STORING RDF GRAPHS USING NL-ADDRESSING[1]

Krassimira Ivanova, Vitalii Velychko, Krassimir Markov

Abstract*: NL-addressing is a possibility to access information using natural language words as addresses of the information stored in the multi-dimensional numbered information spaces. For this purpose the internal encoding of the letters is used to generate corresponded co-ordinates. The tool for working in such style is named OntoArM. Its main principles, functions and using for storing RDF graphs are outlined in this paper.*

Keywords*: NL-addressing, RDF graphs, ontology representations.*

ACM Classification Keywords*: D.4.2 Storage Management; E.2 Data Storage Representations.*

Introduction

Resource Description Framework (RDF) is the W3C recommendation for semantic annotations in the Semantic Web. RDF is a standard syntax for Semantic Web annotations and languages [Klyne & Carroll, 2004].

The underlying structure of any expression in RDF is a collection of triples, each consisting of a ***subject***, a ***predicate*** and an ***object***. A set of such triples is called an ***RDF graph***. This can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link (hence the term "graph") (Fig.1).

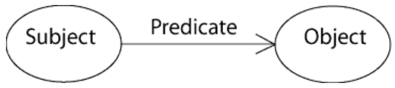

Fig. 1. RDF triple

[1] Reprinted from: G.Setlak, M.Alexandrov, K.Markov (Eds.), Artificial Intelligence Methods and Techniques for Business and Engineering Applications. ITHEA, 2012, Rzeszow, Poland; Sofia, Bulgaria, ISBN: 978-954-16-0057-3 (printed), ISBN: 978-954-16-0058-0 (online). pp. 84-98

Each triple represents a statement of a relationship between the things denoted by the nodes that it links. Each triple has three parts: (1) subject, (2) object, and (3) a predicate (also called a *property*) that denotes a relationship. The direction of the arc is significant: it always points toward the object. The nodes of an RDF graph are its subjects and objects.

The assertion of an RDF triple says that some relationship, indicated by the predicate, holds between the things denoted by subject and object of the triple. The assertion of an RDF graph amounts to asserting all the triples in it, so the meaning of an RDF graph is the conjunction (logical AND) of the statements corresponding to all the triples it contains. A formal account of the meaning of RDF graphs is given in [Hayes, 2004].

The state of the art with respect to existing storage and retrieval technologies for RDF data is given in [Hertel et al, 2009]. Different repositories are imaginable, e.g. main memory, files or databases. RDF schemas and instances can be efficiently accessed and manipulated in main memory. For persistent storage the data can be serialized to files, but for large amounts of data the use of a database management system is more reasonable. Examining currently existing RDF stores we found that they are using relational and object-relational database management systems. Storing RDF data in a relational database requires an appropriate table design. There are different approaches that can be classified in (1) generic schemas, i.e. schemas that do not depend on the ontology, and (2) ontology specific schemas.

In the following we will present a new approach for organizing graph data bases, called Natural Language Addressing (NL-Addressing) and will illustrate it for the most important ontological table designs.

Natural Language Addressing (NL-Addressing)

The idea of Natural Language Addressing (NL-Addressing) is very simple. It is based on the computer internal representation of the word as strings of codes in any system of encoding (ASCII, UNICODE, etc.).

For example, the ASCII encoding of the word „accession" has the next representation: 97 99 99 101 115 115 105 111 110. It may be used as co-ordinate array, which indicates a point in the multidimensional information space, where the corresponded information may be stored.

It is clear, the words have different lengths and, in addition, some phrases may be assumed as single concepts. This means that we need a tool for managing multidimensional information spaces with possibility to support all needed dimensions in one integrated structure.

