernandez-Breis, 2011. OWLPath: An OWL ontology-guided query editor. IEEE Trans. Syst. Man Cybernetics, 41: 121-136. DOI: 10.1109/TSMCA.2010.2048029

Woo, E.M., M.J. Park and C.W. Chung, 2008. An efficient storage schema construction and retrieval technique for querying OWL data (2008). Korean Information Science Society.

Liu, Z. and H. Li, 2008. An ontology-based virtual storage system. Proceedings of the IEEE International Conference on Networking, Architecture and Storage, Jun. 12-14, IEEE Xplore Press, Chongqing, pp: 185-186. DOI: 10.1109/NAS.2008.47

Zhou, J., M. Wang, S. Zhang and H. Sun, 2006. Semi-structure Data Management by Bi-directional Integration between XML and RDB. Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design, May 3-5, IEEE Xplore Press, Nanjing, pp: 1-5. DOI: 10.1109/CSCWD.2006.253208

Zhuan, L. and W. Yuanzhen, 2006. An approach for XML inference control based on RDFDatabase Exp. Syst. Appli., 4080: 338-347. DOI: 10.1007/11827405_33