INTRODUCTION TO STORING GRAPHS BY NL-ADDRESSING

Krassimira B. Ivanova, Koen Vanhoof, Krassimir Markov, Vitalii Velychko

***Abstract**: This paper introduces an approach to storing graphs based on the Natural Language Addressing in multidimensional numbered information spaces. A sample graph is analyzed to find its proper NL-representation. Taking in account the interrelations between nodes and edges, a "multi-layer" representation is possible and identifiers of nodes and edges can be avoided. The advantages and disadvantages of NL-addressing for the multi-layer representation of graphs are discussed.*

***Keywords**: graphs; addressing; natural language addressing.*

***ACM Classification Keywords**: D.4.3 File Systems Management, Access methods.*

Introduction

This paper introduces an original approach to storing graphs based on the so called "Natural Language Addressing" (NL-addressing) in multidimensional numbered information spaces [Ivanova et al, 2012; Ivanova et al, 2013].

Firstly, our attention will be paid to addressing and naming (labeling) in graphs with regards to introducing the NL-addressing. A sample graph will be analyzed to find its proper NL-representation. Taking in account the interrelations between nodes and edges, we will see that a "multi-layer" representation is possible and the identifiers of nodes and edges can be avoided.

As result of the analysis of the examples, the advantages and disadvantages of NL-addressing for the multi-layer representation of graphs will be discussed. The conclusion is that if we will use the indexed files or relational data bases, the disadvantages may make the implementation impossible. A solution of this problem may be the using of NL-addressing which consists in assuming the internal computer codes of letters as co-ordinates in a multidimensional information space.

Addressing and naming (labeling) in graphs

Graph theory may be said to have its beginning in 1736 when EULER considered the (general case of the) Königsberg bridge problem:

> "Is there a walk crossing each of the seven bridges of Königsberg (now Kaliningrad, Russia) exactly once?" [Euler, 1736] (Figure 1)

What is interesting in the scheme on Figure 1 is that every bridge has two kinds of identification:

- Its own ***name***: Krämer Br. (Shopkeeper Br.), Schmiede Br. (Blacksmith Br.), Grüne Br. (Green Br.), Köttel Br. (Guts, Giblets Br.), Honig Br. (Honey Br.), Holz Br. (Wooden Br.), and Hohe Br. (High Br.);
- Its own ***address***: a, b, c, d, e, f, g.

This reflects in our further research in two directions – *naming* and *addressing*, respectively.

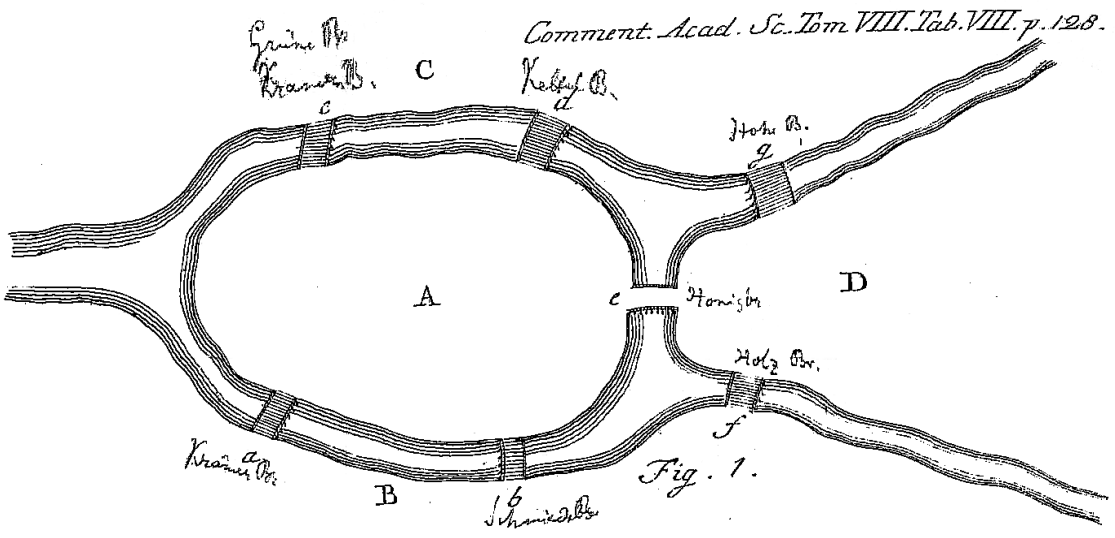


Figure 1. Illustration of Königsberg bridge problem [Euler, 1736]

As example of the ***addressing direction***, we may point the term "*addressable graph*" [Harju, 2011].

For instance, a computer network can be presented as a graph ***G***, where the vertices are the node computers, and the edges indicate the direct links. Each computer ***v*** has an address ***a(v)***, a bit string (of zeros and ones). The length of an address is the number of its bits. A message, that is sent to ***v***, is preceded by the address ***a(v)***. The Hamming distance ***h(a(v), a(u))*** of two addresses of the same length is the number of places, where ***a(v)*** and ***a(u)*** differ; e.g., h(00010, 01100) = 3 and h(10000, 00000) = 1.

It would be a good way to address the vertexes so that the Hamming distance of two vertices is the same as their distance in ***G***. In particular, if two vertices were adjacent, their addresses should differ by one symbol. This would make it easier for a node computer to forward a message.

A graph ***G*** is said to be addressable, if it has an addressing such that ***dG(u, v) = h(a(u), a(v))*** (Figure 2).

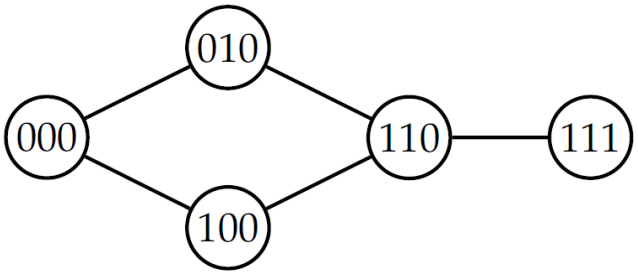


Figure 2. Example of an addressable graph

As example of the ***naming direction***, we may point the term "*named graphs*" [NG, 2013].

"Named Graphs" is the idea that having multiple RDF graphs in a single document/repository and naming them with URIs provides useful additional functionality built on top of the RDF Recommendations [RDF, 2013].

Let remember, the 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") (Figure 3).

Figure 3. RDF triple

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].

A *Named Graph* is an RDF graph which is assigned a name in the form of an URIref. The name of a graph may occur either in the graph itself, in other graphs, or not at all. Graphs may share URIrefs but not blank nodes. Named Graphs can be seen as a reformulation of quads in which the fourth element's distinct syntactic and semantic properties are clearly distinguished, and the relationship to RDF's triples, abstract syntax and semantics is clearer [Carroll et al, 2005].

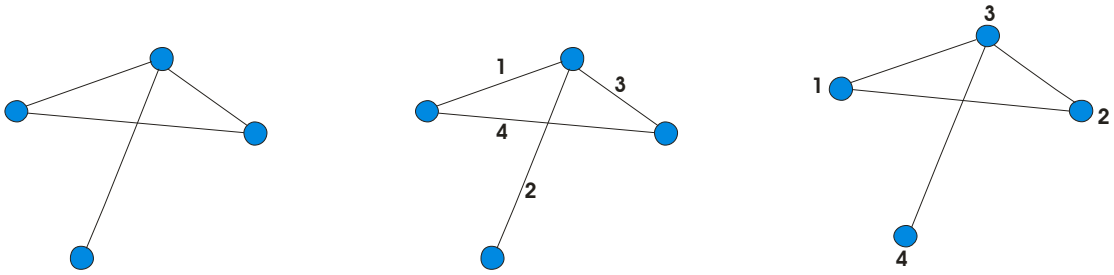
In other words, a Named Graph is a set of triples named by an URI. This URI can then be used outside or within the graph to refer to it.

Named Graphs aim at more complex RDF application areas like:

- Data syndication and lineage tracing;
- Ontology versioning;
- Modeling context;
- Modeling access control;
- Expressing privacy preferences;
- Scoping assertions.

Named graphs are kind of "*Labeled Graphs*". A graph labeling is an assignment of integers to the vertices or edges, or both, subject to certain conditions. Graph labeling was first introduced in the late 1960s. In the intervening years dozens of graph labeling techniques have been studied in over 1000 papers [Gallian, 2011].

A labeled graph ***G = (V, E)*** is a finite series of graph vertices ***V*** with a set of graph edges ***E*** of 2-subsets of ***V***. The term "labeled graph" when used without qualification means a graph with each node labeled differently (but arbitrarily), so that all nodes are considered distinct for purposes of enumeration [Weisstein, 2013] (Figure 4).



unlabeled graph *edge-labeled graph* *vertex-labeled graph*

Figure 4. Labeled graphs

NL-addressing in graphs

As it is seen from Figure 1, the set of addresses is isomorphic to set of names, i.e. the correspondence between two sets is one-one. This means that using of one or other of these sets closely depends of the interpreter and its functionality. If the interpreter is a computer or a mathematician, the addresses (numbers or letters) are preferable. If the interpreter is an end-user (human), the names (natural language words or phrases) are preferable.