The independence of dimensionality limitations is very important for developing new intelligent systems aimed to process high-dimensional data. To achieve this, we need information models and corresponding access methods to cross the boundary of the dimensional limitations and to obtain the possibility to work with information spaces with variable and practically unlimited number of dimensions. Such possibility is given by the Multi-Dimensional Information Model (MDIM) [Markov, 2004] and correspond Multi-Dimensional Access Method (MDAM) [Markov, 1984]. Its advantages have been demonstrated in many practical realizations during more than twenty-five years. In recent years, this kind of memory organization has been implemented in the area of intelligent systems memory structuring for several data mining tasks and especially in the area of association rules mining [Mitov et al, 2009]. Here we will show its applicability for organizing of RDF stores.

Multi-dimensional numbered information spaces

Main structures of Multi-Dimensional Information Model (MDIM) are *basic information elements, information spaces, indexes* and *meta-indexes,* and *aggregates*. The definitions of these structures are remembered below.

The ***basic information element*** ($\boldsymbol{BIE}$) of MDIM is an arbitrary long string of machine codes (bytes). When it is necessary, the string may be parceled out by lines. The length of the lines may be variable.

Let ***the universal set*** $\boldsymbol{UBIE}$ be the set of all $\boldsymbol{BIE}$.

Let $\boldsymbol{E}_1$ be a set of basic information elements. Let μ_1 be a function, which defines a biunique correspondence between elements of the set $\boldsymbol{E}_1$ and elements of the set $\boldsymbol{C}_1$ of positive integer numbers, i.e.:

$$\boldsymbol{E}_1 = \{e_i \mid e_i \in \boldsymbol{UBIE}, i=1,\dots,m_1\},\ \boldsymbol{C}_1 = \{c_1 \mid c_i \in \boldsymbol{N}, i=1,\dots,m_1\};\ \boldsymbol{\mu_1 : E_1 \leftrightarrow C_1}$$

The elements of $\boldsymbol{C}_1$ are said to be numbers (co-ordinates) of the elements of $\boldsymbol{E}_1$.

The triple $\boldsymbol{S_1 = (E_1, \mu_1, C_1)}$ is said to be a ***numbered information space of range 1*** (one-dimensional or one-domain information space).

Let $\boldsymbol{NIS}_1$ be a set of all one-dimensional information spaces.

The triple $\boldsymbol{S_2 = (E_2, \mu_2, C_2)}$ is said to be a ***numbered information space of range 2*** (two-dimensional or multi-domain information space of range two) iff the elements of $\boldsymbol{E}_2$ are numbered information spaces of range one (i.e. belong to the set $\boldsymbol{NIS}_1$) and μ_2 is a function which defines a biunique correspondence between elements of $\boldsymbol{E}_2$ and elements of the set $\boldsymbol{C}_2$ of positive integer numbers, i.e.:

$$\boldsymbol{E}_2 = \{e_i \mid e_i \in \boldsymbol{NIS}_1, i=1,\dots,m_2\},\ \boldsymbol{C}_2 = \{c_i \mid c_i \in \boldsymbol{N}, i=1,\dots,m_2\};\ \boldsymbol{\mu_2 : E_2 \leftrightarrow C_2}$$

Let $\boldsymbol{NIS}_{n-1}$ be a set of all (n-1)-dimensional information spaces.

The triple $\boldsymbol{S_n = (E_n, \mu_n, C_n)}$ is said to be a ***numbered information space of range n*** (n- dimensional or multi-domain information space) iff the elements of $\boldsymbol{E}_n$ are numbered information spaces of range n-1 (belong to the set $\boldsymbol{NIS}_{n-1}$) and μ_n is a function which defines a biunique correspondence between elements of $\boldsymbol{E}_n$ and elements of the set $\boldsymbol{C}_n$ of positive integer numbers, i.e.:

$$\boldsymbol{E}_n = \{e_j \mid e_j \in \boldsymbol{NIS}_{n-1}, j=1,\dots,m_n\},\ \ \boldsymbol{C}_n = \{c_j \mid c_j \in \boldsymbol{N}, j=1,\dots,m_n\};\ \boldsymbol{\mu_n : E_n \leftrightarrow C_n}$$

The information space $\boldsymbol{S}_n$, which contains all information spaces of a given application is called ***information base*** of range **n**. The concept information base without indication of the range is used as generalized concept to denote all available information spaces.

