A new approach for storing RDF triples based on ontology modularization

Meisam Booshehri [1], **Kamran Zamanifar** [2] **and Shahdad Shariatmadari** [3]
[1,2] Department of Computer Engineering, Najafabad Branch, Islamic Azad University, Najafabad, Iran.
[3] Department of Computer Engineering, Shiraz Branch, Islamic Azad University, Shiraz, Iran.

Abstract *- Managing RDF data in an effective manner is one of significant factors in realizing semantic web vision. Currently, there are two general approaches for storing RDF data which are column-oriented and row-oriented and each one has advantages and disadvantages. These two approaches are used in relational DBMSs. After reviewing different approaches and methods for storing RDF triples, we put forward a suitable bridging idea of ontology modularization that makes RDF data management systems outperform. The proposed approach will divide an ontology into several modules in order to reduce the size of information sets that we are working with in a specific moment at run time. We propose two new storing methods (which are table per ontology module method and vertically partitioned module method). We will analyze the effects of applying these methods on standard RDF repositories and figure out some directions that should be considered in designing the RDF repositories.*

Keywords- RDF repository, Ontology Modularization, RDF Triples, Column-oriented Approach, Row-oriented Approach.

1. Introduction

The content of annotation consists of some rich semantic information. These annotations are consumed both by human agents and software agents [1, 2, and 3]. Tim Burners Lee, the inventor of web, has presented a data model called "RDF Model" used for recourse annotation [1, 2, and 3]. This model covers three concepts including: resources, properties and statements. A resource is a thing you talk about it (can reference it).

[1]He is a Master Student in department of Computer Engineering, Najaf Abad Branch, Islamic Azad University, Najaf Abad, Iran. (email: m_booshehri@sco.iaun.ac.ir)

[2]He is an assistant professor in department of computer engineering, Najafabaad branch, Islamic Azad University, Najafabad, Iran. (email: Zamanifar@eng.ui.ac.ir)

[3]He is faculty member in department of computer engineering, shiraz branch, Islamic Azad University, Shiraz Iran. Also, he is PhD candidate in department of computer science and information technology, University Putra Malaysia. (email: shariatmadari@iaushiraz.ac.ir)

Every resource has a URI. RDF definitions are itself Resources [1, 2, and 3]. Properties, defines relationship to other resources or atomic values. Statements consist of resources, properties and values [3, 4]. Values can be resources or atomic XML data [3, 4].Using RDF Model is a common method for resource annotation. Using relational databases is one of common techniques for storing annotations. To date two types of DBMSs (row- oriented and column-oriented) have been constructed [10, 11, 12] for this purpose. And each type uses various approaches to form data tables.

In this paper, we propose two new storing methods based on the idea of ontology modularization. Ontology modularization [18] is used in ontology engineering as a way to structure ontologies, meaning that the construction of a large ontology should be based on the combination of self-contained, independent and reusable knowledge components.
In this paper we propose a new approach for storing RDF triples in relational databases based on ontology modularization.

The remainder of paper is organized as follows. In section 2 we talk about related work in which we review the most important existing storage methods. In section 3 we propose our new storage methods. In section 4 we discuss the new methods and compare them with corresponding methods. And in section 5 we conclude and talk about future work.

2. Related Work

It is important to select the best solution for storing and retrieving RDF information. [13] classifies state-of-the-art RDF storage and indexing schemes in two subcategories. First subcategory includes relational schemes that use RDBMSs for storing RDF data and the second subcategory includes native schemes that build RDF-specific stores and indexes from scratch. Although native schemes perform well because of their suitable design, relational schemes are preferred to use as a solution for RDF data management and this is because of the maturity, generality and scalability of relational databases [13]. In this context we must put forward a question. "How the table design should be for storing RDF triples?" By now some

standard storage methods for row-oriented database systems has been proposed including Horizontal Table [5], Vertical Table [7, 10], Horizontal Class [8], Table per Property [8] and Hybrid approach [8]. Also with the birth of column-oriented database systems some other techniques have been proposed including vertically partitioned method [14] and sextuple indexing technique [13 , 15] etc. In the following sections we survey these methods.

2.1. Row-oriented database systems storage methods

Here we introduce storage methods which have been used in row-oriented RDBMSs and discuss the benefits and drawbacks of these methods.

2.1.1. Horizontal Table method

This storing method uses a universal table in the database for all existing ontologies [5]. Fields of this table includes instances' IDs, instance types and values of instance properties. You can see a representation [8] for horizontal table in Fig. 1. The domain of all columns in the table except the column "type" could be both URIs and literals.

