Appendix A: Program Realizations

NL-addressing access method, presented in this research, has been implemented in several experimental program systems. The main of them are:

- WordArM (): a system for storing dictionaries and thesauruses using NL-addressing;
- OntoArM (): a system for storing ontologies using NL-addressing and multi-layer archive structures;
- RDFArM (): a system for storing RDF-graphs using NL-addressing.

For the purposes of this research, WordArM, OntoArM, and RDFArM are embedded as components of the program complex INFOS ("**IN**telligence **FO**rmation **S**ystem"). The front panel of system INFOS is shown on Figure 86 below.

Figure 86. The front panel of system INFOS

A1. WordArM

WordArM is a system for storing dictionaries and thesauruses through natural language addressing.

WordArM is upgrade over Natural Language Addressing Access Method and corresponded Archive Manager called **NL-ArM**, realized in this research. WordArM is aimed to store libraries of terms and their definitions. WordArM concepts are organized in multi-layer hash tables (information spaces with variable size). The definition of each term is stored in a container located by appropriate path - mapping of the natural language word or phrase, which presents the concept.

There is no limit on the number of terms in a WordArM archive, but their total length plus internal hash indexes could not exceed the file length (4G, 8G, etc.) which is enough space for several millions of concepts' definitions. There is no limit on the number of files in the data base, as well as their location, including the Internet. This permits to store unlimited number of concepts' definitions.

WordArM has two modes of operation: Automated and Manual.

The automated mode supports reading the input information from file (concepts with definitions to be stored in the archive or only list of concepts to receive their definitions from the archive). The result is storing the definitions in the WordArM archive or exporting definitions from the WordArM archive in the file.

The manual mode does the same but only for one concept which is entered manually from the corresponded screen panel.

To support these modes, WordArM has two main operations – information storing (NLA-Write) and information reading (NLA-Read), which have two variants – for automatic input and output of data from and to files, and the for manually performing these operations.

➢ *WordArM automated mode functions*

The WordArM panel for working in automated mode is shown on Figure 87.

By "**NLA-Write**" button the function for storing definitions from a file can be activated.

Each concept and its definition occupy one record in the file. There is no limit for the number of records in the file. After pressing the "NLA-Write" button, the system reads records sequentially from the file and for each of them:

(1) Transform the concept into path;

(2) Store the definition of this concept in the container located by the path.

The input file is in CSV file format. Its records have the next format:

<word/words>;<definition><CR>.

After storing the concepts' definitions, WordArM displays the contents of the input file in the window near to the "NLA-Write" button. Before the information from the file, two informative lines for time measurement in milliseconds are shown (Figure 88):

- Total time used for storing all instances from the file;
- Average time used for storing of one instance.

Figure 87. The WordArM panel for working in automated mode

Time used is highly dependent on the possibilities of operational environment and speed of computer hardware.

In the case of the Figure 88, 23412 instances were stored for 22105 milliseconds and one instance has been stored for average time of 0.94 milliseconds. In other words, one thousand instances are processed for about one second.

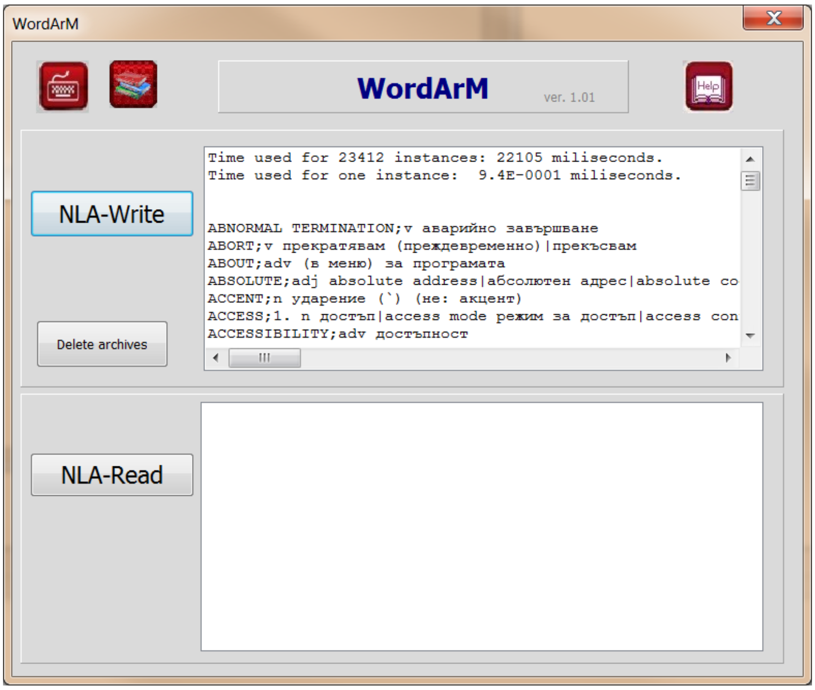

Figure 88. Content of WordArM input file with two informative lines

In the same panel (Figure 87) corresponded button enables deleting the work archive of the WordArM (ArmDict.dat, which in this version for test control is stored on the hard disk but not in the computer memory). WordArM is completed with compressing program and after storing the information prepares small archive for long time storage.

By “**NLA-Read**” button, the function for reading definitions from the WordArM archive can be activated. In the automated mode, NLA-Read uses as input a file with concepts (each on a separate line) and extract from the archive theirs definitions. If any definition does not exist, the output is empty definition.

Each concept and its definition occupy one record in the output file. There is no limit to the number of records in the file. After pressing the “NLA-Read” button, the system reads concepts sequentially from the input file and for each of them:

(1) Transform the concept into path;

(2) Extract the definition of this concept from the container located by the path.

The output file is in CSV file format. Its records have the next format:

<sequential number><word/words>;<definition><CR>.

The content of the output file is displayed in the window next to the NLA-Read button. Before the information from the file, two informative lines are shown (Figure 89):

- Total time used for extracting of all instances;
- Average time used for extracting of one instance,

in milliseconds.

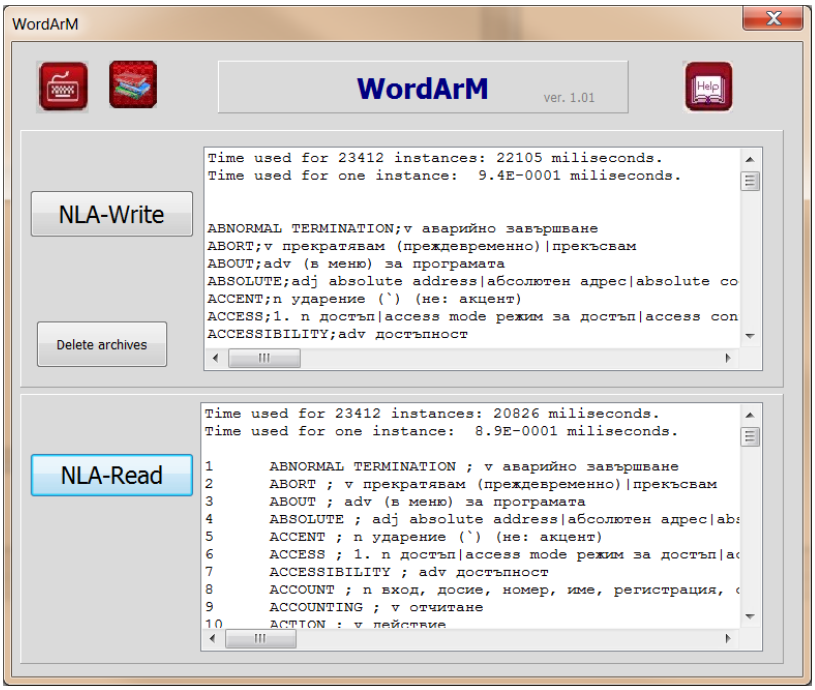

Figure 89. Content of WordArM output file with two informative lines

The time used is highly dependent on possibilities of operational environment and speed of computer hardware.

In the case of the Figure 89, 23412 instances were extracted for 22105 milliseconds and one instance has been extracted for average time of 0.94 milliseconds. In other words, more than one thousand instances are processed for about one second.

Finally, the form has three service buttons:

- The first ( ) serves as a transition to the form for manual input and output of data to/from the system archive;
- The second ( ) is connected to the module for adjusting the environment of the system – archives, input and output information, etc.;
- The third ( ) activates the help text (user guide) of the system.

➢ ***WordArM manual mode functions***

The WordArM panel for working in manual mode is shown on Figure 90.

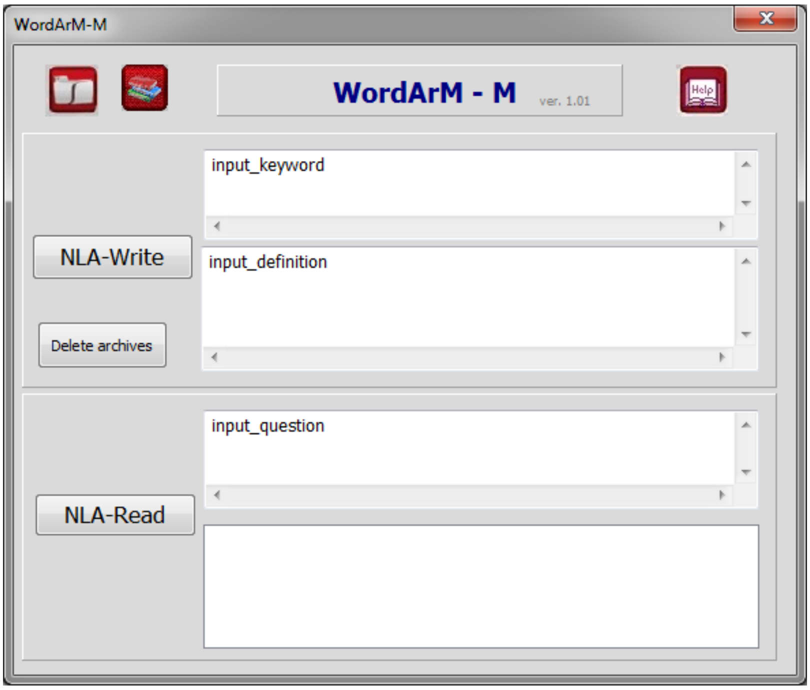

Figure 90. The WordArM panel for working in manual mode

By "**NLA-Write**" button the function for storing definitions from the form can be activated.