Is it possible one and the same string of letters to be used as both name and address?

The positive answer is given by NL-addressing.

To illustrate NL-addressing in the multi-dimensional information spaces let see an example (Figure 5).

For instance, a separate *basic information element* (**BIE**) may be a letter, a word, a phrase of words (string). Such information element is colored in magenta on Figure 5 and has number 114.

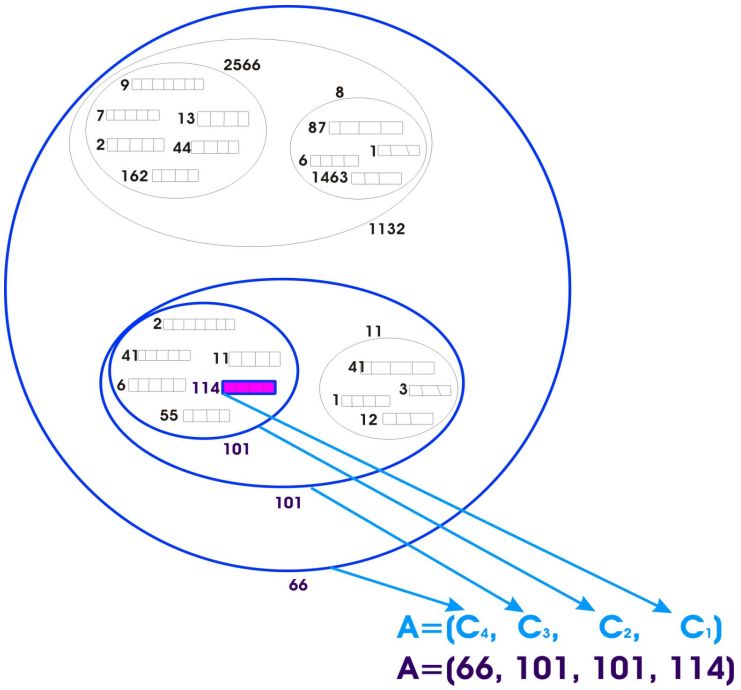


Figure 5. Example of a space address

Let a set of such elements is numbered by the human and stored in any archive (file). If we have several such sets, we may number them again and store in a common archive. And so on. This way we receive a specific hierarchy of numbered sets. If one will write the sequence of numbers of the sets starting from the one which contain all others, he will create a *space address*. The space address showed on Figure 5 means that the space with number 66 contains the space with number 101. The numbering is unique for every set. Because of this, there is no problem to have the same numbers in the included sets what is illustrated at the Figure 5 – set 101 contains element with number 101 which is a set of elements. Finally, the last set contains element with number 114 which is not a set but string of symbols.

In other words, the space address of this string is *A = (66, 101, 101, 114)* and its content may be written or read directly using this address.

Now we may to illustrate the idea of NL-addressing.

Consider the space address we just have seen – *(66, 101, 101, 114)*.

If we assume these numbers as ASCII codes, i.e. *66 = B, 101 = e, 114 = r,* we may "understand" the space address as the word "*Beer*" (Figure 6).

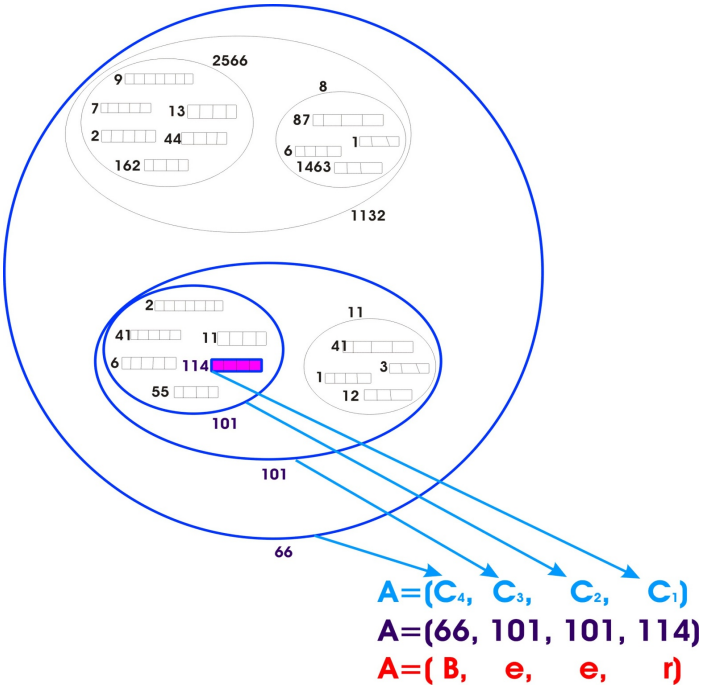


Figure6. Example of natural language space address

At the end, we have to illustrate the **BIE** (content) which may be stored at such NL-space address. It may be arbitrary long string of words. In our example we choose the **BIE** to be the remarkable aphorism of Benjamin Franklin: *"Beer is proof that God loves us and wants us to be happy"* (Figure 7).

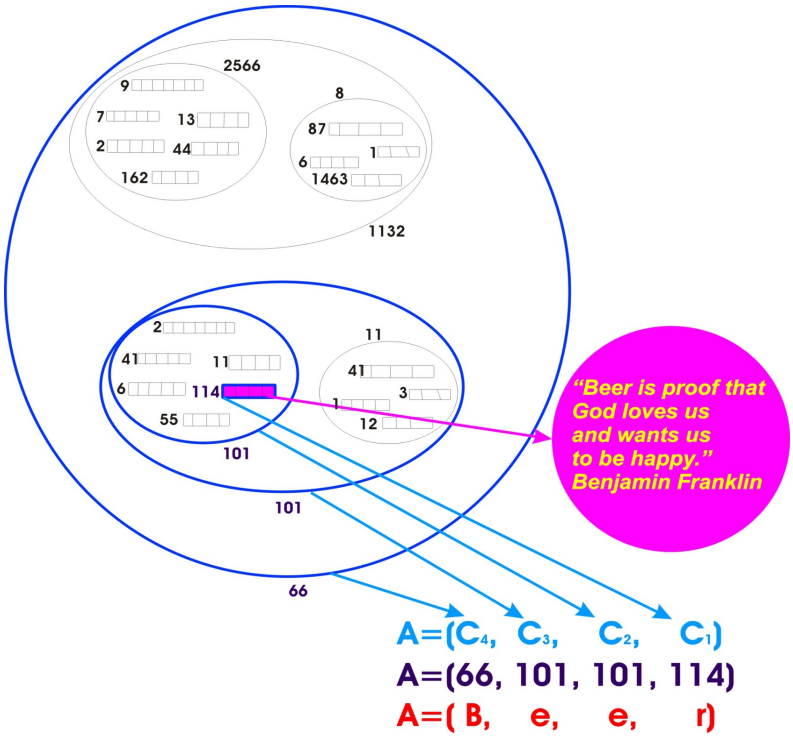


Figure 7. Example of content accessed by NL-addressing

In this case (Figure 7) the couple *{(space address A), (BIE)}* is:

{(B, e, e, r), ("Beer is proof that God loves us and wants us to be happy." Benjamin Franklin)}

To access the text, we have to convert index *(B, e, e, r)* to index *(66, 101, 101, 114)* and to use corresponded access operations, i.e. we have the consequence:

Beer =>
=> (B, e, e, r) =>
=> (66, 101, 101, 114) =>
=> ("Beer is proof that God loves us and wants us to be happy." Benjamin Franklin).

To illustrate using of NL-addressing, let see two simple examples.

- **Example 1. Multi-layer representation of a sample graph**

Consider a sample graph [GraphDB, 2012], which contains three named nodes *(Alice, Bob, Chess)* with addresses *(Id.1, Id.2 and Id.3)*, six labeled edges connecting them *(knows, knows, is_member, members, members, is_member)* with addresses *(Id.100, ..., Id.105)*, and some features and their values (Figure 5).

To represent this graph in the computer memory we have to use (identificators as) pointers. It is possible to present the information in two tables – one for nodes (Table 1) and another for the edges (Table 2). The names of columns and values of the corresponded cells are easy understandable.