Instance's ID	Type	Property_1	Property_2	…	Property_n
…#1	Class_A	Value_a	Value_b	…	Value_c
..#2	Class_B	Value_d	Value_e	…	Value_f

Figure 1. A representation for horizontal table

The advantages of this method come below:

1- Subject and predicate are stored once [5, 8].

2- Short query run time [5, 8].

3- Its structure is simple because every instance has one entry in the table [5, 8].

On the other hand we can mention five cases as disadvantages of this method which come below:

1- Abundance of columns: considering the size of knowledge-base and number of properties, we can say that the relational database system would be limited with number of columns [5, 8].

2- Limitation of property values (lack of support in the case of multi-valued properties): the said structure forces every property to have only one value; however, a large number of properties are naturally multi-valued [8].

3- Sparseness: it is obvious that every property has a corresponding field in the table despite the fact that many records might have no values for filling fields. Therefore many fields in the database may have null values [5, 8].

4- Difficulty of maintenance: whenever a new ontology is being incorporated into a system or a system changes an existing ontology, the said table needs to be reorganized. If the table is huge, such changes could be so expensive [5, 8].

5- Decrease in performance: this approach results in a large-size database. Also load time in this approach is longer in comparison to other ones [6, 8].

Another type of this approach is using a table for each ontology. It is obvious that this solution decreases the number of columns and cost of changes in every table, however, in the case of this type of horizontal table still limitations of properties exists and tables in this method have larger number of columns in comparison to other methods and there exists sparseness problem yet.

2.1.2. Vertical Table Method

In this method, we have a universal table [7]. The table has only three fields including subject, predicate and object. In predicate field we can store both property names and the string "type" which indicates that the current record defines the type of a resource. In subject field we can store the URI of resources and in object field we can store class names and values of properties. You can see a representation [8] for vertical table in Figure 2. Some benefits of this method are its simple structure, ease of maintenance and fixed number of columns. This method is the most straightforward relational method [13].

Predicate	Subject	Object
Type	…#1	Class_A
Property_1	…#1	Value_a
Property_2	…#1	Value_b
…	…	…

Figure 2. A representation for vertical table

It is very general that is every type of data could be stored in this format [13]. Also some disadvantages of this method are as follows:

1- High query run time [7, 8].

2- As mentioned in [8] "This design means that any query has to search the whole database and queries that involve joins will be especially expensive. In particular, queries about the members of a class will be particularly difficult, because there is no explicit treatment of the class hierarchy".

3-we have to save values in string format.

2.1.3. Horizontal class method

This method is similar to horizontal table method but with smaller tables. In this method we have a separable table for every class in ontology. A representation [8] for horizontal class table has been shown in Figure 3.

ID	Property_1	Property_2	…	Property_n
…#1	Value_a	Value_b	…	Value_c

Figure 3. A representation for horizontal class

The advantages of this method are as follows:

1- Less sparseness in comparison to horizontal table method [8].

2- the most important benefit of this method is that the query on the property of an instance or a set of instances is done effectively [8].

3- Lower query run time in comparison to horizontal table method [8].

One of the disadvantages of this method is that the properties which have no explicit domain should be repeated in all the tables [8].

2.1.4. Table per property method

This method was first used in row-oriented RDBMSs but it is more suitable for column-oriented RDBMSs**.** In this method every property has a corresponding table [5]. A representation of such tables has been shown in Figure 5. Instances of all classes are saved in a table named "Type" in which every record relates an instance to a class.

Subject	Object
…#1	Class_A
…#1	Class_B

Figure 4. A representation for the table "Type"

Subject	Object
…#1	Value_a
…#2	Value_d

Figure 5. A representation for the table of a property

A representation[8] for the table "Type" has been shown in Figure 4. Some advantages of this method are short time for simple queries and decrease in size of tables [8] .Some disadvantages of this method includes increase in number of tables and high run time for complicated queries due to lots of join operations that must be done [8].

2.1.5 Hybrid method

This method combines different storing methods therefore in comparison to other methods performance increases. Also there is less limitations and totally less drawbacks in comparison to the state in which we use every storing method independently [9]. This method has been used in many data repositories [8].

2.2. Column-oriented database systems storage methods

Here we talk about methods which are naturally suitable for and have been used in column-oriented RDBMSs. We discuss the advantages and disadvantages of these methods as well.