Each concept and its definition can be given in corresponded fields on the screen form (Figure 91).

After pressing the "NLA-Write" button, the system reads information from the fields and:

(1) Transform the concept into path;

(2) Store the definition of this concept in the container located by the path.

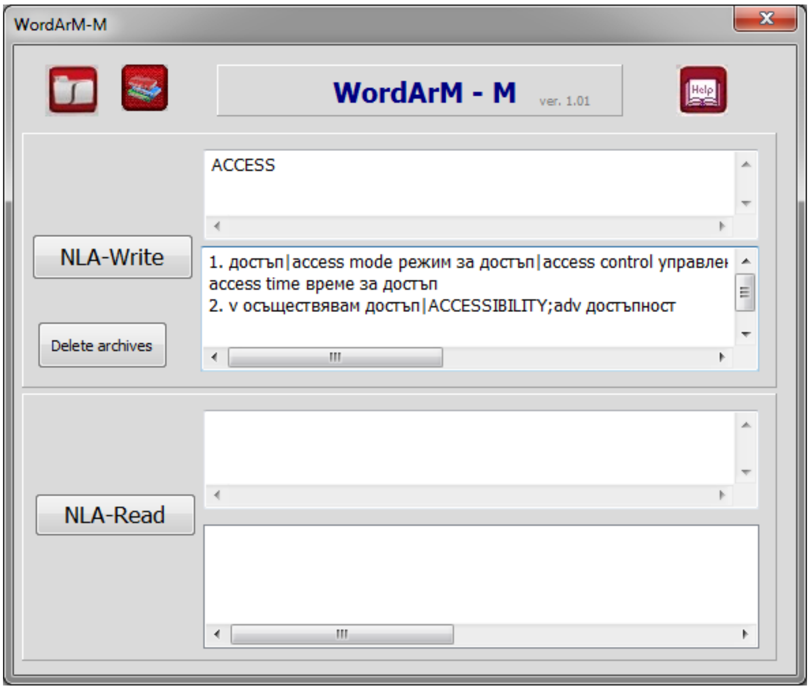

Figure 91. Manual input of the concept and its definition

By “**NLA-Read**” button the function for reading a definition from the WordArM archive can be activated. In the manual mode, NLA-Read uses as input the concept given in the screen field and extracts from the archive its definition (Figure 92). If the definition does not exist, the output is empty definition.

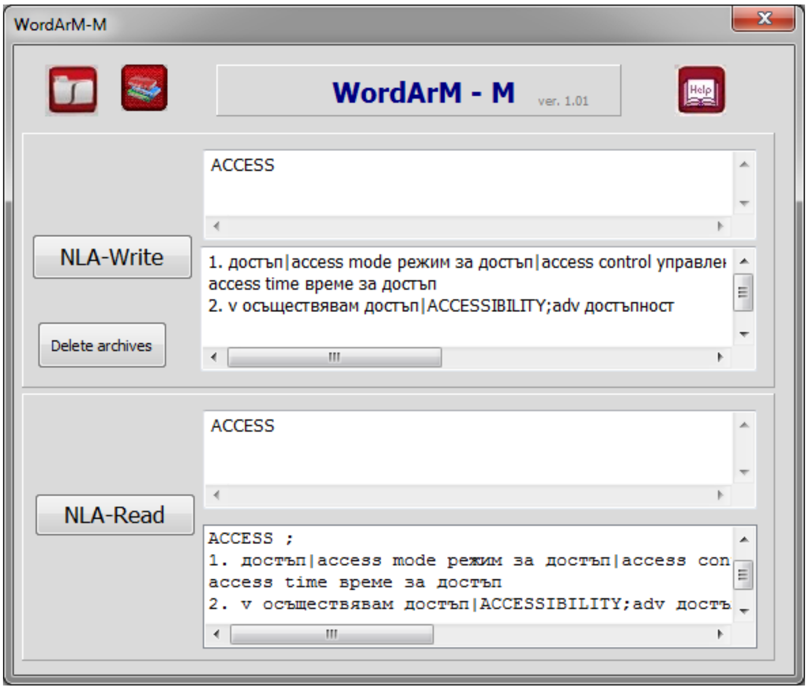

Figure 92. Manual output of the concept and its definition

After pressing the “NLA-Read” button, the system reads concept from the screen field and:

(1) Transform the concept into path;

(2) Extract the definition of this concept from the container located by the path.

The NL-addressing supports multi-language work. In other words, in the same archive we may have definitions of the concepts from different languages (Figure 93).

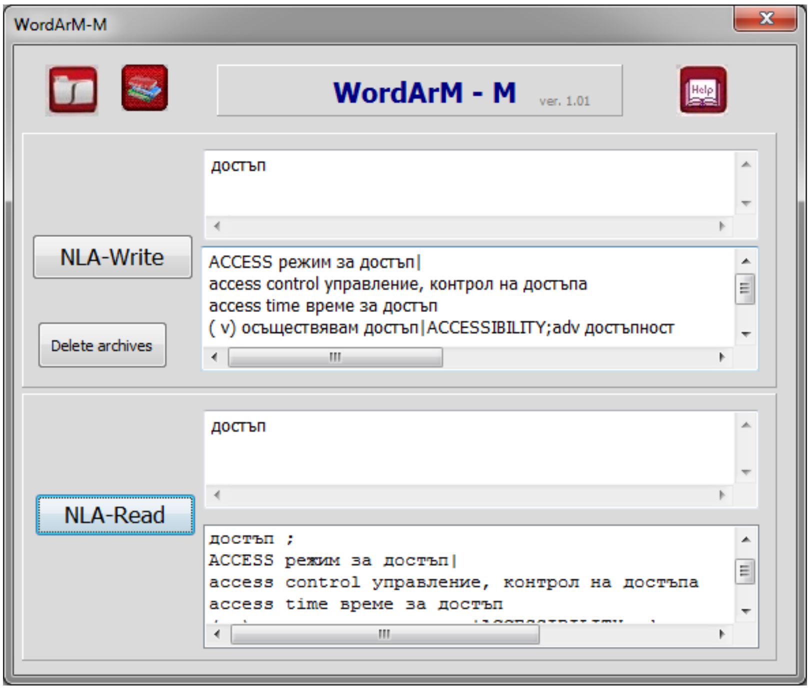

Figure 93. Simultaneous work with concepts defined in different languages.

The form for manual work has three service buttons.

– The first () serves as a return to the form for automatic input and output of data to/from the system archive;
– The second () is connected to the module for adjusting the environment of the system – archives, input and output information, etc.;
– The third () activates the help text (user guide) of the system.

The exit from the system can be done by the conventional way for Windows - by clicking on the cross in the upper right corner of the form.

A2. OntoArM

OntoArM is a system for storing ontologies through natural language addressing.

OntoArM is upgrade over Natural Language Addressing Access Method and corresponded Archive Manager called NL-ArM, realized in this research. OntoArM is aimed to store libraries of ontologies in multi-layer hash tables (information spaces with variable size). Each ontological element can be stored by appropriate path, which is set by a natural language word or phrase.

In OntoArM, the length of ontological element (string) can vary from 0 to 1G bytes. There is no limit on the number of strings in an archive, but their total length plus internal hash indexes may not exceed the capacity of the file system for one file (length of 4G, 8G, etc.). There is no limit on the number of files in the data base, as well as their location, including the Internet.

The main idea for storing ontologies in OntoArM follows the idea of multi-lear ontology representation. In other words, the ontology relations are assumed as layers and the ontology concepts are assumed as paths valid for all layers.

The information about concepts as well information about the links of the concepts with other concepts is stored in the corresponded containers located by the path in the corresponded layers.

OntoArM has two modes of operation: Automated and Manual.

➢ *OntoArM automated mode functions*

The OntoArM panel for working in automated mode is shown on Figure 94.

The main functions are Onto-Write and Onto-Read for which there are corresponded buttons.

By "**Onto-Write**" button the function for storing ontology definitions from a file can be activated.

Each triple (subject, relation and object) occupy one record in the input file. There is no limit to the number of records in the file. After pressing the "Onto-Write" button, the system reads records sequentially from the file and for each of them:

(1) Transform the subject (concept) into path;

(2) Store the object (definition and links) of the subject (concept) in the container located by the path in the file which corresponds to the layer given as relation in the triple.

The input file is in CSV file format. Its records have the next format:

<subject>;<relation>;<object><CR>.

After storing the triples, in the panel near to the "Onto-Write" button, OntoArM displays two informative lines (Figure 94):

- Total time used for storing all instances from the file;
- Average time used for storing of one instance, in ticks (milliseconds).

The time used is highly dependent on possibilities of operational environment and speed of computer hardware.

In the case of Figure 94, 117709 instances were stored for 96643 milliseconds and one instance has been stored for average time of 0.82 milliseconds. In other words, one thousand instances were processed for about less than one second.