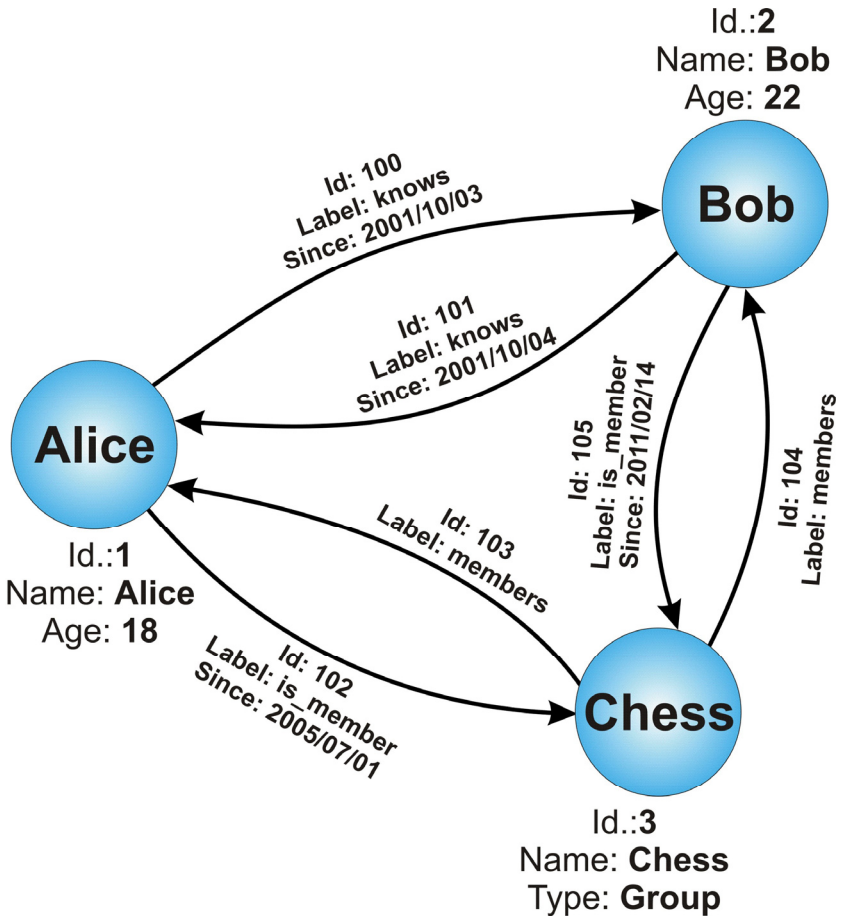


Figure 5. A sample named/addressed graph

Table 1. Description of nodes of the sample graph

Node Id.	Name	from-edges	to-edges	Age	Type
1	**Alice**	**101; 103**	**100; 102**	**18**	**-**
2	**Bob**	**100; 104**	**101; 105**	**22**	**-**
3	**Chess**	**102; 105**	**103; 104**	**-**	**Group**

Table 2. Description of the edges of the sample graph

Edge Id.	Label	from node	to node	Since
100	**knows**	**1**	**2**	**2001/10/03**
101	**knows**	**2**	**1**	**2001/10/04**
102	**is_member**	**1**	**3**	**2005/07/01**
103	**members**	**3**	**1**	**-**
104	**members**	**3**	**2**	**-**
105	**is_member**	**2**	**3**	**2011/02/14**

Each table corresponds to one type of components – to nodes or to the edges. Every row corresponds to one identificator (address) and connected to it features.

If we will take in account the interrelations between nodes and edges, we will see that another ("multi-layer") representation is possible and the identifiers of nodes and edges can be avoided (Table 3).

Table 3. Multi-layer representation of the sample graph.

		addresses		
		Alice	*Bob*	*Chess*
layers	*Age*	**18**	**22**	
	Type			**Group**
	knows; since	**Bob; 2001/10/03**	**Alice; 2001/10/04**	
	members			**Alice; Bob**
	is_member since	**Chess; 2005/07/01**	**Chess; 2011/02/14**	

In this case, names of nodes are addresses (i.e. names of columns) and names of edges will define the different "layers" (i.e. – rows of the table) in which the corresponded values are stored at the address (column) given by node names.

To find all edges from given node we have to start with node name (column), for instance "Bob", and read all information from different layers (rows) stored at address (in column) "Bob". In other words, all needed information for the node is in the column with corresponded node name.

If we will have possibility for NL-addressing it will reduce the information to be stored on the disc - only the cells with text in bold of Table 3 will be stored. At a rough estimate, the sum of filled cells in:

- The first case (Table 1 + Table 2) has at least 18 + 30 = 48 filled cells for the two tables and real need of additional indexing to speed up the access;
- In the second case (Table 3) – 8 filled cells and no need of additional indexing.

The graph database models

Let remember that the "*graph database model*" is a model in which the data structures for the schema and/or instances are modeled as a directed, possibly labeled, graph, or generalizations of the graph data structure, where data manipulation is expressed by graph-oriented operations and type constructors, and appropriate integrity constraints can be defined over the graph structure [Angles & Gutierrez, 2008].

Graph database model can be defined as those in which data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors.

The notion of graph database model can be conceptualized with respect to three basic components, namely: (1) Data structures; (2) Transformation language; (3) Integrity constraints. Hence, a graph database model is characterized as follows:

- Data and/or the schema are represented by graphs, or by data structures generalizing the notion of graph (hypergraphs or hypernodes) [Guting, 1994; Levene & Loizou, 1995; Kuper & Vardi, 1984; Paredaens et al, 1995; Kunii, 1987; Graves et al, 1995a; Gyssens et al, 1990].
- Data manipulation is expressed by graph transformations, or by operations whose main primitives are on graph features like paths, neighborhoods, subgraphs, graph patterns, connectivity, and graph statistics (diameter, centrality, etc.) [Gyssens et al, 1990; Graves et al, 1995a; Guting, 1994];
- Integrity constraints enforce data consistency. These constraints can be grouped in schema-instance consistency, identity and referential integrity, and functional and inclusion dependencies. Examples of these are: labels with unique names, typing constraints on nodes, functional dependencies, domain and range of properties [Graves et al, 1995b; Kuper & Vardi, 1993; Klyne & Carroll, 2004; Levene & Poulovassilis, 1991].

Graph database models are applied in areas where information about data interconnectivity or topology is more important, or as important, as the data itself. In these applications, the data and relations among the data are usually at the same level. Introducing graphs as a modeling tool has several advantages for this type of data:

- It allows for a more natural modeling of data. Graph structures are visible to the user and they allow a natural way of handling applications data, for example, hypertext or geographic data. Graphs have the advantage of being able to keep all the information about an entity in a single node and showing related information by edges connected to it [Paredaens et al, 1995]. Graph objects (like paths and neighborhoods) may have first order citizenship; a user can define some part of the database explicitly as a graph structure [Guting, 1994], allowing encapsulation and context definition [Levene & Poulovassilis, 1990].
- Queries can refer directly to this graph structure. Associated with graphs are specific graph operations in the query language algebra, such as finding shortest paths, determining certain subgraphs, and so forth. Explicit graphs and graph operations allow users to express a query at a high level of abstraction. To some extent, this is the opposite of graph manipulation in deductive databases, where often, fairly complex rules need to be written [Guting, 1994]. It is not important to require full knowledge of the structure to express meaningful queries [Abiteboul et al, 1997]. Finally, for purposes of browsing it may be convenient to forget the schema [Buneman et al, 1996].
- For implementation, graph databases may provide special graph storage structures, and efficient graph algorithms for realizing specific operations [Guting, 1994]. These structures are very important – *the features of the graph storage structures influence the corresponded to them operations and algorithms*.

➢ **Example 2. The toy genealogy graph database**

Let's look at another example - the toy genealogy from [Angles & Gutierrez, 2008] (Figure 6.). The genealogy diagram (right-hand side) is represented as two tables (left-hand side) NAME-LASTNAME and PERSON-PARENT (Children inherit the last name of the father just for modeling purposes).

NAME	LASTNAME
George	Jones
Ana	Stone
Julia	Jones
James	Deville
David	Deville
Mary	Deville

PERSON	PARENT
Julia	George
Julia	Ana
David	James
David	Julia
Mary	James
Mary	Julia

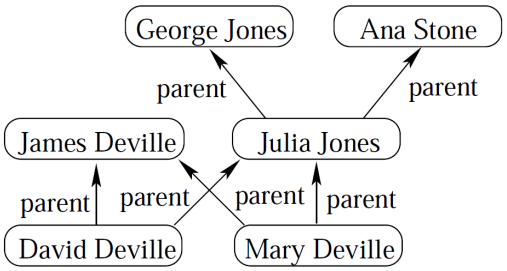


Figure 6. Running example: the toy genealogy graph database

The "multi-layer" representation of the family tree from Figure 6 is given in Table 5.