2.2.1. Vertically partitioned method

Property tables method did not fit well with the semi-structured nature of RDF data and SPARQL queries with unbound variables in the property positions. Thus to solve such limitations Abadi et. al. [22] proposed the vertically partitioned method. This method is the same table per property method with the difference that Abadi et. al. have integrated this method with a column-oriented RDBMS and some side techniques such as data compression have been used to improve the performance of this technique. Some advantages of this method are:

1- Support for multi-valued attributes [22].

2- Support for heterogeneous records [22].

3- Only those properties accessed by a query need to be read [22].

4- No clustering algorithms are needed [22].

Also one of the disadvantages of this method is that "if the property in a query is bound to a variable, then the rows returned from each property table must be union-ed. In the case where the property is not part of the result, then the union operator must also perform a duplicate elimination. Finally, since the data is not clustered on objects, a query which joins on objects, will not allow the use of a fast (linear) merge join" [15, 21]. However Abadi et. al. in [22] have mentioned that "although the vertically partitioned approach require more joins relative to the property table approach, properties are joined using simple, fast (linear) merge joins".

2.2.2. Sextuple indexing technique

The idea of sextuple indexing has been proposed in [15] for the first time. This technique points to the fact that a RDF triple is a three-dimensional entity and could be indexed in six ways. Thus using this approach allows fast merge-joins for any pair of two triple patterns [13, 15]. In spite of this fact

neither Hexastore nor RDF-3X (as native schemes for storing RDF data) doesn't point to the ways that realize sextuple indexing technique in relational database systems[15, 23]. Therefore Xin Wang et al. in [13] propose a novel storage technique for RDF data in which sextuple indexing is applied to column-oriented RDBMSs. As mentioned in [13] "The experimental results shows that this method outperforms the row-oriented RDBMS approach by upto an order of magnitude, and is competitive to the best native RDF store RDF-3X". Also in our point of view applying sextuple indexing technique could be effective as we mention in the next sections.

3. Proposed approach

In our proposed method we try to reduce the size of information domain which we are working with in a specific moment at run time.

3.1. Ontology Modularization

Nowadays big size of existing ontologies is a crucial problem. Maintaining these ontologies which might have more than one hundred thousand concepts is difficult [16]. Also reusing the whole ontology is time-consuming and costly [17]. A solution to solve this problem is to divide ontology into some parts with a special subject. These parts are ontology modules and this process is called ontology modularization. There are various definitions for an ontology module. [18] defines ontology module as follows: "An ontology module is a reusable component of a larger or more complex ontology, which is self-contained but bears a definite association to other ontology modules, including the original ontology". Also as mentioned in [17] we say a module is self-contained while special reasoning tasks such as inclusion relation or query answering within a module are possible with no need to access other modules.

By now different approaches for ontology modularization have been proposed[24,25,26,27,28,29,30,31], including logic-based approaches[28,31] and Graph theory-based approaches[28] etc. We think that all these approaches could be useful in different places. But in this paper we don't use a specific approach or a special partitioning algorithm for the ontology modularization step in our proposed RDF storage approach.

The proposed approach in this paper will divide an ontology into several modules in order to reduce the size of information sets that we are working with in a specific moment at run time. So here we consider a two-column table named "Module_Class_Table" in which every record relates a class to the ontology module that the class belongs to. A representation of such table has been showed in the figure 6.

Class_Name	Module_ID
Class_A	…#10
Class_B	…#2

Figure 6. A representation for Module_Class_Table

In spite of previous storing methods which work on the whole ontology, our approach emphasizes on ontology modules as the base of designing database tables. Based on this new approach, we suggest two new methods w.r.t. the viewpoint of both row-oriented RDBMSs and column-oriented RDBMSs.

Before explaining these two methods we define two types of properties: intra-module properties and inter-module properties. Intra-module properties are those which are only related to concepts among an ontology module. And inter-module properties are those which connect couples of concepts from different modules. It is obvious that we may have both of these two types of properties within an ontology. Also with respect to this classification we can categorize queries into two subcategories which are *intra-module queries* and *inter-module queries*. An intra-module query work only on the information and concepts within a specific module (tables extracted from a specific ontology module), however, an inter-module query may work on the information and concepts within a specific ontology (especially among modules). Of course an inter-module query could be a combination of some intra-module queries and some inter-module queries.

Considering the classifications mentioned above, when a query is applied to a database system based on our approach, at first it must be recognized if the query is inter-module or intra-module. Such decision can be made by a simple query on the Module_Class_Table. Therefore for supporting our new approach, we think that a preprocessing unit should be embedded in RDF data repositories such as Jena and SW-store in which queries are analyzed to determine if a query is inter-module or intra-module.