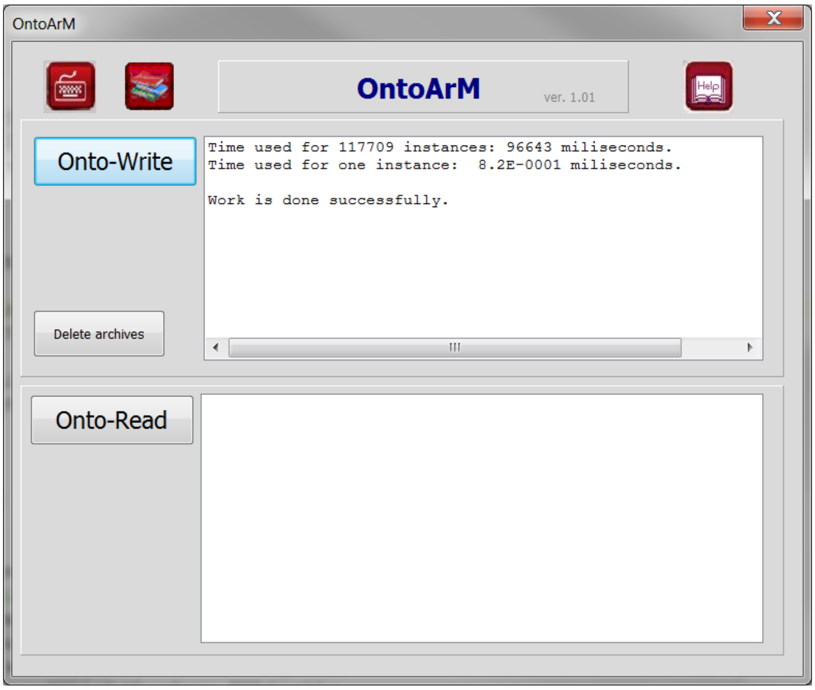

Figure 94. Content of OntoArM Onto-Write panel with informative lines

By "**Onto-Read**" button the function for reading objects (definitions) from the OntoArM archive can be activated. In the automated mode, Onto-Read uses as input a file with subjects (concepts) and relations (each couple on a separate line) and extract from corresponded layer theirs objects (definitions). If any object does not exist, the output is empty.

Each subject (concept), its relation and object (definition) occupy one record in the output file. There is no limit to the number of records in the file. After pressing the "Onto-Read" button, the system reads concepts sequentially from the input file and for each of them:

(1) Transform the subject (concept) into path;

(2) Extract the definition of this concept from relation layer using the path to locate it.

The output file is in CSV file format. Its records have the next format:

<subject>;<relation>;<object><CR>.

In the panel next to the Onto-Read button, two informative lines are shown (Figure 95):

- Total time used for extracting of all instances;
- Average time used for extracting of one instance, in ticks (milliseconds).

The time used is highly dependent on possibilities of operational environment and speed of computer hardware.

In the case of the Figure 95, 112945 instances were extracted for 89950 milliseconds and one instance has been extracted for average time of 0.80 milliseconds. In other words, more than one thousand instances are processed for less than one second.

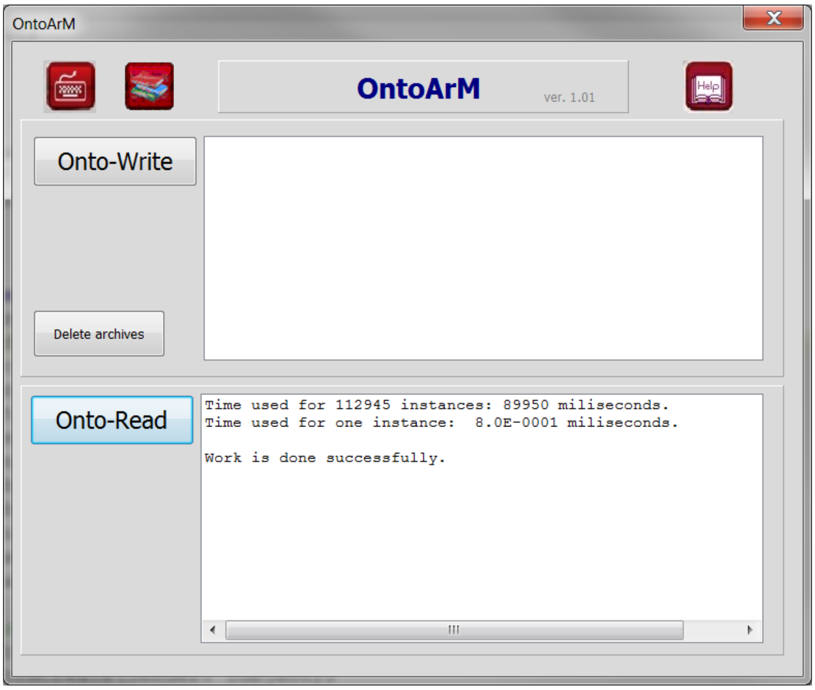

Figure 95. Content of OntoArM Onto-Read panel with informative lines

The form has three service buttons:

– The first () serves as a transition to the form for manual input and output of data to/from the system archive;
– The second () is connected to the module for adjusting the environment of the system – archives, input and output information, etc.;
– The third () activates the help text (user guide) of the system.

In the same panel, there is a button which enables deleting the work archives of the OntoArM (for test control in this version, they are stored on the hard disk but not in the computer main memory). OntoArM is completed with compressing program and after storing the information prepares small archive for long time storage.

➢ ***OntoArM manual mode functions***

The OntoArM panel for working in manual mode is shown on Figure 96.

By "**Onto-Write**" button the function for storing RDF-triples can be activated.

Each subject (concept) and its relation and object (definition) can be given in corresponded fields on the screen form (Figure 96).

After pressing the "Onto-Write" button, the system reads information from the fields and:

(1) Transform the subject (concept) into path;

(2) Store the object (definition) of this subject (concept) in the container located by the path in the layer pointed by the relation.

By "**Onto-Read**" button the function for reading RDF-objects (definitions) from the OntoArM archive can be activated. In the manual mode, Onto-Read uses as input the subject (concept) given in the screen field and extract from the archive its definition. If the definition does not exist, the output is empty definition.

There are two possibilities:

(1) To extract object from concrete layer given by corresponded relation;

(2) To extract from all layers the objects which correspond to given subject.

After pressing the "Onto-Read" button, the system reads concept from the screen field and:

(1) Transform the concept into path;

(2) Extract the object (definition) of this concept from the container located by the path in the given layer or from all layers (if the relation is replaced by an asterisk "*"; see the request in Figure 97 and the result in Figure 98).

The form for manual work has three service buttons.

- The first ( ) serves as a return to the form for automatic input and output of data to/from the system archive;
- The second ( ) is connected to the module for adjusting the environment of the system – archives, input and output information, etc.;
- The third ( ) activates the help text (user guide) of the system.

The exit from the system can be done by the conventional way for Windows - by clicking on the cross in the upper right corner of the form.