Table 5. Multi-layer representation of the family tree from Figure 6

		addresses					
		George	Ana	Julia	James	David	Mary
layers	lastname	***Jones***	***Stone***	***Jones***	***Deville***	***Deville***	***Deville***
	parent_of			***George; Ana***		***James; Julia***	***James; Julia***

The NL-addressing means direct access to content of each cell. Because of this, for NL-addressing the problem of recompiling the database after updates does not exist. In addition, the multi-layer representation and natural language addressing *reduce resources* (24 cells in Figure 6 vs. 9 cells in the case of Table 5) and *avoid using of supporting indexes for information retrieval services* (B-trees, hash tables, etc.).

Discussion

The state of the art with respect to existing storage and retrieval technologies for RDF graphs 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.

Graph database models took off in the eighties and early nineties alongside object oriented models. Their influence gradually died out with the emergence of other database models, in particular geographical, spatial, semi-structured, and XML. Recently, the need to manage information with graph-like nature has reestablished the relevance of this area [Angles & Gutierrez, 2008].

The graph oriented approach for storing ontologies became one of the preferred. Some of the world's leading companies and products which support extra-large ontology bases are presented on page of W3C [LTS, 2012]. It should be noted, there exists a gradual transition from relational to non-relational models for organizing

ontological data. Perhaps the most telling example is the system AllegroGraph® 4.9 [AlegroGraph, 2012] of the FRANZ Inc.

Concluding the examples, let point on advantages and disadvantages of the proposed here multi-layer NL-representation of graphs.

The main advantages are:

- Reducing the number of tables, which represent the graph;
- Reducing the number of filled cells.

The main disadvantages are:

- The tables are sparse;
- Avoiding pointers we receive a variety of names, which have different lengths and cause difficulties for the implementations in the data bases where the fixed length is preferable;
- The number of nodes may be very great and this way needs corresponded number of columns in the table (in any cases hundreds or thousands).

Multi-domain information model (MDIM)

If we will use the indexed files or relational data bases, the disadvantages of such data models may make the implementation impossible. We propose another approach which is based on addressing with variable number of co-ordinates. For instance, the ASCII internal representation of the word "accession" is the following unique sequence of numbers: *(97, 99, 99, 101, 115, 115, 105, 111, 110)*. It is its NL-address (9 co-ordinates) and its internal computer representation. Using the same approach, the names of our sample graph will be represented as follow (Table 6):

Table 6. Names of the nodes of the sample graph and theirs co-ordinate representations

name	co-ordinates	comment
Alice:	(65, 108, 105, 99, 101)	5 co-ordinates (dimensions)
Bob:	(66, 111, 98)	3 co-ordinates (dimensions)
Chess:	(67, 104, 101, 115, 115)	5 co-ordinates (dimensions)

As it is seen from Table 6, words (and phrases) have different lengths and require using of addressing by co-ordinate arrays with variable length, i.e. to have variable dimensions in one and the same time. Such addressing we call "multi-dimensional".

Fortunately, there exist approach called "Multi-Domain Information Model" (MDIM) and corresponded to it "Multi-Domain Access Method" (MDAM) which permit operating with multi-dimensional addressing. Detailed and formal presentations of MDIM have been given in [Markov, 1984; Markov, 2004]. There exist several realizations of MDAM for different hardware and/or software platforms. The most resent one is the Archive Manager – ArM [Markov et al, 2008]. We have upgraded it for NL-addressing approach and applied for NL-storing of graphs. Below we will illustrate this model by short comments and several figures.

We consider the type of memory organization, which is based on the numbering as a main approach. The main idea consists in mapping the values of the objects' attributes (symbol or real; point or interval) to integer numbers of the elements of corresponding ordered sets. This way, each object will be described by a vector of integer values, which may be used as the co-ordinate address in the multi-dimensional information space. This type of memory organization is called "*Multi-dimensional numbered information spaces*".

The process of mapping the names to numbers permits the use of mathematical functions and address vectors for accessing the information instead of search engines.

As an example we will consider a numbered modification of Table 3 where the names of addresses and layers are replaced by integer numbers (Table 7).

Table 7. Numbered representation of the sample graph

		addresses		
		1	*2*	*3*
layers	*1*	**18**	**22**	
	2			**Group**
	3	**Bob; 2001/10/03**	**Alice; 2001/10/04**	
	4			**Alice; Bob**
	5	**Chess; 2005/07/01**	**Chess; 2011/02/14**	

Every row of the Table 7is a numbered information space of range 1 because all elements in the every row are numbered separately. In the example this is indicated by couples *(number : element)*. This way we have:

row 1: (1 : 18); (2 : 22); (3 : None).

row 2: (1 : None); (2 : None); (3 : Group).

row 3: (1 : Bob; 2001/10/03); (2 : Alice; 2001/10/04); (3 : None).

row 4: (1 : None); (2 : None); (3 : Alice; Bob).

row 5: (1 : Chess; 2005/07/01); (2 : Chess; 2011/02/14); (3 : None).

The set of rows of the Table 7 is numbered information space of range 2 because all rows are numbered and all elements in the every row are numbered, too. This is indicated by triple

(number of row, number of element : element).

This way we have a matrix below (called Example_table 1 to be distinguished from other tables in this paper):

Example_table 1:

(1, 1 : 18); (1, 2 : 22); (1, 3 : None).

(2, 1 : None); (2, 2 : None); (2, 3 : Group).

(3, 1 : Bob; 2001/10/03); (3, 2 : Alice; 2001/10/04); (3, 3 : None).

(4, 1 : None); (4, 2 : None); (4, 3 : Alice; Bob).

(5, 1 : Chess; 2005/07/01); (5, 2 : Chess; 2011/02/14); (5, 3 : None).

Space addresses of the elements of Example_table 1 above are the couples *(1, 1), (1, 2), ..., (5, 3)*. For instance, the couple *(3, 2)* is address of string "*Alice; 2001/10/04*". If we add leading zeroes the address remain the same, i.e. *(0, 0, 0, 3, 2) = (3, 2) => "Alice; 2001/10/04"*.

A space indexes i_1 and i_2 over Example_table 1 above may be the sequences:

i_1*: {(2, 3), (3, 2), (5, 1), (3, 3)}*

i_2*: {(1, 2), (1, 3), (1, 2), (1, 3)}*

The indexes may be not sorted (see i_1) and may contain repeated addresses (see i_2).

If we assume that the equally numbered elements of the rows of Example_table 1 above corresponds each other than we may build different "sub-tables" (*ST*) such as:

ST_1*: {row 3, row 4, row 2};*

ST_2*: {row 5, row 5, row 1}.*

ST_1 and ST_2 are *aggregates*. There are no restrictions on the rules for creating the aggregates.

The aggregate ST_1 contains the next table

ST_1:

(3, 1 : Bob; 2001/10/03); (3, 2 : Alice; 2001/10/04); (3, 3 : None).

(4, 1 : None); (4, 2 : None); (4, 3 : Alice; Bob).

(2, 1 : None); (2, 2 : None); (2, 3 : Group).

The aggregate ST_2 contains the next table

ST_2:

(5, 1 : Chess; 2005/07/01); (5, 2 : Chess; 2011/02/14); (5, 3 : None).

(5, 1 : Chess; 2005/07/01); (5, 2 : Chess; 2011/02/14); (5, 3 : None).

(1, 1 : 18); (1, 2 : 22); (1, 3 : None).

Finally, the Table 7 is an aggregate which is constructed by all layers (rows). We may represent it as an aggregate as follow:

- Every layer (row) will be separate information space of range 1 and will be stored in a separate file with name of the layer.
- The names Alice, Bob, and Chess will be space addresses common for all layers, i.e. the corresponded elements will be accessible via one and same addresses.

As result we will have the following structure of files (Table 8).

Table 8. Aggregate built by information from Table 3

file name	content
Age	(Alice - 18); (Bob - 22); (Chess - None).
Type	(Alice - None); (Bob - None); (Chess - Group).
knows_since	(Alice – Bob : 2001/10/03); (Bob – Alice : 2001/10/04); (Chess - None).
members	(Alice - None); (Bob - None); (Chess - Alice; Bob).
is_member	(Alice – Chess : 2005/07/01); (Bob – Chess : 2011/02/14); (Chess - None).

Taking in account that the Alice, Bob, and Chess are not real text strings but NL-addresses, i.e. *Alice = (65, 108, 105, 99, 101)*, *Bob = (66, 111, 98)*, and *Chess = (67, 104, 101, 115, 115)* the aggregate has the following structure (Table 9).

Table 9. Storing format of the aggregate from Table 8

file name	space addresses		
	(65, 108, 105, 99, 101)	(66, 111, 98)	(67, 104, 101, 115, 115)
Age	**18**	**22**	
Type			**Group**
knows_since	**Bob; 2001/10/03**	**Alice; 2001/10/04**	
members			**Alice; Bob**
is_member	**Chess; 2005/07/01**	**Chess; 2011/02/14**	

Only the strings in bold will be stored on the disk.