The sequence $\boldsymbol{A} = (c_n, c_{n-1}, \dots, c_1)$, where $c_i \in \boldsymbol{C}_i$, $i=1, \dots, n$ is called ***multidimensional space address*** of range ***n*** of a basic information element. Every space address of range ***m***, ***m < n***, may be extended to space address of range ***n*** by adding leading *n-m* zero codes. Every sequence of space addresses $A_1, A_2, \dots, A_k$, where ***k*** is arbitrary positive number, is said to be a ***space index***.

Every index may be considered as a basic information element, i.e. as a string, and may be stored in a point of any information space. In such case, it will have a multidimensional space address, which may be pointed in the other indexes, and, this way, we may build a hierarchy of indexes. Therefore, every index, which points only to indexes, is called ***meta-index***.

The approach of representing the interconnections between elements of the information spaces using (hierarchies) of meta-indexes is called ***poly-indexation*.**

Let $G = \{S_i \mid i=1, \dots, n\}$ be a set of numbered information spaces.

Let $\boldsymbol{\tau} = \{v_{ij} : S_i \to S_j \mid i=const, j=1, \dots, n\}$ be a set of mappings of one "main" numbered information space $\boldsymbol{S}_i \in G \mid i=const$, into the others $\boldsymbol{S}_J \in G, j=1, \dots, n$, and, in particular, into itself.

The couple: $D = (G, \boldsymbol{\tau})$ is said to be an "***aggregate***".

It is clear, we can build **m** aggregates using the set G because every information space $\boldsymbol{S}_J \in G, j=1, \dots, n$, may be chosen to be the main information space.

Operations in the MDIM

After presenting the information structures, we need to remember the operations, which are admissible in the model. In MDIM, we assume that ***all*** *information elements of* ***all*** *information spaces* ***exist***.

If for any $\boldsymbol{S}_i : \boldsymbol{E}_i = \emptyset \wedge \boldsymbol{C}_i = \emptyset$, than it is called ***empty***.

Usually, most of the information elements and spaces are empty. This is very important for practical realizations.

Because of the rule that all structures exist, we need only two operations with a ***BIE***: updating and getting the value and two service operations: getting the length of a ***BIE*** and positioning in a ***BIE***.

Updating, or simply – ***writing*** the element, has several modifications with obvious meaning: writing as a whole; appending/inserting; cutting/replacing a part; deleting.

There is only one operation for getting the value of a ***BIE***, i.e. ***read*** a portion from a ***BIE*** starting from given position. We may receive the whole ***BIE*** if the starting position is the beginning of ***BIE*** and the length of the portion is equal to the ***BIE*** length.

We have only one operation with a **single space** – *clearing (deleting) the space*, i.e. replacing all ***BIE*** of the space with Ø (empty ***BIE***). After this operation, all ***BIE*** of the space will have zero length. Really, the space is cleared via replacing it with empty space.

We may provide two operations with **two spaces**: (1) *copying* and (2) *moving* the first space in the second. The modifications concern how the ***BIE*** in the recipient space are processed. We may have: copy/move with clearing the recipient space; copy/move with merging the spaces.

The first modifications first clear the recipient space and after that provide a copy or move operation. The second modifications may have two types of processing: destructive or constructive. The ***destructive merging*** may be "conservative" or "alternative". In the conservative approach, the ***BIE*** of recipient space remains in the result if it is with none zero length. In the other approach – the ***BIE*** from donor space remains in the result. In the ***constructive merging*** the result is any composition of the corresponding ***BIE*** of the two spaces.

Of course, the move operation deletes the donor space after the operation.

Special kind of operations concerns the *navigation* in a space. We may receive the space address of the ***next*** or ***previous***, ***empty*** or ***non-empty***, elements of the space starting from any given co-ordinates.

The possibility to count the number of non empty elements of a given space is useful for practical realizations.

Operations with indexes, meta-indexes, and aggregates in the MDIM are based on the classical logical operations – intersection, union, and supplement, but these operations are not so trivial. Because of the complexity of the structure of the information spaces, these operations have two different realizations.