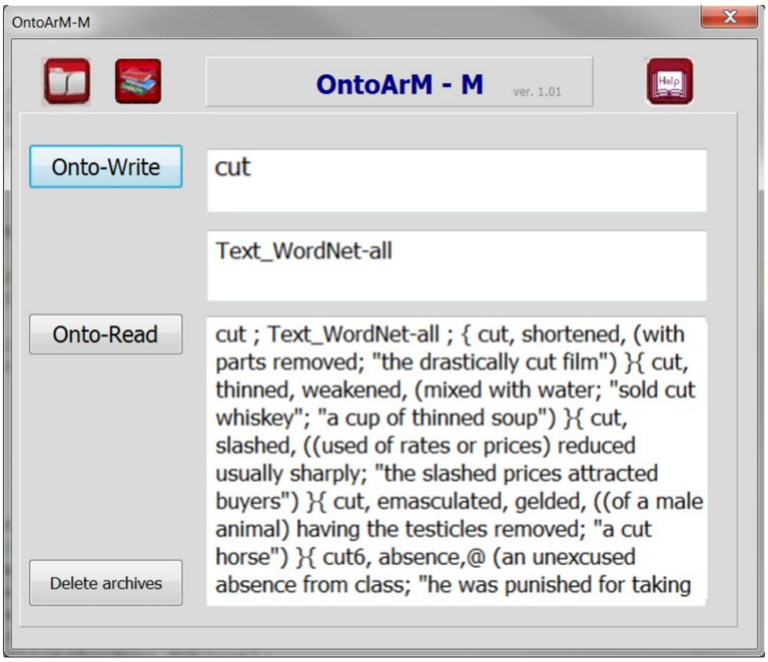

Figure 96. Manual input of the RDF-triple

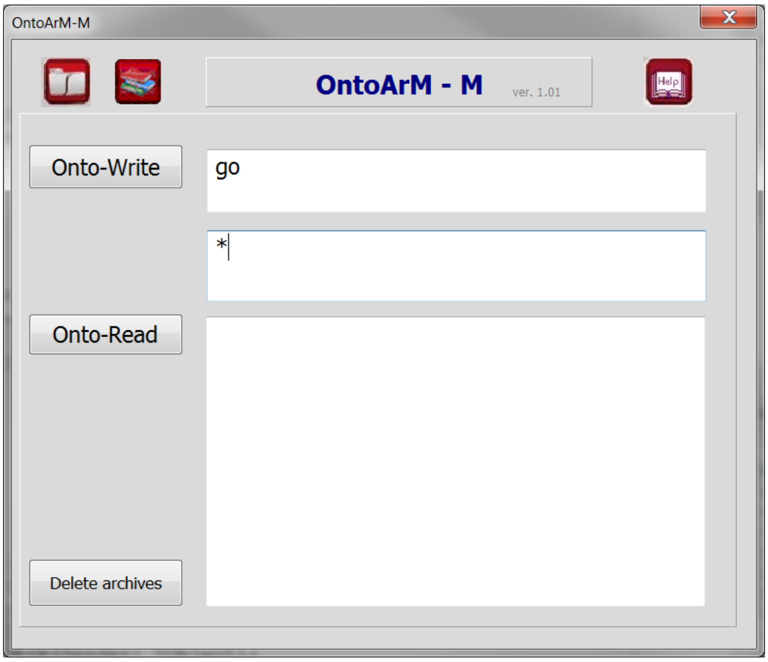

Figure 97. Manual reading the RDF-triple

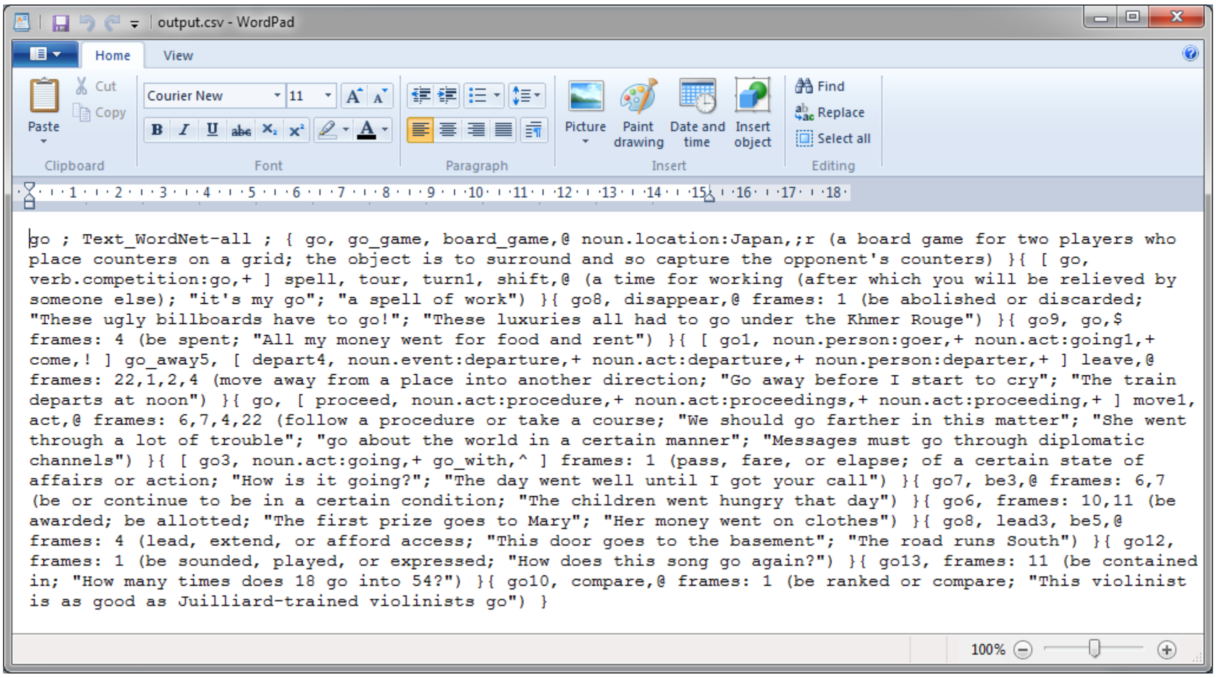

Figure 98. A part from reading from all layers

A3. RDFArM

RDFArM is a system for storing large sets of RDF triples and quadruples through natural language addressing.

RDFArM is upgrade over Natural Language Addressing Access Method and corresponded Archive Manager called **NL-ArM**, realized in this research. RDFArM is aimed to store archives of RDF triples and quadruples in multi-layer hash tables (information spaces with variable size). Each RDF element can be stored by appropriate path, which is set by a natural language word or phrase.

In RDFArM, the length of RDF element (string) can vary from 0 to 1G bytes. There is no limit on the number of strings in an archive, but their total length plus internal indexes may not exceed the capacity of the file system for one file (length of 4G, 8G, etc.). There is no limit on the number of files in the data base, as well as their location, including the Internet.

The data of RDFArM are encoded in N-Triples or N-Quads format.

The N-Quads is a format that extends N-Triples with context. Each triple in an N-Quads document can have an optional context value [N-Quads, 2013]:

<subject> <predicate> <object> <context>.

as opposed to N-Triples, where each triple has the form:

<subject> <predicate> <object>.

The main idea for storing RDF-graphs in RDFArM follows the one of multi-layer representation. In other words, the RDF-relations are assumed as layers and the RDF-subjects are assumed as paths valid for all layers. The objects as well as contexts are stored in the containers located by the path in the corresponded layers. Due to great number of relations – about several thousand – using separated files for layers is not effective. In this research we have proposed special algorithm for representing the layers.

For easy reading below we reproduce main algorithms of RDFArM.

- ***Algorithm for storing based on NL-addressing***

1. Read a quadruple from input file.
2. Assign unique numbers to the <subject>, <predicate>, <object>, and <context>, respectively denoted by NS, NP, NO, and NC. The algorithm of this step is given below.
3. Store the structures:
   - {NO; NC} in the "object" index archive using the path (NS, NP);
   - {NS; NC} in the "subject" index archive using the path (NP, NO);
   - {NP; NC} in the "predicate" index archive using the path (NS, NO).
4. Repeat from 1 until there are new quadruples, i.e. till end of file.
5. Stop.

➢ ***Algorithm for assigning unique numbers***

1. A separate counters for the <subject>, <predicate>, <object>, and <context> are used. Counters start from 1.
2. A separate NL-archives for the <subject>, <predicate>, <object>, and <context> are used.
3. In every NL-archive, using the values of respectively <subject>, <predicate>, <object>, and <context> as paths:

   IF no counter value exist at the corresponded path

   THAN

   - Store value of corresponded counter in the container located by the path;
   - Store the content of <subject>, <predicate>, <object>, or <context> respectively in corresponded data archive in hash table 1 (domain 1) using the value of the counter as path;
   - Increment the corresponded counter by 1.

   ELSE assign the existing value of counter as number of NS, NP, NO, and NC, respectively.
4. Return

➢ ***Algorithm for reading based on NL-addressing***

1. Read the request from screen form or file. The request may contain a part of the elements of the quadruple. Missing elements are requested to be found.
2. From every NL-archive, using the values of given respectively <subject>, <predicate>, <object>, or <context> as NL-addresses read the values of corresponded counters NS, NP, NO, or NC.
3. If the corresponded co-ordinate couple exist, read the structures:
   - {NO; NC} from the "object" index archive using path (NS, NP);
   - {NS; NC} from the "subject" index archive using path (NP, NO);
   - {NP; NC} from the "predicate" index archive using path (NS, NO).
4. **IF** all elements of the set {NS, NP, NO, NC} are given:

   THAN using the set {NS, NP, NO, NC} read the quadruple elements (from corresponded data archives).

   ELSE using given values of the elements of the set {NS, NP, NO, NC} scan all possible values of the unknown elements to reconstruct the set {NS, NP, NO, NC}. The result contains all possible quadruples for the requested values.
5. End.

No search indexes are needed and no recompilation of the data base is required after any update or adding new information in the data base.

A screenshot from the RDFArM program is shown at Figure 99.

The main functions are RDF-Write and RDF-Read for which there are corresponded buttons.

By "**RDF-Write**" button the function for storing RDF triples or quadruples from a file can be activated. The recognition of the case (triples or quadruples) is made automatically. The lines of triples do not contain the fourth element, i.e. the context of the quadruples.

Each triple (subject, relation, and object) or quadruple (subject, relation, object, and context) occupy one record in the input file. There is no limit to the number of records in the file. After pressing the "RDF-Write" button, the system reads records sequentially from the file and for each of them executes the algorithms given above.

The input file is in the next formats:

<subject> <relation> <object> .<CR>

or

<subject> <relation> <object> <context> .<CR>.

After storing the triples or quadruples, RDFArM displays two informative lines in the panel near to the "RDF-Write" button (Figure 99):

- Total time used for storing all instances from the file;
- Average time used for storing of one instance, in milliseconds.

The time used is highly dependent on the possibilities of the operational environment and the speed of the computer hardware.

In the case of the Figure 99, 15472624 quadruple instances were stored for 63437758 milliseconds and one instance has been stored for average time of 4.1 milliseconds. In other words, about 250 quadruples were processed for about less than one second.

By "**RDF-Read**" button the function for reading RDF triples or quadruples from the RDFArM archives can be activated. RDF-Read uses as input a file with requests similar to SPARQL requests and extracts from the archives the requested information. For example, the same input file as for RDF-Write may be used as file with request. The missing elements may be given by <?>.

In other words, if any of parameters are not given, i.e. any from <subject>, <predicate>, <object>, or <context>, as in SPARQL requests, the rest are used as constant addresses and omitted parameters scan all non empty co-ordinates for given position. This way all possible requests like (?S-?P-?O), (S-P-?O), (S-?P-O), (?S-P-O), etc., are covered (S stands for subject, P for property, O for object). For more information about SPARQL see [SPARQL, 2013] as well as short outline of it at the end of Appendix B.

Each extracted triple or quadruple occupies one record in the output file. There is no limit to the number of records in the file. After pressing the "RDF-Read" button, the system reads requests sequentially from the input file and for each of them executes the algorithm given above.

The output file has the next formats:

- for quadruples:

<subject><relation><object><context> . <CR>

- for triples:

<subject><relation><object> . <CR>

In the window next to the RDF-Read button, two informative lines are shown (Figure 100):

- Total time used for extracting of all quadruple instances;
- Average time used for extracting of one instance in milliseconds.