Assuming that names of the files are NL-addresses, too, we may build more complex structure (Table 10) in one file using concatenated space addresses *(name | file_name)* where NL-addresses are the elements of the sets:

name : {Alice, Bob, Chess}

file_name : {Age, Type, knows_since, members, is_member}.

Again, only strings in bold will be stored on disk.

Table 10. Concatenated NL-addresses of data from Table 3

(Alice|Age : **18**)	(Bob|Age : **22**)	(Chess|Age : None)
(Alice|Type : None)	(Bob|Type : None)	(Chess|Type : **Group**)
(Alice|knows_since : **Bob; 2001/10/03**)	(Bob|knows_since : **Alice; 2001/10/04**)	(Chess|knows_since : None)
(Alice|members : None)	(Bob|members : None)	(Chess|members : **Alice; Bob**)
(Alice|is_member : **Chess; 2005/07/01**)	(Bob|is_member : **Chess; 2011/02/14**)	(Chess|is_member : None)

Representing characteristics of the nodes and edges

A question about representing the characteristics of the nodes and edges rises from the analysis of the Table 9 and Table 10. At the Figure 5 they have been written as any comments to nodes and edges.

The ***characteristics of nodes*** (viz. age, type) may be represented as additional loop edges of type "*has_characteristics*" and different characteristics may be given by keywords and corresponded values for these edges.

The ***characteristics of edges*** (viz. since) may be represented as additional information to the node pointed by the corresponded edge. This information may be given again by corresponded keywords and theirs values.

For instance, the final table representation of our sample graph is given in Table 11 and corresponded to it representation by triples is given in Table 12.

The final version of the sample graph based on the information of Table 11 (or Table 12) is shown at Figure 9.

Table 11. Final variant of the aggregate from Table 8

file names	space addresses		
	Alice	*Bob*	*Chess*
has_characteristics	**Alice - Age: 18**	**Bob - Age: 22**	**Chess - Type: Group**
knows	**Bob - since : 2001/10/03**	**Alice - since: 2001/10/04**	
members			**Alice - since: 2005/07/01; Bob - since: 2011/02/14**
is_member	**Chess - since: 2005/07/01**	**Chess - since: 2011/02/14**	

Finally, using NL-addressing, the multi-layer representation is easily understandable by humans and interpretable by the computers. To illustrate this, let see the description by triples of the sample graph (Table 12).

Table 12. Representation of the sample graph by triples

Subject	Relation	Object
Alice	*has_characteristics*	**Alice – Age : 18**
Alice	*knows*	**Bob – since : 2001/10/03**
Alice	*is_member*	**Chess – since : 2005/07/01**
Bob	*has_characteristics*	**Bob – Age : 22**
Bob	*knows*	**Alice – since : 2001/10/04**
Bob	*is_member*	**Chess – since : 2011/02/14**
Chess	*has_characteristics*	**Chess –Type : Group**
Chess	*members*	**Alice; Bob**

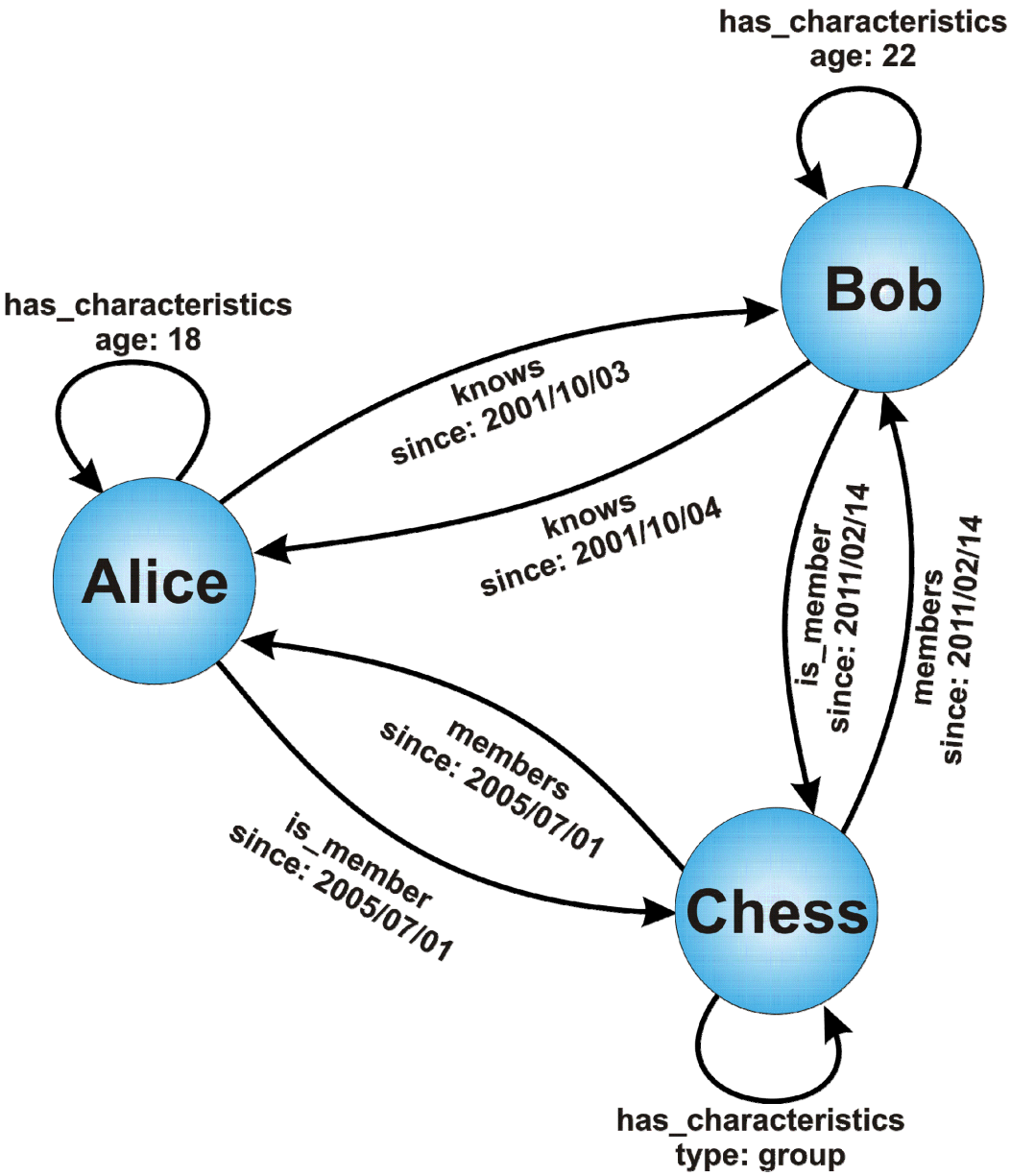


Figure 9. Final variant of the sample graph

OWL and RDF descriptions of the sample graph

To make comparison, corresponded OWL and RDF descriptions of the sample graph are given in Table 13 and Table 14 in the Appendix. They were prepared using Protégé 4.2.

Protégé is developed by the "Stanford Center for Biomedical Informatics Research" at the Stanford University School of Medicine. This is a tool which allows a user to construct domain ontology, customize data entry forms and enter data. The tool can be easily extended to access other knowledge based embedded applications. For example, Graphical widgets can be added for tables and diagrams. Protégé can also be used by other applications to access the data [protégé, 2012; protege-owl, 2012]. There is an additional option in Protégé, which serves the storing of ontologies in various relational databases, called OntoBase [Yabloko, 2011]. It should be noted that the same name "OntoBase" is used in [Pan & Pan, 2006], but without any connection to Protégé. Figure 10 illustrates the using of Protégé by graphical representation of our sample graph.

The example on Figure 10 and descriptions given in the Appendix show that the sentence:

"OWL and RDF are easy readable by humans"

is not the all truth.

Linearization the information is suitable solution for telecommunication and computer processing, but it is not easy understandable by humans. Let remember two dimensional representation of the sample graph. What is presented in four rows in Table 11 is the same as one presented on the pages with small font in the Appendix.