Every information space is built by two sets: the set of co-ordinates and the set of information elements. Because of this, the operations with indexes, meta-indexes, and aggregates may be classified in two main types: (1) operations based only on co-ordinates, regardless of the content of the structures; (2) operations, which take in account the content of the structures:

- The operations based only on the co-ordinates are aimed to support information processing of analytically given information structures. For instance, such structure is the table, which may be represented by an aggregate. Aggregates may be assumed as an extension of the relations in the sense of the model of Codd [Codd, 1970]. The relation may be represented by an aggregate if the aggregation mapping is one-one mapping. Therefore, the aggregate is a more universal structure than the relation and the operations with aggregates include those of relation theory. What is the new is that the mappings of aggregates may be not one-one mappings.
- In the second case, the existence and the content of non empty structures determine the operations, which can be grouped corresponding to the main information structures: elements, spaces, indexes, and meta-indexes. For instance, such operation is the **projection**, which is the analytically given space index of non-empty structures. The projection is given when some coordinates (in arbitrary positions) are fixed and the other coordinates vary for all possible values of coordinates, where non-empty elements exist. Some given values of coordinates may be omitted during processing.

Other operations are transferring from one structure to another, information search, sorting, making reports, generalization, clustering, classification, etc.

OntoArM

The program realization of MDIM is called Multi-Domain Access Method (MDAM). For a long period, it has been used as a basis for organization of various information bases. There exist several realizations of MDAM for different hardware and/or software platforms. The most resent one is the FOI Archive Manager – ArM [Markov et al, 2008]. The newest MDAM realization is called ArM32 (for MS Windows). [Markov, 2004]

The OntoArM is an ontological graph oriented access method but not a middleware in the sense of [Hertel et al, 2009]. It is an upgrade of ArM32.

The OntoArM ontological elements are organized in ontological graph spaces with variable ranges. There is no limit for the ranges of the spaces. Every ontological element may be accessed by a corresponding multidimensional space address (coordinates) given via NL-word or phrase. Therefore, we have two main constructs of the physical organizations of OntoArM – ontological spaces and ontological elements.

In OntoArM the length of the ontological element (string) may vary from 0 up to 1G bytes. There is no limit for the number of strings in an archive but their total length plus internal indexes could not exceed the limited length of the file system for a single file (4G, 8G, etc.). There is no limit for the numbers of files in the information base as well as for theirs dispositions.

OntoArm operations inherited from ArM32

The operations with basic information elements are:

- *ArmRead* (reading a part or a whole element);
- *ArmWrite* (writing a part or a whole element);
- *ArmAppend* (appending a string to an element);
- *ArmInsert* (inserting a string into an element);
- *ArmCut* (removing a part of an element);
- *ArmReplace* (replacing a part of an element);
- *ArmDelete* (deleting an element);
- *ArmLength* (returns the length of the element in bytes).

The operations over the spaces are:

- *ArmDelSpace* (deleting the space),
- *ArmCopySpace* and *ArmMoveSpace* (copying/moving the first space in the second in the frame of one file),
- *ArmExportSpace* (copying one space from one file the other space, which is located in other file).

The operations, aimed to serve the navigation in the information spaces return the space address of the ***next*** or ***previous***, ***empty*** or ***non-empty*** elements of the space starting from any given co-ordinates. They are *ArmNextPresent, ArmPrevPresent, ArmNextEmpty*, and *ArmPrevEmpty*.

The projections' operations return the space address of the ***next*** or ***previous non-empty*** elements of the projection starting from any given co-ordinates. They are *ArmProjNext* and *ArmProjPrev*.

The operations, which create indexes, are:

– *ArmSpaceIndex* – returns the space index of the non-empty structures in the given information space;
– *ArmProjIndex* – gives the space index of basic information elements of a given projection

The service operations for counting non-empty elements or subspaces are correspondingly:

– *ArmSpaceCount* – returns the number of the non-empty structures in given information space;
– *ArmProjCount* – gives the number of elements of given (hierarchical or arbitrary) projection.