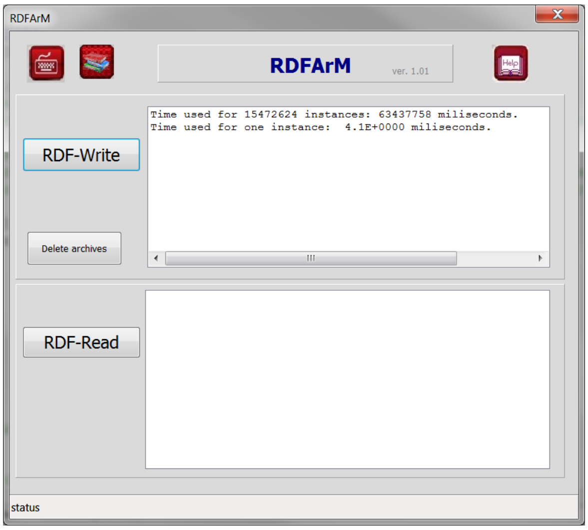

Figure 99. Content of RDFArM RDF-Write panel with informative lines

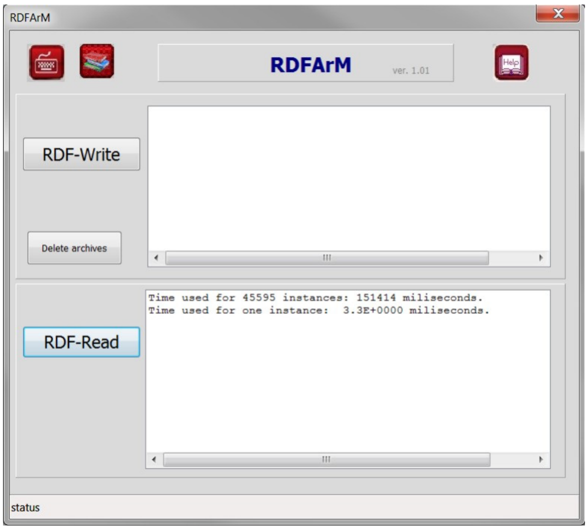

Figure 100. Content of RDFArM RDF-Read panel with informative lines

The time used is highly dependent on possibilities of operational environment and speed of computer hardware.

In the case of the Figure 100, 45595 quadruple instances were extracted for 151414 milliseconds and one instance has been extracted for average time of 3.3 milliseconds. In other words, about 330 quadruple instances are processed for about one second.

The RDFArN form (Figure 99 or Figure 100) has three service buttons:

- The first () serves as a transition to the form for manual input and output of data to/from the system archive (not realized in this version of RDFArM);
- The second () is connected to the module for adjusting the environment of the system – archives, input and output information, etc.;
- The third () activates the help text (user guide) of the system.

In the same panel there is a button which enables deleting the work archives of the RDFArM (for test control in this version, they are stored on the hard disk but not in the computer memory). RDFArM is completed with compressing program and after storing the information prepares small archive for long time storage.

A4. Results from experiment with simulating parallel processing

Table 70. RDFArM loading results for infoboxes-fixed.nt

check point	triples stored	ms for all	ms for one	ms for last 100000	ms for one	counted to the check point number of: Subjects	Relations	Objects
Processor 1								
1	100000	218011	2.2	218011	2.2	8420	4081	100000
2	200000	437848	2.2	219837	2.2	16803	5280	200000
3	300000	653457	2.2	215609	2.2	24675	6179	300000
4	400000	876008	2.2	222551	2.2	32383	6988	400000
5	500000	1103489	2.2	227481	2.3	40883	7569	500000
6	600000	1315541	2.2	212052	2.1	46320	7828	600000
7	700000	1526704	2.2	211163	2.1	51519	7998	700000
8	800000	1738694	2.2	211990	2.1	57213	8070	800000
9	900000	1954147	2.2	215453	2.2	62640	8104	900000
10	1000000	2185356	2.2	231209	2.3	68897	8152	1000000
11	1100000	2420652	2.2	235296	2.4	74653	8171	1100000
12	1200000	2699223	2.2	278571	2.8	82531	8459	1200000
13	1300000	2976328	2.3	277105	2.8	91373	8838	1300000
14	1400000	3235976	2.3	259648	2.6	99051	9245	1400000
15	1500000	3504266	2.3	268290	2.7	107697	9661	1500000
16	1600000	3782322	2.4	278056	2.8	116246	10018	1600000
17	1700000	4059848	2.4	277526	2.8	124483	10298	1700000
18	1800000	4334956	2.4	275108	2.8	133475	10559	1800000
19	1900000	4610547	2.4	275591	2.8	142046	10857	1900000
20	2000000	4886794	2.4	276247	2.8	150909	11132	2000000
21	2100000	5170326	2.5	283532	2.8	159404	11380	2100000
22	2200000	5461751	2.5	291425	2.9	168023	11566	2200000
23	2300000	5767841	2.5	306090	3.1	177475	11979	2300000
24	2400000	6073447	2.5	305606	3.1	185859	12313	2400000
25	2500000	6388803	2.6	315356	3.2	194303	12543	2500000
26	2600000	6692209	2.6	303406	3.0	202175	12740	2600000
27	2700000	6989734	2.6	297525	3.0	210116	12963	2700000
28	2800000	7276324	2.6	286590	2.9	218060	13165	2800000
29	2900000	7564520	2.6	288196	2.9	226962	13369	2900000
30	3000000	7824885	2.6	260365	2.6	236327	13511	3000000
31	3100000	8073052	2.6	248167	2.5	244449	13846	3100000
32	3200000	8343589	2.6	270537	2.7	251980	14103	3200000

check point	triples stored	ms for all	ms for one	ms for last 100000	ms for one	counted to the check point number of: Subjects	Relations	Objects
33	3300000	8608697	2.6	265108	2.7	259160	14293	3300000
34	3400000	8883415	2.6	274718	2.7	267094	14440	3400000
35	3500000	9156884	2.6	273469	2.7	274781	14565	3500000
36	3600000	9432569	2.6	275685	2.8	282159	14721	3600000
37	3700000	9699503	2.6	266934	2.7	290531	14823	3700000
38	3800000	9983502	2.6	283999	2.8	298560	14947	3800000
39	3900000	10268516	2.6	285014	2.9	307578	15247	3900000
40	4000000	10551097	2.6	282581	2.8	317286	15427	4000000
41	4100000	10832382	2.6	281285	2.8	326106	15545	4100000
42	4200000	11112294	2.6	279912	2.8	334027	15651	4200000
43	4300000	11386591	2.6	274297	2.7	341882	15792	4300000
44	4400000	11668797	2.7	282206	2.8	349800	15901	4400000
45	4500000	11960940	2.7	292143	2.9	357571	16018	4500000
46	4600000	12250431	2.7	289491	2.9	365372	16120	4600000
47	4700000	12533791	2.7	283360	2.8	372637	16256	4700000
48	4800000	12824577	2.7	290786	2.9	380369	16456	4800000
49	4900000	13109404	2.7	284827	2.8	388418	16624	4900000
50	5000000	13394043	2.7	284639	2.8	396155	16714	5000000
Processor 2								
51	5100000	13608873	2.7	214939	2.1	404619	20417	5100000
52	5200000	13845510	2.7	236637	2.4	412889	21532	5200000
53	5300000	14059700	2.7	214190	2.1	421384	22454	5300000
54	5400000	14277415	2.6	217715	2.2	430157	23178	5400000
55	5500000	14500028	2.6	222613	2.2	438947	23785	5500000
56	5600000	14722252	2.6	222224	2.2	447721	24271	5600000
57	5700000	14968702	2.6	246450	2.5	456764	24617	5700000
58	5800000	15225823	2.6	257121	2.6	465311	25118	5800000
59	5900000	15499511	2.6	273688	2.7	473764	25660	5900000
60	6000000	15785648	2.6	286137	2.9	482970	25993	6000000
61	6100000	16066840	2.6	281192	2.8	491804	26256	6100000
62	6200000	16351401	2.6	284561	2.8	500115	26527	6200000
63	6300000	16656602	2.6	305201	3.1	508784	26839	6300000
64	6400000	16952083	2.6	295481	3.0	519796	27093	6400000
65	6500000	17212028	2.6	259945	2.6	537865	27242	6500000
66	6600000	17519740	2.7	307712	3.1	546098	27462	6600000

check point	triples stored	ms for all	ms for one	ms for last 100000	ms for one	counted to the check point number of: Subjects	Relations	Objects
67	6700000	17812850	2.7	293110	2.9	553493	27617	6700000
68	6800000	18116209	2.7	303359	3.0	565582	27722	6800000
69	6900000	18415076	2.7	298867	3.0	576688	27953	6900000
70	7000000	18694879	2.7	279803	2.8	591877	28139	7000000
71	7100000	18980658	2.7	285779	2.9	599777	28268	7100000
72	7200000	19291833	2.7	311175	3.1	607957	28428	7200000
73	7300000	19605442	2.7	313609	3.1	616114	28582	7300000
74	7400000	19915104	2.7	309662	3.1	624151	28872	7400000
75	7500000	20222457	2.7	307353	3.1	631947	29101	7500000
76	7600000	20539420	2.7	316963	3.2	639630	29275	7600000
77	7700000	20791190	2.7	251770	2.5	643433	29345	7700000
78	7800000	21049075	2.7	257885	2.6	648774	29464	7800000
79	7900000	21335774	2.7	286699	2.9	659720	29550	7900000
80	8000000	21680255	2.7	344481	3.4	668261	29814	8000000
81	8100000	22008590	2.7	328335	3.3	676543	29967	8100000
82	8200000	22324056	2.7	315466	3.2	683679	30111	8200000
83	8300000	22669426	2.7	345370	3.5	692084	30346	8300000
84	8400000	23015046	2.7	345620	3.5	700571	30829	8400000
85	8500000	23351587	2.7	336541	3.4	709278	30915	8500000
86	8600000	23682075	2.8	330488	3.3	717808	31150	8600000
87	8700000	23980349	2.8	298274	3.0	731008	31248	8700000
88	8800000	24315424	2.8	335075	3.4	739380	31336	8800000
89	8900000	24665412	2.8	349988	3.5	747578	31452	8900000
90	9000000	25012717	2.8	347305	3.5	755601	31627	9000000
91	9100000	25349087	2.8	336370	3.4	763840	31740	9100000
92	9200000	25689465	2.8	340378	3.4	771909	31818	9200000
93	9300000	26027160	2.8	337695	3.4	779697	31971	9300000
94	9400000	26381642	2.8	354482	3.5	788584	32073	9400000
95	9500000	26735904	2.8	354262	3.5	796082	32188	9500000
96	9600000	27075877	2.8	339973	3.4	804137	32282	9600000
97	9700000	27429313	2.8	353436	3.5	813123	32345	9700000
98	9800000	27785963	2.8	356650	3.6	821841	32493	9800000