3.2. Proposed method for row-oriented RDBMSs (Table per ontology method)

In this method every ontology module is stored into a separable table. Also in this method we use a universal vertical table which contains RDF triples that describe the inter-relations between concepts of ontology modules.

3.3. Proposed method for column-oriented RDBMSs (vertically partitioned module method)

Here we propose a new method named "vertically partitioned module method". In this method each intra-module property has a corresponding two-column property table. However,

inter-module properties are mentioned in a universal vertical table which contains RDF triples that describe the inter-relations between concepts of ontology modules. Also intra-module property tables could be indexed by sextuple indexing technique and combined with other side techniques such as data compression techniques.

4. Discussion

As mentioned in [13] the most straightforward relational method for storing RDF triples is vertical table method. This method is very general that is every existing data type could be stored in this format [13]. The basic problem of this method is the problem of expensive self-joins over this vertical triples table which is possibly large [13]. These self-joins is because of SPARQL queries with multiple triple patterns [13]. Here we quit this method because of its basic problem and won't compare it with other methods.

Here we introduce a useful factor named "Working Information Set" or WIS. WIS is the smallest subset of information in a domain to which access probability is more than other subsets of information in the same domain in different time intervals. We use this relative concept to compare the performance of our methods with the others.

Another concept that we need it in this discussion is data fragmentation which is a topic discussed in distributed databases. It is classified into three subcategories: horizontal fragmentation, vertical fragmentation and hybrid fragmentation [19, 20].

Our approach emphasizes on ontology modules as the database design basis. It is obvious that the number of extracted tables from an ontology module is less than the number of extracted tables from the whole ontology. Moreover, existing data in the tables of a module is less than existing data in the corresponding tables of the whole ontology. It means that focusing on modules instead of the whole ontology, results in decrease in the size of WIS. And this subject causes lower load time and more performance. In our point of view this is one of the benefits that ontology modularization brings to column-oriented databases as well as row-oriented databases.

If we consider horizontal table as a reference table, horizontal class tables and property tables are produced by hybrid fragmentation of horizontal table. Also using ontology modules (as database design basis) instead of considering whole ontology is similar to data fragmentation techniques which we call it logical data fragmentation.

In this paper, we consider module extraction as a new type of data fragmentation in the context of RDF database systems. The more effective algorithms for module extraction we use the more suitable logical data fragmentation we have. As a result we can say that ontology module tables are also produced by hybrid fragmentation of horizontal table. Also vertically partitioned module tables are produced by hybrid fragmentation of each ontology module table.

It seems that separating ontologies into modules is a justifiable data fragmentation. Increasing the degree of concurrency and system throughput are two important benefits of data fragmentation in distributed databases [19, 20]. Therefore module extraction and use of ontology module (as database design basis) would make us closer to these two benefits.

On the other hand there are two important disadvantages for data fragmentation as follows:

1-If we have some requirements which are in conflict with data fragmentation, the performance would decrease. For instance it is costly to retrieve several different parts that must be joined or unioned [20].

2-.During data fragmentation some attributes that is related to an association relationship may be separated into several parts and located in distinct sites. This will cause the problem of difficulty in semantic control of data and difficulty in integrity control as well [20].

According to the self-contained feature of an ontology module, we can say that the problems mentioned above are not serious about ontology modules.

Based on the above discussions, it is better to compare *horizontal table* method, *horizontal class* method and *table per ontology module* method with each other because these three methods are naturally suitable for row-oriented RDBMSs. On the other hand it is better to compare *table per property* method, *vertically partitioned* method and *vertically partitioned module* method with each other since these methods are naturally suitable for column-oriented RDBMSs.

4.1. Discussion about Table per ontology module method

Generally this method is a middle method between horizontal table method and horizontal class method. If the ontology has only one module, this method is equivalent to horizontal table method. Also if we consider every ontology class as a module, this storing method is equivalent to horizontal class method. But in our point of view, because of strong associative relationships that some classes might have with each other, it seems that mapping a class to a module is not suitable.

We consider three general states for ontologies existing in a domain and then we compare it with previous methods. Suppose we have extracted 'n' ontologies.

State 1- None of 'n' ontologies has the capability of modularization. In this state there are n tables for storing RDF triples and the new method is exactly similar to the type of horizontal approach in which every ontology is stored in a separate table. So the advantages and disadvantages of this method would be the same as horizontal table method.