OntoArm RDF graph oriented operations

Converting strings into space addresses

There are two internal operations for conversion:

- *ArmStr2Addr* – converts string to space address. Four ASCII symbols or two UNICODE 16 symbols form one co-ordinate word. This reduces four, respectively – two, times the space' dimensions. The string is extended with leading zeroes if it is needed.
- *ArmAddr2Str* – converts space address in ASCII or UNICODE string. The leading zeroes are not included in the string.

The operations for conversion are not needed for the end-user because they are used by the upper level operations given below. All OntoArM operations access the information by NL-addresses (given by a NL-words or phrases). Because of this we will not point specially this feature.

OntoArM operations for storing and receiving RDF information

There are two main operations for creating the RDF-store:

- *OntoArmWrite* – writes a buffer (usually NL-string).
- *OntoArmRead* – reads a buffer (usually NL-string).

It is clear; to work easily with RDF graphs, several additional operations are needed:

- *OntoArmAppend (appending a string to an element);*
- *OntoArmInsert (inserting a string into an element);*
- *OntoArmCut (removing a part of an element);*
- *OntoArmReplace (replacing a part of an element);*
- *OntoArmDelete (deleting an element);*
- *OntoArmLength (returns the length of the element in bytes).*

OntoArM operations for graph navigation

The operations, aimed to serve the navigation in the graph are context depended – the format of the elements is important for the navigation. If the element is an NL-index, the navigation operation may take its ***next*** or ***previous*** NL-word for further processing. If the element has more complicated structure, the navigation operations have to be accommodated to it. In general, these operations are usual ones for navigating in the graph structures.

NL-Addressing for ontology generic schemas

Vertical representation

The simplest RDF generic schema is the triple store with only one table required in the database. The table contains three columns named *Subject*, *Predicate* and *Object*, thus reflecting the triple nature of RDF statements. This corresponds to the *vertical representation* for storing objects in a table [Agrawal et al, 2001].

The greatest advantage of this schema is that no restructuring is required if the ontology changes. Adding new classes and properties to the ontology can be realized by a simple INSERT command in the table. On the other hand, performing a query means searching the whole database and queries involving joins become very expensive. Another aspect is that the class hierarchy cannot be modeled in this schema, what makes queries for all instances of a class rather complex [Hertel et al, 2009].

It is easy to store this schema via OntoArM. The *Subject* will be the address and all its couples (*Predicate*, *Object*) may be stored at one and the same address. This way with one operation all arcs of the node of the graph will be received. There exists another variant of organization where the *Predicate* may be additional co-ordinate or name of the archive. In this case, additional operations for reading arcs will be needed. Nevertheless, in all cases the OntoArM will have linear complexity O(max_L), where max_L is the maximal length of the word or phrases, used for NL-addressing. In the same time, the relational table has complexity at least O(n log n), where n is number of all indexed elements (words), if we will take in account supporting indexing and binary search. Of course, the memory for binary indexes exceeds the OntoArM memory for internal indexes. At the end, the time for direct access is many times less then via binary search. The speed experiments with *Firebird* relation data base had showed about 30-ty times for reading and more than 90-ty times for writing in ArM's favor [Markov et al, 2008].

Normalized triple store

The triple store can be used in its pure form [Oldakowski et al, 2005], but most existing systems add several modifications to improve performance or maintainability. A common approach, the so-called *normalized triple store*, is adding two further tables to store resource URIs and literals separately as shown in Fig. 2, which requires significantly less storage space [Harris & Gibbins, 2003]. Furthermore, a hybrid of the simple and the normalized triple store can be used, allowing storing the values themselves either in the triple table or in the resources table [Jena2, 2012].