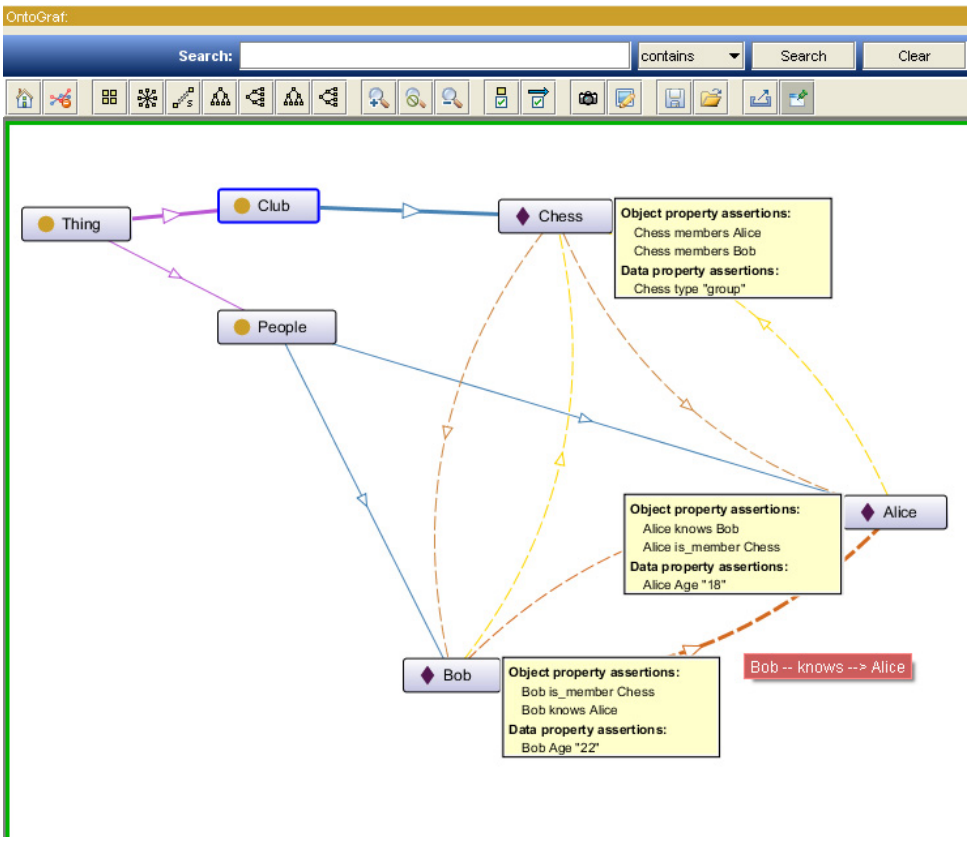


Figure 10. Protégé graphical representation of the sample graph

Conclusion

In January 1978, John F. Shoch from "Xerox Palo Alto Research Center" had written a very interesting note [Shoch, 1978a]. Later in the same year he had published this note in the paper [Shoch, 1978b]. This classical paper became as a mile stone in the further research concerning the naming, addressing and routing at the first place with its "extremely general definition" [Shoch, 1978a]:

The "name" of a resource indicates "what" we seek,

an "address" indicates "where" it is, and

a "route" tell us "how to get there".

This definition gives us a quick and intuitive understanding of the fundamental concepts of naming. Informally, a *name is a string of symbols that identifies an object*, thus both a human readable text-string and a binary number can be a name. Ideally, all objects would be named and handled in a uniform manner [Jording & Andreasen, 1994].

Shoch gave "some further detail to flesh this out" [Shoch, 1978a]:

I. A "***name***" is a symbol, usually a human-readable string, identifying some resource, or set of resources. The name (what we seek) needs to be bound to the address (where it is).

II. An "***address***", however, is the data structure whose format can be recognized by all elements in the domain, and which defines the fundamental addressable object. The address (where something is) needs to be bound to the route (how to get there).

III. A "***route***" is the specific information needed to forward a piece of information to its specified address.

Thus, a "name" may be used to derive an "address", which may then be used to derive a "route".

There is an interesting similarity between this structure and mechanisms used in programming languages (where one must bind a value to a variable), or in operating systems (where one must link a particular piece of code into a module) [Shoch, 1978a].

Shoch's definition failed to capture that addresses are names too and names must eventually be mapped to routes [Jording & Andreasen, 1994]. In this sense, the idea of NL-addressing is *to use encoding of the name as route* in a multi-dimensional information space and this way to speed the access to stored information.

In this paper, firstly our attention was paid to addressing and naming (labeling) in graphs with regards to introducing the NL-addressing in graphs. A sample graph was analyzed to find its proper representation by triples.

Taking in account the interrelations between nodes and edges, we saw that a "multi-layer" representation is possible and the identifiers of nodes and edges can be avoided. As result of the analysis of the examples, the advantages and disadvantages of the multi-layer representation of graphs were pointed and the conclusion was that if we will use the indexed files or relational data bases, the disadvantages of such data models may make the implementation impossible.

A solution of this problem may be the using of NL-addressing which consists in assuming the internal computer codes of letters as co-ordinates in multi-dimensional information space. Different words and phrases have different lengths and require using of addressing with variable length of the co-ordinate arrays, i.e. to have variable dimensions in one and the same time. Such addressing we call "multidimensional".

Our starting point of realization of our approach was the Multi-Domain Information Model (MDIM) [Markov, 2004] and corresponded Multi-Domain Access Method (MDAM) [Markov, 1984], which we upgraded to NL-addressing

approach to apply for storing graphs. The possibility to use coordinates is good for graph models where it is possible to replace search with addressing. Hence, the advantages of the numbered information spaces are:

- The possibility to build growing space hierarchies of information elements;
- The great power for building interconnections between information elements stored in the information base;
- The practically unlimited number of dimensions (this is the main advantage of the numbered information spaces for graphs where it is possible "*to address, not to search*");

The NL-addressing and multi-layer organization of the information, together with the model of representing the characteristics, are good basis for implementing this approach for real solutions.

Appendix

Table 13. The Protégé QWL description of the sample graph

```
Prefix(owl:=<http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/2000/01/rdf-schema#>)

Ontology(<http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-
        18>

Declaration(Class(<http://www.co-ode.org/ontologies/ont.owl#Club>))
Declaration(Class(<http://www.co-ode.org/ontologies/ont.owl#People>))
Declaration(ObjectProperty(<http://www.co-ode.org/ontologies/ont.owl#is_member>))
AnnotationAssertion(<http://www.co-ode.org/ontologies/ont.owl#since> <http://www.co-ode.org/ontologies/ont.owl#is_member> "")
InverseObjectProperties(<http://www.co-ode.org/ontologies/ont.owl#members> <http://www.co-ode.org/ontologies/ont.owl#is_member>)
InverseFunctionalObjectProperty(<http://www.co-ode.org/ontologies/ont.owl#is_member>)
Declaration(ObjectProperty(<http://www.co-ode.org/ontologies/ont.owl#knows>))
AnnotationAssertion(<http://www.co-ode.org/ontologies/ont.owl#since> <http://www.co-ode.org/ontologies/ont.owl#knows> "")
Declaration(ObjectProperty(<http://www.co-ode.org/ontologies/ont.owl#members>))
AnnotationAssertion(<http://www.co-ode.org/ontologies/ont.owl#since> <http://www.co-ode.org/ontologies/ont.owl#members> "")
InverseObjectProperties(<http://www.co-ode.org/ontologies/ont.owl#members> <http://www.co-ode.org/ontologies/ont.owl#is_member>)
FunctionalObjectProperty(<http://www.co-ode.org/ontologies/ont.owl#members>)
Declaration(DataProperty(<http://www.co-ode.org/ontologies/ont.owl#Age>))
DataPropertyDomain(<http://www.co-ode.org/ontologies/ont.owl#Age> <http://www.co-ode.org/ontologies/ont.owl#People>)
DataPropertyDomain(<http://www.co-ode.org/ontologies/ont.owl#Age> DataAllValuesFrom(<http://www.co-ode.org/ontologies/ont.owl#Age>
        <http://www.w3.org/2001/XMLSchema#decimal>))
Declaration(DataProperty(<http://www.co-ode.org/ontologies/ont.owl#type>))
DataPropertyDomain(<http://www.co-ode.org/ontologies/ont.owl#type> DataAllValuesFrom(<http://www.co-ode.org/ontologies/ont.owl#type>
        <http://www.w3.org/2000/01/rdf-schema#Literal>))
Declaration(NamedIndividual(<http://www.co-ode.org/ontologies/ont.owl#Alice>))
ClassAssertion(<http://www.co-ode.org/ontologies/ont.owl#People> <http://www.co-ode.org/ontologies/ont.owl#Alice>)
ObjectPropertyAssertion(Annotation(<http://www.co-ode.org/ontologies/ont.owl#since> "2005/07/01") <http://www.co-
        ode.org/ontologies/ont.owl#is_member> <http://www.co-ode.org/ontologies/ont.owl#Alice> <http://www.co-ode.org/ontologies/ont.owl#Chess>)
ObjectPropertyAssertion(Annotation(<http://www.co-ode.org/ontologies/ont.owl#since> "2001/10/03") <http://www.co-ode.org/ontologies/ont.owl#knows>
        <http://www.co-ode.org/ontologies/ont.owl#Alice> <http://www.co-ode.org/ontologies/ont.owl#Bob>)
DataPropertyAssertion(<http://www.co-ode.org/ontologies/ont.owl#Age> <http://www.co-ode.org/ontologies/ont.owl#Alice> "18")
Declaration(NamedIndividual(<http://www.co-ode.org/ontologies/ont.owl#Bob>))
ClassAssertion(<http://www.co-ode.org/ontologies/ont.owl#People> <http://www.co-ode.org/ontologies/ont.owl#Bob>)
ObjectPropertyAssertion(Annotation(<http://www.co-ode.org/ontologies/ont.owl#since> "2011/02/14") <http://www.co-
        ode.org/ontologies/ont.owl#is_member> <http://www.co-ode.org/ontologies/ont.owl#Bob> <http://www.co-ode.org/ontologies/ont.owl#Chess>)
ObjectPropertyAssertion(Annotation(<http://www.co-ode.org/ontologies/ont.owl#since> "2001/10/04") <http://www.co-ode.org/ontologies/ont.owl#knows>
        <http://www.co-ode.org/ontologies/ont.owl#Bob> <http://www.co-ode.org/ontologies/ont.owl#Alice>)
DataPropertyAssertion(<http://www.co-ode.org/ontologies/ont.owl#Age> <http://www.co-ode.org/ontologies/ont.owl#Bob> "22")
Declaration(NamedIndividual(<http://www.co-ode.org/ontologies/ont.owl#Chess>))
ClassAssertion(<http://www.co-ode.org/ontologies/ont.owl#Club> <http://www.co-ode.org/ontologies/ont.owl#Chess>)
ObjectPropertyAssertion(Annotation(<http://www.co-ode.org/ontologies/ont.owl#since> "2011/02/14") <http://www.co-ode.org/ontologies/ont.owl#members>
        <http://www.co-ode.org/ontologies/ont.owl#Chess> <http://www.co-ode.org/ontologies/ont.owl#Bob>)
ObjectPropertyAssertion(Annotation(<http://www.co-ode.org/ontologies/ont.owl#since> "2005/07/01") <http://www.co-ode.org/ontologies/ont.owl#members>
        <http://www.co-ode.org/ontologies/ont.owl#Chess> <http://www.co-ode.org/ontologies/ont.owl#Alice>)
DataPropertyAssertion(<http://www.co-ode.org/ontologies/ont.owl#type> <http://www.co-ode.org/ontologies/ont.owl#Chess> "group")
Declaration(AnnotationProperty(<http://www.co-ode.org/ontologies/ont.owl#since>))
)
```