State 2- the number of extracted modules from existing ontologies is abundant and the number of classes of the modules is few. In this state the advantages and disadvantages of the new method are very similar to the advantages and disadvantages of horizontal class method.

State 3 – The existing ontologies have the capability of being modularized and also the number of extracted modules is average. It seems that this state is the best state for this method. In our point of view such state could show itself in multifaceted ontologies (ontologies that cover different domains of information). Some features of this method are as follows:

- This method has less sparseness in comparison to horizontal table method; however, its sparseness is more than horizontal class method.
- The number of table columns in this method is fewer than the number of table columns in horizontal table method and more than the number of table columns in horizontal class method.
- Comparing to horizontal table method, this method has fewer number of tables but in comparison to horizontal class method it has larger number of tables.
- Maintenance process in this method is easier than horizontal table method. Surely it comes from benefits of modules in software engineering.
- Less load time in comparison to horizontal table method and longer load time comparing with horizontal class method.

Performance of this method is related to the ontologies of the application which we want to design a database for it. If we have large ontologies that have mostly dependant components, we can say that this storing method has a better performance in comparison to other previous storing methods. But if we have ontologies with coherent components and there is no capability for transforming them into many modules, we can say that this storing method won't have a good performance.

Generally this new method could be useful with respect to features mentioned above. Composing this new method with other storing methods seems to be useful too.

4.2. Discussion about Vertically partitioned module method

Some row-oriented RDBMSs use table per property method combining with other methods. Actually vertically partitioned method is the improved version of table per property method. Also if we compare vertically partitioned method with vertically partitioned module method, we can see that superiority of vertically partitioned module method comes from its smaller WIS. This fact leads to the lower size of tables, lower load time and possibly higher performance. Generally we think that vertically partitioned module method improves the vertically partitioned method from this aspect .Of course it is obvious that separating ontologies into modules in a defective manner decreases performance.

5. Conclusion and future work

In this paper we proposed a new approach for storing RDF triples in relational databases based on ontology modularization and we presented two new storing methods which are *table per ontology module method* and *vertically partitioned module method*. The *table per ontology module* method seems to be naturally suitable for row-oriented RDBMSs. In this study, we show that in some cases this method is an appropriate alternative to horizontal table method and horizontal class method.

The second method is *vertically partitioned module* method. This method can be integrated with some techniques such as sextuple indexing method [15] to get a better performance. This method is naturally suitable for column-oriented RDBMSs.

Our approach is based on ontology modularization in which we divide an ontology into several modules in order to reduce the size of WIS.WIS is the smallest subset of information in a domain to which access probability is more than other subsets of information in the same domain in different time intervals. In this paper we have compared the *table per ontology module method* with the existing methods which are naturally suitable for row-oriented RDBMSs. Also we have compared *vertically partitioned module method* with the existing methods which are naturally suitable for column-oriented RDBMSs. Totally our preliminary study shows that our new methods results in smaller WIS in comparison to other previous corresponding methods. It is obvious that Smaller WIS leads to the lower load time and totally higher performance.

In our point of view ontology modularization is a type of data fragmentation in the context of RDF database systems. Therefore our approach results in increasing degree of concurrency and system throughput in distributed environments. Also according to the self-contained feature of an ontology module, we can say that the problems [20] of data fragmentation are not serious about ontology modules and totally our storing approach.

As future work we are going to implement our methods which have been explained in this paper. This is related to designing a new model for RDF data repositories which enables them to recognize modules in a specified ontology. It seems that it is better to consider a new unit in a data repository which pursues module extraction smartly and automatically. For designing this model the structure of data repositories must be examined precisely.

Also a preprocessing unit for analyzing queries should be embedded into data repositories to determine if a query is inter-module or intra-module. Therefore we must design this unit.

Moreover, we are planning to select and extend a suitable partitioning algorithm among existing ones [28] that modularize ontologies in an efficient way.

6. References

[1] The Semantic Web, Scientific American, May 2001, Tim Berners-Lee, James Hendler and Ora Lassila

[2] Tim Berners-Lee, Eric Miller, Jim Hendler, "Integrating applications on the Semantic Web", Journal IEEE Japan, 122(10):676-680, 2002.

[3] "Practical RDF". (2003) RDF/XML focused but well written and highly recommended http://www.oreilly.com/catalog/pracrdf/

[4] Michael C. Daconta et al., "The Semantic Web: A Guide to the Future of XML, Web Services and Knowledge Management" Complementary alternative http://www.amazon.co.uk/exec/obidos/ASIN/0471432571/

[5] R. Agrawal, A. Somani, and Y. Xu. Storage and Querying of E-Commerce Data. In Proc. of VLDB, 2001.