Trilpes:

Subject	Predicate	IsLiteral	Object
r1	*r2*	*False*	*r3*
r1	*r4*	*True*	*l1*
…	…	…	…

Resources:

ID	URI
r1	*…#1*
r2	*…#2*
…	…

Literals:

ID	Value
l1	*Value1*
…	…
…	…

Fig. 2. Normalized triple store

In a further refinement, the Triples table can be split horizontally into several tables, each modeling an RDF(S) property:

– SubConcept for the rdfs:subClassOf property, storing the class hierarchy

- SubProperty for the rdfs:subPropertyOf property, storing the property hierarchy
- PropertyDomain for the rdfs:domain property, storing the domains and cardinalities of properties
- PropertyRange for the rdfs:range property, storing the ranges of properties
- ConceptInstances for the rdf:type property, storing class instances
- PropertyInstances for the rdf:type property, storing property instances
- AttributeInstances for the rdf:type property, storing instances of properties with literal values

These tables only need two columns for *Subject* and *Object*. The table names implicitly contain the predicates. This schema separates the ontology schema from its instances, explicitly models class and property hierarchies and distinguishes between class-valued and literal-valued properties [Broekstra, 2005; Gabel et al, 2004].

The normalized triple store is ready for representing via OntoArM. Only what we have to do is to take in account the representing all arcs from a node by one space NL-index and the representing all properties as an aggregate. The *Subject* will be the NL-address and only *Object* will be saved. Possibility to concatenate all *Objects* for a *Subject* reduces the size of memory and time. There are different approaches for building the aggregate – using additional co-ordinate to the *Subjects’* values or to use separate archives for storing the information.

In all cases, the OntoArM has linear complexity O(max_L), the relation data base – at least O(n log n).

NL-Addressing for ontology specific schemas

Horizontal representation

Ontology specific schemas are changing when the ontology changes, i.e. when classes or properties are added or removed. The basic schema consists of one table with one column for the instance ID, one for the class name and one for each property in the ontology. Thus, one row in the table corresponds to one instance. This schema is corresponding to the *horizontal representation*

[Agrawal et al, 2001] and obviously has several drawbacks: large number of columns, high sparsity, inability to handle multi-valued properties and the need to add columns to the table when adding new properties to the ontology, just to name a few.

Horizontally splitting this schema results in the so called one-table-per class schema - one table for each class in the ontology is created. A class table provides columns for all properties whose domain contains this class. This is tending to the classic entity-relationship-model in database design and benefits queries about all attributes and properties of an instance.

However, in this form the schema still lacks the ability to handle multi-valued properties, and properties that do not define an explicit domain must then be included in each table. Furthermore, adding new properties to the ontology again requires restructuring existing tables [Hertel et al, 2009].

The horizontal representation is an example of a set of aggregates in the sense of OntoArM. Storing every class in a separate archive gives possibility to add properties without restructuring existing tables because the aggregate may be described by a meta-index. Again, NL-addressing in OntoArM has linear complexity O(max_L), the relation data base representation – at least O(n log n).

Decomposition storage model

Another approach is vertically splitting the schema, what results in the one-table-per-property schema, also called the *decomposition storage model*.

In this schema one table for each property is created with only two columns for *Subject* and *Object*. RDF(S) properties are also stored in such tables, e.g. the table for rdf:type contains the relationships between instances and their classes.

This approach is reflecting the particular aspect of RDF that properties are not defined inside a class. However, complex queries considering many properties have to perform many joins, and queries for all instances of a class are similarly expensive as in the generic triple schema [Hertel et al, 2009].

In practice, a hybrid schema combining the table-per-class and table-per property schemas is used to benefit from the advantages of both of them. This schema contains one table for each class, only storing there a unique ID for the specific instance. This replaces the modeling of the rdf:type property. For all other properties tables are created as described in the table-per-property approach (Fig. 3) [Pan & Heflin, 2004]. Thus, changes to the ontology do not require changing existing tables, as adding a new class or property results in creating a new table in the database.

ClassA:

ID
…#1
…

Property1:

Subject	Object
…#1	…#3
…	…

ClassB:

ID
…#3
…

Fig. 3. Hybrid schema

A possible modification of this schema is separating the ontology from the instances. In this case, only instances are stored in the tables described above.