check point	triples stored	ms for all	ms for one	ms for last 100000	ms for one	counted to the check point number of: Subjects	Relations	Objects
99	9900000	28109571	2.8	323608	3.2	836180	32625	9900000
100	10000000	28449029	2.8	339458	3.4	847168	32681	10000000
Processor 3								
101	10100000	28672907	2.8	223909	2.2	858184	35627	10100000
102	10200000	28891308	2.8	218401	2.2	865025	36877	10200000
103	10300000	29101176	2.8	209868	2.1	870357	37444	10300000
104	10400000	29323883	2.8	222707	2.2	879121	38221	10400000
105	10500000	29544703	2.8	220820	2.2	887773	38742	10500000
106	10600000	29766006	2.8	221303	2.2	896372	39170	10600000
107	10700000	30002581	2.8	236575	2.4	904348	39430	10700000
108	10800000	30251855	2.8	249274	2.5	912712	39720	10800000
109	10900000	30512704	2.8	260849	2.6	921471	40159	10900000
110	11000000	30782695	2.8	269991	2.7	930652	40430	11000000
111	11100000	31077740	2.8	295045	3.0	938759	40783	11100000
112	11200000	31364485	2.8	286745	2.9	947817	41214	11200000
113	11300000	31634523	2.8	270038	2.7	957333	41485	11300000
114	11400000	31924966	2.8	290443	2.9	966208	42005	11400000
115	11500000	32224893	2.8	299927	3.0	975275	42305	11500000
116	11600000	32534368	2.8	309475	3.1	984634	42584	11600000
117	11700000	32821644	2.8	287276	2.9	993400	43438	11700000
118	11800000	33111307	2.8	289663	2.9	1001889	43728	11800000
119	11900000	33410080	2.8	298773	3.0	1010671	44353	11900000
120	12000000	33723891	2.8	313811	3.1	1019361	44873	12000000
121	12100000	34040495	2.8	316604	3.2	1030942	45035	12100000
122	12200000	34291719	2.8	251224	2.5	1053185	45057	12200000
123	12300000	34570322	2.8	278603	2.8	1071104	45145	12300000
124	12400000	34831733	2.8	261411	2.6	1090954	45326	12400000
125	12500000	35155528	2.8	323795	3.2	1103013	45534	12500000
126	12600000	35515298	2.8	359770	3.6	1111404	45749	12600000
127	12700000	35838688	2.8	323390	3.2	1125568	45882	12700000
128	12800000	36174262	2.8	335574	3.4	1135662	46061	12800000

check point	triples stored	ms for all	ms for one	ms for last 100000	ms for one	counted to the check point number of: Subjects	Relations	Objects
129	12900000	36516980	2.8	342718	3.4	1143829	46202	12900000
130	13000000	36851633	2.8	334653	3.3	1155053	46375	13000000
131	13100000	37190530	2.8	338897	3.4	1163060	46497	13100000
132	13200000	37513842	2.8	323312	3.2	1172871	46687	13200000
133	13300000	37818403	2.8	304561	3.0	1185111	46800	13300000
134	13400000	38167299	2.8	348896	3.5	1194422	46985	13400000
135	13500000	38429006	2.8	261707	2.6	1200001	47120	13500000
136	13600000	38641183	2.8	212177	2.1	1202148	47126	13600000
137	13700000	38849850	2.8	208667	2.1	1204022	47171	13700000
138	13800000	39066458	2.8	216608	2.2	1206950	47248	13800000
139	13900000	39290569	2.8	224111	2.2	1211272	47435	13900000
140	14000000	39561121	2.8	270552	2.7	1218686	47552	14000000
141	14100000	39861252	2.8	300131	3.0	1226371	47731	14100000
142	14200000	40182317	2.8	321065	3.2	1234818	47945	14200000
143	14300000	40496269	2.8	313952	3.1	1243563	48114	14300000
144	14400000	40821407	2.8	325138	3.3	1252499	48274	14400000
145	14500000	41137028	2.8	315621	3.2	1261448	48400	14500000
146	14600000	41448265	2.8	311237	3.1	1271270	48483	14600000
147	14700000	41747366	2.8	299101	3.0	1280957	48629	14700000
148	14800000	42012958	2.8	265592	2.7	1297124	48680	14800000
149	14900000	42321497	2.8	308539	3.1	1306316	48760	14900000
150	15000000	42631221	2.8	309724	3.1	1314612	48889	15000000
Processor 4								
151	15100000	42852181	2.8	221038	2.2	1323411	52220	15100000
152	15200000	43071503	2.8	219322	2.2	1331462	54515	15200000
153	15300000	43284865	2.8	213362	2.1	1339737	55400	15300000
154	15400000	43499897	2.8	215032	2.2	1348229	56049	15400000
155	15472624	43652528	2.8	152631	2.1	1354298	56338	15472624
total	**15472624**	**43652528**	**2.8**			**1354298**	**56338**	**15472624**

A5. Results from experiment with 100 millions triples

Table 71 contains results from an experiment for loading 100 millions triples from BSBM 100M [BSBMv3, 2009].

The check points were on every 100000 triples.

For every check point, the average time in ms for writing one triple is shown. In third column the corresponded value of log n is given.

Table 71. Comparison of NLArM storing time and log n for 100 millions triples

triples	ms	log n
100000	2.5	16.61
200000	2.6	17.61
300000	2.4	18.19
400000	2.2	18.61
500000	2.2	18.93
600000	2.3	19.19
700000	2.3	19.42
800000	2.3	19.61
900000	2.3	19.78
1000000	2.3	19.93
1100000	2.3	20.07
1200000	2.2	20.19
1300000	2.2	20.31
1400000	2.2	20.42
1500000	2.2	20.52
1600000	2.2	20.61
1700000	2.2	20.70
1800000	2.2	20.78
1900000	2.2	20.86
2000000	2.2	20.93
2100000	2.1	21.00
2200000	2.2	21.07
2300000	2.2	21.13
2400000	2.2	21.19
2500000	2.2	21.25
2600000	2.3	21.31
2700000	2.2	21.36
2800000	2.3	21.42
2900000	2.2	21.47
3000000	2.2	21.52
3100000	2.2	21.56
3200000	2.2	21.61
3300000	2.2	21.65
3400000	2.2	21.70
3500000	2.3	21.74
3600000	2.3	21.78
3700000	2.3	21.82
3800000	2.2	21.86
3900000	2.3	21.90
4000000	2.3	21.93
4100000	2.2	21.97
4200000	2.3	22.00
4300000	2.3	22.04
4400000	2.3	22.07
4500000	2.2	22.10
4600000	2.3	22.13
4700000	2.2	22.16
4800000	2.3	22.19
4900000	2.3	22.22
5000000	2.3	22.25
5100000	2.3	22.28
5200000	2.3	22.31
5300000	2.2	22.34
5400000	2.2	22.36
5500000	2.3	22.39
5600000	2.2	22.42
5700000	2.2	22.44
5800000	2.3	22.47
5900000	2.2	22.49
6000000	2.3	22.52
6100000	2.3	22.54
6200000	2.2	22.56
6300000	2.3	22.59
6400000	2.2	22.61
6500000	2.2	22.63
6600000	2.2	22.65
6700000	2.3	22.68
6800000	2.2	22.70
6900000	2.2	22.72
7000000	2.2	22.74
7100000	2.2	22.76
7200000	2.2	22.78
7300000	2.2	22.80
7400000	2.3	22.82
7500000	2.2	22.84
7600000	2.4	22.86
7700000	2.3	22.88
7800000	2.3	22.90
7900000	2.2	22.91
8000000	2.3	22.93
8100000	2.3	22.95
8200000	2.2	22.97
8300000	2.3	22.98
8400000	2.2	23.00
8500000	2.2	23.02
8600000	2.2	23.04
8700000	2.2	23.05
8800000	2.3	23.07
8900000	2.2	23.09
9000000	2.3	23.10
9100000	2.3	23.12
9200000	2.3	23.13
9300000	2.2	23.15
9400000	2.2	23.16
9500000	2.3	23.18
9600000	2.2	23.19
9700000	2.2	23.21
9800000	2.3	23.22
9900000	2.3	23.24
10000000	2.2	23.25
10100000	2.3	23.27
10200000	2.3	23.28
10300000	2.3	23.30
10400000	2.3	23.31
10500000	2.3	23.32

triples	ms	log n
10600000	2.3	23.34
10700000	2.2	23.35
10800000	2.3	23.36
10900000	2.2	23.38
11000000	2.3	23.39
11100000	2.2	23.40
11200000	2.2	23.42
11300000	2.2	23.43
11400000	2.2	23.44
11500000	2.4	23.46
11600000	2.3	23.47
11700000	2.4	23.48
11800000	2.4	23.49
11900000	2.3	23.50
12000000	2.3	23.52
12100000	2.4	23.53
12200000	2.3	23.54
12300000	2.4	23.55
12400000	2.3	23.56
12500000	2.3	23.58
12600000	2.3	23.59
12700000	2.3	23.60
12800000	2.4	23.61
12900000	2.3	23.62
13000000	2.4	23.63
13100000	2.3	23.64
13200000	2.4	23.65
13300000	2.3	23.66
13400000	2.3	23.68
13500000	2.3	23.69
13600000	2.3	23.70
13700000	2.4	23.71
13800000	2.3	23.72
13900000	2.4	23.73
14000000	2.3	23.74
14100000	2.3	23.75
14200000	2.3	23.76
14300000	2.3	23.77
14400000	2.3	23.78
14500000	2.3	23.79
14600000	2.2	23.80
14700000	2.2	23.81
14800000	2.3	23.82
14900000	2.3	23.83
15000000	2.3	23.84
15100000	2.4	23.85
15200000	2.3	23.86
15300000	2.3	23.87
15400000	2.3	23.88
15500000	2.3	23.89
15600000	2.3	23.90