[6] D. Florescu and D. Kossman. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. Technical report, INRIA, France, May 1999.

[7] D. Beckett and J. Grant, Mapping Semantic Web Data with RDBMSes. 2001, http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/

[8] Z. Pan and J. Heflin, "DLDB: Extending Relational Databases to Support Semantic Web Queries". Technical Report: LU-CSE-04-2006

[9] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis & K.Tolle, "On Storing Voluminous RDF Description: The case of Web Portal Catalogs", In Proc. of the 4th International Workshop on the Web and Databases (WebDB2001) in conjunction with ACM SIGMOD'01 Conference, 2001.

[10] Stonebraker et al. . "C-Store: A Column-Oriented DBMS." ,VLDB, 2005.

[11] Daniel J. Abadi, Samuel Madden, Nabil Hachem, "Column-stores vs. row-stores: how different are they really?", In Proc. SIGMOD Conference, 2008: 967-980

[12] Harizopoulos, Liang, Abadi, Madden . "Performance Tradeoffs in Read-Optimized Databases", In proc. VLDB, 2006.

[13] Xin Wang, Shuyi Wang, Pufeng Du and Zhiyong Feng. "Storing and Indexing RDF Data in a Column-Oriented DBMS". In proc. IEEE Conference, 2010.

[14] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach, "Scalable semantic web data management using vertical partitioning," In *Proc. VLDB*, pp. 411–422, 2007.

[15] C. Weiss, P. Karras, and A. Bernstein, "Hexastore: sextuple indexing for semantic web data management", In Proc. VLDB, pp. 1008–1019, 2008.

[16] Schlicht, A, Stuckenschmidt H. . " A Flexible Partitioning Tool for Large Ontologies". In proc. International Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM , 2008.

[17]B. Konev, C. Lutz, D. Walther, and F. Wolter, "Logical Difference and Module Extraction with CEX and MEX", In Proc. Description Logics, 2008.

[18] Paul Doran, "Ontology reuse via ontology modularization", In proc. KnowledgeWeb PhD Symposium, 2006.

[19] A. Silberschatz, et al., "Database System Concepts", 5th edition, Mc Graw Hill, 2006.

[20] M. T. Ozsu, et al., "Principles of Distributed Database Systems", Prentice Hall, USA, 1999.

[21]Lefteris Sidirourgos, Romulo Goncalves, Martin Kersten, Niels Nes, Stefan Manegold. "Column Store Support for RDF Data Management: not all swans are white". In Proc. VLDB ,2008.

[22] Daniel J. Abadi, Adam Marcus, Samuel Madden, Kate Hollenbach: "SW-Store: a vertically partitioned DBMS for Semantic Web data management". VLDB J. 18(2): 385-406 , 2009.

[23] T. Neumann and G. Weikum, "RDF-3X: a RISC-style engine for RDF", In *Proc. VLDB*, pp. 647–659, 2008.

[24] H. Stuckenschmidt and M.C.A. Klein, "Structure-Based Partitioning of Large Concept Hierarchies", In Proc. International Semantic Web Conference, 2004, pp.289-303.

[25] Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt, Marta Sabou, "Criteria and Evaluation for Ontology Modularization Techniques. Modular Ontologies" 2009: 67-89

[26] Jie Bao, Doina Caragea, Vasant Honavar, "Modular Ontologies - A Formal Investigation of Semantics and Expressivity", In proc. Asian Semantic Web Conference, 2006, pp.616-631

[27] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, Aditya Kalyanpur, "Automatic Partitioning of OWL Ontologies Using E-Connections", in proc. Description Logics, 2005.

[28] Pathak J, Johnson TM, Chute CG. "A Survey of Modular Ontology Techniques and their Applications in the Biomedical Domain. International Journal of Integrated Computer-Aided Engineering", Vol. 16(3), 2009; pp. 225-242.

[29] Palmisano, I., Tamma, V., Payne, T. R., Doran, P.. "Task Oriented Evaluation of Module Extraction Techniques", In Proc. The Eighth International Semantic Web Conference (ISWC'09), October 25th-29th 2009.

[30] Julian Seidenberg, Alan Rector. "Web Ontology Segmentation: Analysis, Classification and Use" In Proc. the 15th international conference on World Wide Web, 2006, pp. 13-22.

[31] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler, "Extracting Modules from Ontologies: A Logic-based Approach", In Proceedings of OWLED, 2007.