Information about the ontology schema is stored separately in four additional tables *Class*, *Property*, *SubClass* and *SubProperty* [Alexaki et al, 2001]. These tables can be further refined storing only the property ID in the Property table and the domain and range of the property in own tables Domain and Range [Broekstra, 2005]. This approach is similar to the refined generic schema, where the ontology is stored the same way and only the storage of instances is different.

To reduce the number of tables, single-valued properties with a literal as range can be stored in the class tables. Adding new attributes would then require changing existing tables. Another variation is to store all class instances in one table called Instances. This is especially useful for ontologies where there is a large number of classes with only few or no instances [Alexaki et al, 2001].

The decomposition storage model is memory and time consuming due to duplicating the information and generation of too much binary search indexes. It is very near to the OntoArM style and may be directly implemented using NL-addressing but this will be not efficient. NL-addressing permits new possibilities due to omitting of explicit given information – names as well as binary indexes. The feature tables may be replaced by NL-addressing access to corresponded points of the information space where all information about given *Subject* will exist. This way we will reduce the needed memory and time. At the end, let point again, that NL-addressing has linear complexity O(max_L) and the relation data base representation – at least O(n log n).

Conclusion

NL-addressing is a possibility to access information using natural language words as addresses of the information stored in the multi-dimensional numbered information spaces. For this purpose the internal encoding of the letters is used to generate corresponded co-ordinates. The tool for working in such style is named OntoArM. Its main principles, functions and using for storing RDF graph were outlined in this paper.

There are further issues not pointed above, which may require an extension of the triple-based schemas and thus are affecting the design of the database: (1) Storing multiple ontologies in one database; (2) Storing statements from multiple documents in one database.

Both points are concerning the aspect of provenance, which means keeping track of the source an RDF statement is coming from. When storing multiple ontologies in one database it should be considered that classes, and consequently the corresponding tables, can have the same name. Therefore, either the tables have to be named with a prefix referring to the source ontology [Pan & Heflin, 2004] or this reference is stored in an additional attribute for every statement. A similar situation arises for storing multiple documents in one database. Especially, when there are contradicting statements it is important to know the source of each statement. Again, an additional attribute denoting the source document helps solving the problem [Pan & Heflin, 2004].

The concept of named graphs [Caroll et al, 2004] is including both issues. The main idea is that each document or ontology is modeled as a graph with a distinct name, mostly a URI. This name is stored as an additional attribute, thus extending RDF statements from triples to so-called quads. For the database schemas described above this means adding a fourth column to the tables and potentially storing the names of all graphs in a further table.

All these problems can be solved by OntoArM, because a separated ontology may be represented in one single archive. In addition, the NL-addressing permits accessing the equal names in different ontologies without any additional indexing or using of pointers, identification and etc. Only the NL-words or phrases are enough to access all information in all existing ontologies (resp. graphs).

The linear complexity O(max_L) of NL-addressing is very important for realizing very large triple stores.

OntoArM is implemented in the Institute of Cybernetics V.M. Glushkov at the National Academy of Sciences of Ukraine, Kiev (IC NASU). It has been used for storing ontology information about multiple documents from own data bases as well as from different internet sources.

The further work is concerned to implementing OntoArM for storing multiple ontologies in the libraries of the "Instrumental Complex with Ontological Purpose", which is under developing in the IC NASU.

Acknowledgements

The paper is partially financed by the project ITHEA XXI of the Institute of Information Theories and Applications FOI ITHEA and the Consortium FOI Bulgaria (www.ithea.org, www.foibg.com).

Bibliography

[Agrawal et al, 2001] Agrawal R, Somani A, Xu Y Storage and querying of e-commerce data. In: Proceedings of the 27th Conference on Very Large Data Bases, VLDB 2001,Roma, Italy.

[Alexaki et al, 2001] Alexaki S, Christophides V, Karvounarakis G, Plexousakis D, Tolle K (2001) The ICS-FORTH RDFSuite: Managing voluminous RDF description bases. In: Proceedings of the 2nd International Workshop on the Semantic Web, Hongkong.