Table 14. The Protégé RDF description of the sample graph

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY ont "http://www.co-ode.org/ontologies/ont.owl#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
   <!ENTITY owl "http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY xml "http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/XML/1998/namespace" >
   <!ENTITY rdfs "http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY rdf "http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-
ontology-18#"
    xml:base="http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-
18"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:ont="http://www.co-ode.org/ontologies/ont.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdf="http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#&rdf;"
    xmlns:xml="http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18#http://www.w3.org/XML/1998/namespace">
   <owl:Ontology rdf:about="http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-
18#http://www.semanticweb.org/wiki/ontologies/2013/2/untitled-ontology-18"/>
    <!--
   ///////////////////////////////////////////////////////////////////////////////////
   // Annotation properties
   ///////////////////////////////////////////////////////////////////////////////////
   -->
<!-- http://www.co-ode.org/ontologies/ont.owl#since -->
   <owl:AnnotationProperty rdf:about="&ont;since"/>

   <!--
   ///////////////////////////////////////////////////////////////////////////////////
   // Object Properties
   ///////////////////////////////////////////////////////////////////////////////////
    -->

   <!-- http://www.co-ode.org/ontologies/ont.owl#is_member -->
   <owl:ObjectProperty rdf:about="&ont;is_member">
      <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
      <ont:since></ont:since>
   </owl:ObjectProperty>

   <!-- http://www.co-ode.org/ontologies/ont.owl#knows -->
   <owl:ObjectProperty rdf:about="&ont;knows">
      <ont:since></ont:since>
   </owl:ObjectProperty>

   <!-- http://www.co-ode.org/ontologies/ont.owl#members -->
   <owl:ObjectProperty rdf:about="&ont;members">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
      <ont:since></ont:since>
      <owl:inverseOf rdf:resource="&ont;is_member"/>
   </owl:ObjectProperty>

   <!--
   ///////////////////////////////////////////////////////////////////////////////////
   // Data properties
   ///////////////////////////////////////////////////////////////////////////////////
    -->

   <!-- http://www.co-ode.org/ontologies/ont.owl#Age -->
   <owl:DatatypeProperty rdf:about="&ont;Age">
      <rdfs:domain rdf:resource="&ont;People"/>
      <rdfs:domain>
         <owl:Restriction>
            <owl:onProperty rdf:resource="&ont;Age"/>
            <owl:allValuesFrom rdf:resource="&xsd;decimal"/>
         </owl:Restriction>
      </rdfs:domain>
```

```
    </owl:DatatypeProperty>

    <!-- http://www.co-ode.org/ontologies/ont.owl#type -->
    <owl:DatatypeProperty rdf:about="&ont;type">
        <rdfs:domain>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&ont;type"/>
                <owl:allValuesFrom rdf:resource="&rdfs;Literal"/>
            </owl:Restriction>
        </rdfs:domain>
    </owl:DatatypeProperty>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    // Classes
    ///////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.co-ode.org/ontologies/ont.owl#Club -->
    <owl:Class rdf:about="&ont;Club"/>

    <!-- http://www.co-ode.org/ontologies/ont.owl#People -->
    <owl:Class rdf:about="&ont;People"/>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    // Individuals
    ///////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.co-ode.org/ontologies/ont.owl#Alice -->
    <owl:NamedIndividual rdf:about="&ont;Alice">
        <rdf:type rdf:resource="&ont;People"/>
        <ont:Age>18</ont:Age>
        <ont:knows rdf:resource="&ont;Bob"/>
        <ont:is_member rdf:resource="&ont;Chess"/>
    </owl:NamedIndividual>
    <owl:Axiom>
        <ont:since>2001/10/03</ont:since>
        <owl:annotatedSource rdf:resource="&ont;Alice"/>
        <owl:annotatedTarget rdf:resource="&ont;Bob"/>
        <owl:annotatedProperty rdf:resource="&ont;knows"/>
    </owl:Axiom>
    <owl:Axiom>
        <ont:since>2005/07/01</ont:since>
        <owl:annotatedSource rdf:resource="&ont;Alice"/>
        <owl:annotatedTarget rdf:resource="&ont;Chess"/>
        <owl:annotatedProperty rdf:resource="&ont;is_member"/>
    </owl:Axiom>

    <!-- http://www.co-ode.org/ontologies/ont.owl#Bob -->
    <owl:NamedIndividual rdf:about="&ont;Bob">
        <rdf:type rdf:resource="&ont;People"/>
        <ont:Age>22</ont:Age>
        <ont:knows rdf:resource="&ont;Alice"/>
        <ont:is_member rdf:resource="&ont;Chess"/>
    </owl:NamedIndividual>
    <owl:Axiom>
        <ont:since>2011/02/14</ont:since>
        <owl:annotatedSource rdf:resource="&ont;Bob"/>
        <owl:annotatedTarget rdf:resource="&ont;Chess"/>
        <owl:annotatedProperty rdf:resource="&ont;is_member"/>
    </owl:Axiom>
    <owl:Axiom>
        <ont:since>2001/10/04</ont:since>
        <owl:annotatedTarget rdf:resource="&ont;Alice"/>
        <owl:annotatedSource rdf:resource="&ont;Bob"/>
        <owl:annotatedProperty rdf:resource="&ont;knows"/>
    </owl:Axiom>

    <!-- http://www.co-ode.org/ontologies/ont.owl#Chess -->
    <owl:NamedIndividual rdf:about="&ont;Chess">
        <rdf:type rdf:resource="&ont;Club"/>
        <ont:type>group</ont:type>
```

```
      <ont:members rdf:resource="&ont;Alice"/>
      <ont:members rdf:resource="&ont;Bob"/>
   </owl:NamedIndividual>
   <owl:Axiom>
      <ont:since>2005/07/01</ont:since>
      <owl:annotatedTarget rdf:resource="&ont;Alice"/>
      <owl:annotatedSource rdf:resource="&ont;Chess"/>
      <owl:annotatedProperty rdf:resource="&ont;members"/>
   </owl:Axiom>
   <owl:Axiom>
      <ont:since>2011/02/14</ont:since>
      <owl:annotatedTarget rdf:resource="&ont;Bob"/>
      <owl:annotatedSource rdf:resource="&ont;Chess"/>
      <owl:annotatedProperty rdf:resource="&ont;members"/>
   </owl:Axiom>
</rdf:RDF>