triples	ms	log n
15700000	2.2	23.90
15800000	2.3	23.91
15900000	2.2	23.92
16000000	2.3	23.93
16100000	2.2	23.94
16200000	2.3	23.95
16300000	2.3	23.96
16400000	2.3	23.97
16500000	2.3	23.98
16600000	2.3	23.98
16700000	2.2	23.99
16800000	2.3	24.00
16900000	2.3	24.01
17000000	2.4	24.02
17100000	2.3	24.03
17200000	2.3	24.04
17300000	2.3	24.04
17400000	2.2	24.05
17500000	2.2	24.06
17600000	2.2	24.07
17700000	2.3	24.08
17800000	2.2	24.09
17900000	2.2	24.09
18000000	2.2	24.10
18100000	2.3	24.11
18200000	2.2	24.12
18300000	2.2	24.13
18400000	2.3	24.13
18500000	2.3	24.14
18600000	2.3	24.15
18700000	2.3	24.16
18800000	2.4	24.16
18900000	2.4	24.17
19000000	2.2	24.18
19100000	2.3	24.19
19200000	2.2	24.19
19300000	2.2	24.20
19400000	2.2	24.21
19500000	2.2	24.22
19600000	2.2	24.22
19700000	2.3	24.23
19800000	2.2	24.24
19900000	2.2	24.25
20000000	2.2	24.25
20100000	2.3	24.26
20200000	2.3	24.27
20300000	2.2	24.27
20400000	2.3	24.28
20500000	2.3	24.29
20600000	2.3	24.30
20700000	2.3	24.30

triples	ms	log n
20800000	2.3	24.31
20900000	2.4	24.32
21000000	2.3	24.32
21100000	2.2	24.33
21200000	2.2	24.34
21300000	2.2	24.34
21400000	2.2	24.35
21500000	2.3	24.36
21600000	2.3	24.36
21700000	2.4	24.37
21800000	2.3	24.38
21900000	2.3	24.38
22000000	2.3	24.39
22100000	2.4	24.40
22200000	2.2	24.40
22300000	2.2	24.41
22400000	2.3	24.42
22500000	2.3	24.42
22600000	2.2	24.43
22700000	2.3	24.44
22800000	2.4	24.44
22900000	2.3	24.45
23000000	2.3	24.46
23100000	2.4	24.46
23200000	2.3	24.47
23300000	2.4	24.47
23400000	2.2	24.48
23500000	2.2	24.49
23600000	2.3	24.49
23700000	2.3	24.50
23800000	2.3	24.50
23900000	2.4	24.51
24000000	2.4	24.52
24100000	2.2	24.52
24200000	2.3	24.53
24300000	2.4	24.53
24400000	2.4	24.54
24500000	2.3	24.55
24600000	2.3	24.55
24700000	2.3	24.56
24800000	2.3	24.56
24900000	2.4	24.57
25000000	2.4	24.58
25100000	2.3	24.58
25200000	2.4	24.59
25300000	2.4	24.59
25400000	2.4	24.60
25500000	2.4	24.60
25600000	2.3	24.61
25700000	2.4	24.62
25800000	2.3	24.62

triples	ms	log n
25900000	2.2	24.63
26000000	2.2	24.63
26100000	2.3	24.64
26200000	2.3	24.64
26300000	2.4	24.65
26400000	2.3	24.65
26500000	2.5	24.66
26600000	2.4	24.66
26700000	2.3	24.67
26800000	2.3	24.68
26900000	2.3	24.68
27000000	2.3	24.69
27100000	2.3	24.69
27200000	2.4	24.70
27300000	2.2	24.70
27400000	2.2	24.71
27500000	2.3	24.71
27600000	2.3	24.72
27700000	2.3	24.72
27800000	2.3	24.73
27900000	2.4	24.73
28000000	2.4	24.74
28100000	2.3	24.74
28200000	2.3	24.75
28300000	2.3	24.75
28400000	2.3	24.76
28500000	2.3	24.76
28600000	2.3	24.77
28700000	2.3	24.77
28800000	2.3	24.78
28900000	2.2	24.78
29000000	2.2	24.79
29100000	2.3	24.79
29200000	2.3	24.80
29300000	2.3	24.80
29400000	2.4	24.81
29500000	2.3	24.81
29600000	2.3	24.82
29700000	2.4	24.82
29800000	2.3	24.83
29900000	2.3	24.83
30000000	2.3	24.84
30100000	2.3	24.84
30200000	2.3	24.85
30300000	2.4	24.85
30400000	2.4	24.86
30500000	2.2	24.86
30600000	2.3	24.87
30700000	2.2	24.87
30800000	2.2	24.88
30900000	2.3	24.88

triples	ms	log n
31000000	2.3	24.89
31100000	2.3	24.89
31200000	2.4	24.90
31300000	2.4	24.90
31400000	2.3	24.90
31500000	2.3	24.91
31600000	2.4	24.91
31700000	2.3	24.92
31800000	2.3	24.92
31900000	2.4	24.93
32000000	2.3	24.93
32100000	2.3	24.94
32200000	2.3	24.94
32300000	2.3	24.95
32400000	2.3	24.95
32500000	2.4	24.95
32600000	2.3	24.96
32700000	2.3	24.96
32800000	2.3	24.97
32900000	2.3	24.97
33000000	2.3	24.98
33100000	2.3	24.98
33200000	2.3	24.98
33300000	2.2	24.99
33400000	2.3	24.99
33500000	2.4	25.00
33600000	2.3	25.00
33700000	2.3	25.01
33800000	2.4	25.01
33900000	2.3	25.01
34000000	2.3	25.02
34100000	2.4	25.02
34200000	2.3	25.03
34300000	2.3	25.03
34400000	2.3	25.04
34500000	2.3	25.04
34600000	2.3	25.04
34700000	2.2	25.05
34800000	2.3	25.05
34900000	2.2	25.06
35000000	2.3	25.06
35100000	2.3	25.06
35200000	2.3	25.07
35300000	2.3	25.07
35400000	2.3	25.08
35500000	2.3	25.08
35600000	2.3	25.09
35700000	2.4	25.09
35800000	2.3	25.09
35900000	2.4	25.10
36000000	2.3	25.10

triples	ms	log n
36100000	2.3	25.11
36200000	2.3	25.11
36300000	2.3	25.11
36400000	2.4	25.12
36500000	2.4	25.12
36600000	2.3	25.13
36700000	2.3	25.13
36800000	2.3	25.13
36900000	2.3	25.14
37000000	2.4	25.14
37100000	2.3	25.14
37200000	2.3	25.15
37300000	2.3	25.15
37400000	2.3	25.16
37500000	2.3	25.16
37600000	2.3	25.16
37700000	2.3	25.17
37800000	2.3	25.17
37900000	2.3	25.18
38000000	2.3	25.18
38100000	2.3	25.18
38200000	2.3	25.19
38300000	2.3	25.19
38400000	2.3	25.19
38500000	2.3	25.20
38600000	2.3	25.20
38700000	2.3	25.21
38800000	2.3	25.21
38900000	2.3	25.21
39000000	2.3	25.22
39100000	2.2	25.22
39200000	2.3	25.22
39300000	2.3	25.23
39400000	2.3	25.23
39500000	2.3	25.24
39600000	2.4	25.24
39700000	2.4	25.24
39800000	2.3	25.25
39900000	2.3	25.25
40000000	2.3	25.25
40100000	2.4	25.26
40200000	2.3	25.26
40300000	2.2	25.26
40400000	2.4	25.27
40500000	2.4	25.27
40600000	2.3	25.27
40700000	2.4	25.28
40800000	2.4	25.28
40900000	2.3	25.29
41000000	2.3	25.29
41100000	2.3	25.29

triples	ms	log n
41200000	2.3	25.30
41300000	2.4	25.30
41400000	2.3	25.30
41500000	2.3	25.31
41600000	2.4	25.31
41700000	2.3	25.31
41800000	2.3	25.32
41900000	2.4	25.32
42000000	2.3	25.32
42100000	2.4	25.33
42200000	2.3	25.33
42300000	2.3	25.33
42400000	2.3	25.34
42500000	2.3	25.34
42600000	2.4	25.34
42700000	2.4	25.35
42800000	2.3	25.35
42900000	2.3	25.35
43000000	2.3	25.36
43100000	2.3	25.36
43200000	2.3	25.36
43300000	2.3	25.37
43400000	2.2	25.37
43500000	2.3	25.37
43600000	2.3	25.38
43700000	2.3	25.38
43800000	2.3	25.38
43900000	2.3	25.39
44000000	2.3	25.39
44100000	2.3	25.39
44200000	2.2	25.40
44300000	2.3	25.40
44400000	2.3	25.40
44500000	2.2	25.41
44600000	2.3	25.41
44700000	2.3	25.41
44800000	2.3	25.42
44900000	2.3	25.42
45000000	2.3	25.42
45100000	2.3	25.43
45200000	2.4	25.43
45300000	2.4	25.43
45400000	2.3	25.44
45500000	2.3	25.44
45600000	2.3	25.44
45700000	2.3	25.45
45800000	2.3	25.45
45900000	2.3	25.45
46000000	2.3	25.46
46100000	2.3	25.46
46200000	2.3	25.46