[Broekstra, 2005] Broekstra J. Storage, querying and inferencing for Semantic Web languages. PhD Thesis, Vrije Universiteit, Amsterdam (2005).

[Caroll et al, 2004] Caroll J, Bizer C, Hayes P, Stickler P (2004) Semantic Web publishing using named graphs. In: Proceedings of Workshop on Trust, Security, and Reputation on the SemanticWeb, at the 3rd International SemanticWeb Conference, ISWC 2004, Hiroshima, Japan.

[Codd, 1970] Codd, E.: A relation model of data for large shared data banks. Magazine Communications of the ACM, 13/6, 1970, pp.377 387.

[Gabel et al, 2004] Gabel T, Sure Y, Voelker J (2004) KAON – An overview. Insititute AIFB, University of Karlsruhe. http://kaon.semanticweb.org/main kaonOverview.pdf.

[Harris & Gibbins, 2003] Harris S, Gibbins N 3store: Efficient bulk RDF storage. In: Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems, PSSS 2003, Sanibel Island, FL, USA.

[Hayes, 2004] Patrick Hayes, Editor, *RDF Semantics*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ . Latest version available at http://www.w3.org/TR/rdf-mt/ .

[Hertel et al, 2009] Alice Hertel, Jeen Broekstra, and Heiner Stuckenschmidt. RDF Storage and Retrieval Systems. In: S. Staab and R. Studer (eds.), Handbook on Ontologies, International Handbooks on Information Systems, DOI 10.1007/978-3-540-92673-3, Springer-Verlag Berlin Heidelberg 2009. pp 489-508.

[Jena2, 2012] Jena2 database interface – database layout. http://jena.sourceforge.net/DB/layout.html. (visited at 22.08.2012)

[Klyne & Carroll, 2004] Graham Klyne and Jeremy J. Carroll, Editors, *Resource Description Framework (RDF): Concepts and Abstract Syntax*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ . Latest version available at http://www.w3.org/TR/rdf-concepts/ .

[Markov et al, 2008] Markov K, Ivanova, K., Mitov, I., & Karastanev, S. Advance of the access methods. Int. J. Information Technologies and Knowledge, 2/2, 2008, pp.123-135

[Markov, 1984] Кr.Markov. A Multi-domain Access Method. // Proceedings of the International Conference on Computer Based Scientific Research. Plovdiv, 1984. pp. 558-563.

[Markov, 2004] Markov, K. Multi-domain information model. Int. J. Information Theories and Applications, 11/4, 2004, pp.303-308.

[Mitov et al, 2009] Mitov, I., Ivanova, K., Markov, K., Velychko, V., Vanhoof. K., Stanchev, P. "PaGaNe" – A classification machine learning system based on the multidimensional numbered information spaces. In World Scientific Proc. Series on Computer Engineering and Information Science, No.2, pp.279 286.

[Oldakowski et al, 2005] Oldakowski R, Bizer C, Westphal D RAP: RDF API for PHP. In: Proceedings of Workshop on Scripting for the Semantic Web, SFSW 2005, at 2nd European Semantic Web Conference, ESWC 2005, Heraklion, Greece.

[Pan & Heflin, 2004] Pan Z, Heflin J (2004) DLDB: Extending relational databases to support Semantic Web queries. Technical Report LU-CSE-04-006, Department of Computer Science and Engineering, Lehigh University.

Authors' Information

***Krassimira Ivanova** – University of National and World Economy, Sofia, Bulgaria*
e-mail: krasy78@mail.bg
Major Fields of Scientific Research: Data Mining

***Vitalii Velychko** – Institute of Cybernetics, NASU, Kiev, Ukraine*
e-mail: Velychko@rambler.ru
Major Fields of Scientific Research: Data Mining, Natural Language Processing

***Krassimir Markov** – Institute of Mathematics and Informatics at BAS, Sofia, Bulgaria;*
e-mail: markov@foibg.com
Major Fields of Scientific Research: Multi-dimensional information systems, Data Mining