<!-- Generated by the OWL API (version 3.4.2) http://owlapi.sourceforge.net -->
```

Acknowledgments

The article is partially financed by the project ITHEA NLA (www.ithea.org).

Bibliography

[Abiteboul et al, 1997] Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J. L "The Lorel query language for semistructured data" International Journal on Digital Libraries (JODL) 1, 1, 1997, pp. 68–88.

[AlegroGraph, 2012] AllegroGraph® 4.8, http://www.franz.com/agraph/allegrograph/ (accessed: 25.08.2012).

[Angles & Gutierrez, 2008] Angles R., C. Gutierrez "Survey of Graph Database Models", ACM Computing Surveys, Vol. 40, No. 1, Article 1, Publication date: February 2008, DOI 10.1145/1322432.1322433, http://doi.acm.org/10.1145/1322432.1322433, pp.1 - 39

[Buneman et al, 1996] Buneman, P., Davidson, S., Hillebrand, G., and Suciu, D. "A Query Language and Optimization Techniques for Unstructured Data", SIGMOD Record. 25, 2, 1996, pp. 505-516.

[Carroll et al, 2005] J.J. Carroll, C. Bizer, P. Hayes, P. Stickler. Named Graphs, Provenance and Trust; International World Wide Web Conference (IW3C2); WWW 2005, Chiba, Japan. ACM 1595930469/05/0005. pp. 613-622. http://www.ra.ethz.ch/cdstore/www2005/docs/p613.pdf (accessed: 21.02.2013)

[Euler, 1736] Leonhard Euler. Solutio Problematis a geometriam situs pertinentis. Commentarii Academiae Scientiarum Imperialis Petropolitanae 8 (1736), pp. 128-140. http://www.math.dartmouth.edu/~euler/docs/originals/E053.pdf (accessed: 21.02.2013)

[Gallian, 2011] Joseph A. Gallian. A Dynamic Survey of Graph Labeling - The electronic journal of combinatorics 18, (2011); #DS6; pp. 1-256. http://emis.matem.unam.mx/journals/EJC/Surveys/ds6.pdf (accessed: 21.02.2013)

[GraphDB, 2012] http://www.smartlab.at/tag/graphdb/ (accessed: 01.12.2012)

[Graves et al, 1995a] Graves, M., Bergeman, E. R., and Lawrence, C. B. "A Graph-Theoretic Data Model for Genome Mapping Databases", In Proc. of the 28th Hawaii Int. Conf. on System Sciences (HICSS), IEEE Computer Society, 32, 1995a.

[Graves et al, 1995b] Graves, M., Bergeman, E. R., and Lawrence, C. B. "Graph Database Systems for Genomics", IEEE Engineering in Medicine and Biology, Special issue on Managing Data for the Human Genome Project 11, 6, 1995b.

[Guting, 1994] Guting, R. H. "GraphDB: Modeling and Querying Graphs in Databases", in: Proc. of 20th, Int. Conf. on Very Large Data Bases (VLDB). Morgan Kaufmann, 1994, pp. 297–308.

[Gyssens et al, 1990] Gyssens, M., Paredaens, J., den Bussche, J. V., and Gucht, D. V. A "Graph-Oriented Object Database Model", in: Proc. of the 9th Symposium on Principles of Database Systems (PODS), ACM Press, 1990, pp. 417–424.

[Harju, 2011] Tero Harju "Lecture Notes on Graph Theory", Department of Mathematics University of Turku - Turku, Finland. 2011. http://cs.bme.hu/fcs/graphtheory.pdf (accessed: 21.02.2013).

[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/ (accessed: 21.02.2013).

[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.

[Ivanova et al, 2012] Kr. Ivanova, V. Velychko, Kr. Markov. About NL-addressing (К вопросу о естествено-языконой адрессации). In: V. Velychko et al (ed.), Problems of Computer in Intellectualization. ITHEA® 2012, Kiev, Ukraine - Sofia, Bulgaria, ISBN: 978-954-16-0061-0 (printed), ISBN: 978-954-16-0062-7 (online), pp. 77-83 (in Russian).

[Ivanova et al, 2013] Krassimira B. Ivanova, Koen Vanhoof, Krassimir Markov, Vitalii Velychko, "Introduction on the Natural Language Addressing", International Journal of Information Technologies and Knowledge, Vol.7, Number 2, 2013, pp. 139–146. ISSN 1313-0455

[Jording & Andreasen, 1994] Nick Jording and Flemming Andreasen "A Distributed Wide Area Name Service for an Object Oriented Programming System", DIKU, Department of Computer Science, University of Copenhagen, Denmark, 1994.

[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/ (accessed: 21.02.2013).

[Kunii, 1987] Kunii, H. S. DBMS with Graph Data Model for Knowledge Handling. In Proc. of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow. IEEE Computer Society Press, 1987, pp. 138–142.

[Kuper & Vardi, 1984] Kuper, G. M. and Vardi, M. Y. "A New Approach to Database Logic", In: Proc. of the 3th Symposium on Principles of Database Systems (PODS), ACM Press, 1984, pp. 86 96.

[Kuper & Vardi, 1993] Kuper, G. M. and Vardi, M. Y. "The Logical Data Model" ACM Transactions on Database Systems (TODS) 18, 3, 1993, pp. 379–413.

[Levene & Loizou, 1995] Levene, M. and Loizou, G. "A Graph-Based Data Model and its Ramifications", IEEE Transactions on Knowledge and Data Engineering (TKDE) 7, 5, 1995, pp. 809–823.

[Levene & Poulovassilis, 1990] Levene, M. and Poulovassilis, A. "The Hypernode Model and its Associated Query Language", In: Proc. of the 5th Jerusalem Conf. on Information technology. IEEE Computer Society Press, 1990, pp. 520–530.

[Levene & Poulovassilis, 1991] Levene, M. and Poulovassilis, A. "An Object-Oriented Data Model Formalised Through Hypergraphs", Data & Knowledge Engineering, (DKE) 6, 3, 1991, pp. 205 - 224.

[LTS, 2012] LargeTripleStores http://www.w3.org/wiki/LargeTripleStores (accessed: 29.08.2012)

[Markov et al, 2008] Markov, K., Ivanova, K., Mitov, I., & Karastanev, S. Advance of the access methods. International Journal of Information Technologies and Knowledge, 2(2), 2008, pp. 123–135.

[Markov, 1984] Markov Kr. 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.

[NG, 2013] W3C. Named Graphs http://www.w3.org/2004/03/trix/ (accessed: 21.02.2013).

[Pan & Pan, 2006] Pan D. and Y. Pan "Using Ontology Repository to Support Data Mining", Proceedings of the 6th, World Congress on Intelligent Control and Automation, June 21 - 23, 2006, Dalian, China, pp. 5947 – 5951.

[Paredaens et al, 1995] Paredaens, J., Peelman, P., and Tanca, L. 1995. G-Log: A Graph-Based Query Language. IEEE Transactions on Knowledge and Data Engineering (TKDE) 7, 3, 436–453.

[protégé, 2012] http://protege.stanford.edu (accessed: 25.05.2012).

[protege-owl, 2012] http://protege.stanford.edu/overview/protege-owl.html (accessed: 25.05.2012)

[RDF, 2013] http://www.w3.org/RDF/#specs (accessed: 21.02.2013).

[Shoch, 1978a] John F. Shoch "A note on Inter-Network Naming, Addressing and Routing", Xerox, Palo Alto, Research Center, Palo Alto - California 94305, USA, January 1978. http://www.postel.org/ien/pdf/ien019.pdf (accessed: 21.02.2013)

[Shoch, 1978b] John F. Shoch. Inter-Network Naming, Addressing, and Routing. In Proc. of the Seventeenth IEEE Conference on Computer Communication Networks, pages 72–79, Washington, D.C., 1978.

[Weisstein, 2013] Eric W. Weisstein Labeled Graph, From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/LabeledGraph.html (accessed: 21.02.2013)

[Yabloko, 2011] Yabloko L. OntoBase, Protégé 2011 http://protegewiki.stanford.edu/wiki/OntoBase (accessed: 02.08.2012)

Authors' Information

Ivanova Krassimira *– University of National and World Economy, Sofia, Bulgaria. Institute of Mathematics and Informatics, BAS, Sofia, Bulgaria.*
e-mail: krasy78@mail.bg
Major Fields of Scientific Research: Software Engineering, Business Informatics, Data Mining, Multidimensional multi-layer data structures in self-structured systems

Vanhoof Koen - *Professor Dr.,*
Universiteit Hasselt; Campus Diepenbeek; Department of Applied Economic Sciences
Wetenschapspark 5; bus 6; BE-3590 Diepenbeek; Belgium
e-mail: koen.vanhoof@uhasselt.be
Main research areas: data mining, knowledge retrieval.

Markov Krassimir *– ITHEA ISS IJ, IBS and IRJ Editor in chief, P.O. Box: 775, Sofia-1090, Bulgaria; e-mail: markov@foibg.com*
Major Fields of Scientific Research: General theoretical information research, Multi-dimensional models and information systems

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