triples	ms	log n
46300000	2.3	25.46
46400000	2.3	25.47
46500000	2.3	25.47
46600000	2.3	25.47
46700000	2.3	25.48
46800000	2.3	25.48
46900000	2.3	25.48
47000000	2.3	25.49
47100000	2.4	25.49
47200000	2.3	25.49
47300000	2.3	25.50
47400000	2.4	25.50
47500000	2.3	25.50
47600000	2.3	25.50
47700000	2.3	25.51
47800000	2.3	25.51
47900000	2.3	25.51
48000000	2.4	25.52
48100000	2.3	25.52
48200000	2.3	25.52
48300000	2.3	25.53
48400000	2.3	25.53
48500000	2.3	25.53
48600000	2.3	25.53
48700000	2.3	25.54
48800000	2.3	25.54
48900000	2.3	25.54
49000000	2.3	25.55
49100000	2.3	25.55
49200000	2.3	25.55
49300000	2.3	25.56
49400000	2.3	25.56
49500000	2.3	25.56
49600000	2.3	25.56
49700000	2.4	25.57
49800000	2.5	25.57
49900000	2.5	25.57
50000000	2.3	25.58
50100000	2.4	25.58
50200000	2.3	25.58
50300000	2.3	25.58
50400000	2.4	25.59
50500000	2.3	25.59
50600000	2.3	25.59
50700000	2.4	25.60
50800000	2.3	25.60
50900000	2.3	25.60
51000000	2.3	25.60
51100000	2.3	25.61
51200000	2.3	25.61
51300000	2.3	25.61

triples	ms	log n
51400000	2.3	25.62
51500000	2.4	25.62
51600000	2.3	25.62
51700000	2.3	25.62
51800000	2.3	25.63
51900000	2.3	25.63
52000000	2.3	25.63
52100000	2.3	25.63
52200000	2.3	25.64
52300000	2.3	25.64
52400000	2.3	25.64
52500000	2.2	25.65
52600000	2.3	25.65
52700000	2.3	25.65
52800000	2.3	25.65
52900000	2.3	25.66
53000000	2.2	25.66
53100000	2.3	25.66
53200000	2.3	25.66
53300000	2.3	25.67
53400000	2.2	25.67
53500000	2.3	25.67
53600000	2.2	25.68
53700000	2.3	25.68
53800000	2.3	25.68
53900000	2.3	25.68
54000000	2.3	25.69
54100000	2.3	25.69
54200000	2.3	25.69
54300000	2.1	25.69
54400000	2.2	25.70
54500000	2.3	25.70
54600000	2.2	25.70
54700000	2.3	25.71
54800000	2.2	25.71
54900000	2.3	25.71
55000000	2.3	25.71
55100000	2.2	25.72
55200000	2.4	25.72
55300000	2.3	25.72
55400000	2.3	25.72
55500000	2.3	25.73
55600000	2.3	25.73
55700000	2.3	25.73
55800000	2.3	25.73
55900000	2.3	25.74
56000000	2.3	25.74
56100000	2.4	25.74
56200000	2.3	25.74
56300000	2.3	25.75
56400000	2.3	25.75

triples	ms	log n
56500000	2.3	25.75
56600000	2.3	25.75
56700000	2.3	25.76
56800000	2.4	25.76
56900000	2.3	25.76
57000000	2.3	25.76
57100000	2.2	25.77
57200000	2.3	25.77
57300000	2.3	25.77
57400000	2.3	25.77
57500000	2.3	25.78
57600000	2.3	25.78
57700000	2.2	25.78
57800000	2.3	25.78
57900000	2.3	25.79
58000000	2.3	25.79
58100000	2.3	25.79
58200000	2.3	25.79
58300000	2.2	25.80
58400000	2.3	25.80
58500000	2.3	25.80
58600000	2.3	25.80
58700000	2.3	25.81
58800000	2.3	25.81
58900000	2.3	25.81
59000000	2.3	25.81
59100000	2.3	25.82
59200000	2.3	25.82
59300000	2.3	25.82
59400000	2.3	25.82
59500000	2.3	25.83
59600000	2.3	25.83
59700000	2.3	25.83
59800000	2.2	25.83
59900000	2.3	25.84
60000000	2.3	25.84
60100000	2.3	25.84
60200000	2.3	25.84
60300000	2.3	25.85
60400000	2.3	25.85
60500000	2.3	25.85
60600000	2.4	25.85
60700000	2.3	25.86
60800000	2.4	25.86
60900000	2.3	25.86
61000000	2.3	25.86
61100000	2.4	25.86
61200000	2.3	25.87
61300000	2.3	25.87
61400000	2.3	25.87
61500000	2.4	25.87

triples	ms	log n
61600000	2.3	25.88
61700000	2.3	25.88
61800000	2.3	25.88
61900000	2.4	25.88
62000000	2.3	25.89
62100000	2.4	25.89
62200000	2.3	25.89
62300000	2.3	25.89
62400000	2.3	25.90
62500000	2.3	25.90
62600000	2.3	25.90
62700000	2.3	25.90
62800000	2.3	25.90
62900000	2.3	25.91
63000000	2.4	25.91
63100000	2.4	25.91
63200000	2.3	25.91
63300000	2.3	25.92
63400000	2.3	25.92
63500000	2.4	25.92
63600000	2.3	25.92
63700000	2.3	25.92
63800000	2.3	25.93
63900000	2.3	25.93
64000000	2.3	25.93
64100000	2.3	25.93
64200000	2.3	25.94
64300000	2.3	25.94
64400000	2.3	25.94
64500000	2.3	25.94
64600000	2.3	25.95
64700000	2.3	25.95
64800000	2.3	25.95
64900000	2.3	25.95
65000000	2.2	25.95
65100000	2.3	25.96
65200000	2.3	25.96
65300000	2.3	25.96
65400000	2.3	25.96
65500000	2.2	25.96
65600000	2.3	25.97
65700000	2.3	25.97
65800000	2.3	25.97
65900000	2.3	25.97
66000000	2.3	25.98
66100000	2.3	25.98
66200000	2.4	25.98
66300000	2.3	25.98
66400000	2.3	25.98
66500000	2.3	25.99
66600000	2.3	25.99

triples	ms	log n
66700000	2.3	25.99
66800000	2.3	25.99
66900000	2.3	26.00
67000000	2.4	26.00
67100000	2.3	26.00
67200000	2.3	26.00
67300000	2.4	26.00
67400000	2.3	26.01
67500000	2.3	26.01
67600000	2.3	26.01
67700000	2.3	26.01
67800000	2.3	26.01
67900000	2.3	26.02
68000000	2.3	26.02
68100000	2.3	26.02
68200000	2.3	26.02
68300000	2.3	26.03
68400000	2.3	26.03
68500000	2.3	26.03
68600000	2.3	26.03
68700000	2.3	26.03
68800000	2.3	26.04
68900000	2.3	26.04
69000000	2.4	26.04
69100000	2.4	26.04
69200000	2.3	26.04
69300000	2.4	26.05
69400000	2.4	26.05
69500000	2.3	26.05
69600000	2.4	26.05
69700000	2.3	26.05
69800000	2.3	26.06
69900000	2.3	26.06
70000000	2.3	26.06
70100000	2.3	26.06
70200000	2.3	26.06
70300000	2.3	26.07
70400000	2.3	26.07
70500000	2.3	26.07
70600000	2.3	26.07
70700000	2.2	26.08
70800000	2.3	26.08
70900000	2.3	26.08
71000000	2.2	26.08
71100000	2.2	26.08
71200000	2.2	26.09
71300000	2.3	26.09
71400000	2.2	26.09
71500000	2.3	26.09
71600000	2.3	26.09
71700000	2.3	26.10

triples	ms	log n
71800000	2.2	26.10
71900000	2.2	26.10
72000000	2.3	26.10
72100000	2.2	26.10
72200000	2.3	26.11
72300000	2.3	26.11
72400000	2.3	26.11
72500000	2.3	26.11
72600000	2.2	26.11
72700000	2.3	26.12
72800000	2.2	26.12
72900000	2.3	26.12
73000000	2.3	26.12
73100000	2.3	26.12
73200000	2.4	26.13
73300000	2.3	26.13
73400000	2.3	26.13
73500000	2.4	26.13
73600000	2.4	26.13
73700000	2.4	26.14
73800000	2.3	26.14
73900000	2.3	26.14
74000000	2.4	26.14
74100000	2.4	26.14
74200000	2.4	26.14
74300000	2.4	26.15
74400000	2.4	26.15
74500000	2.4	26.15
74600000	2.5	26.15
74700000	2.4	26.15
74800000	2.5	26.16
74900000	2.4	26.16
75000000	2.4	26.16
75100000	2.4	26.16
75200000	2.4	26.16
75300000	2.5	26.17
75400000	2.4	26.17
75500000	2.4	26.17
75600000	2.4	26.17
75700000	2.4	26.17
75800000	2.5	26.18
75900000	2.4	26.18
76000000	2.3	26.18
76100000	2.3	26.18
76200000	2.3	26.18
76300000	2.3	26.19
76400000	2.2	26.19
76500000	2.2	26.19
76600000	2.2	26.19
76700000	2.2	26.19
76800000	2.2	26.19

triples	ms	log n
76900000	2.3	26.20
77000000	2.3	26.20
77100000	2.2	26.20
77200000	2.2	26.20
77300000	2.2	26.20
77400000	2.2	26.21
77500000	2.3	26.21
77600000	2.3	26.21
77700000	2.3	26.21
77800000	2.2	26.21
77900000	2.2	26.22
78000000	2.2	26.22
78100000	2.2	26.22
78200000	2.3	26.22
78300000	2.2	26.22
78400000	2.2	26.22
78500000	2.2	26.23
78600000	2.3	26.23
78700000	2.2	26.23
78800000	2.2	26.23
78900000	2.3	26.23
79000000	2.3	26.24
79100000	2.3	26.24
79200000	2.2	26.24
79300000	2.2	26.24
79400000	2.3	26.24
79500000	2.3	26.24
79600000	2.2	26.25
79700000	2.2	26.25
79800000	2.2	26.25
79900000	2.2	26.25
80000000	2.2	26.25
80100000	2.2	26.26
80200000	2.3	26.26
80300000	2.2	26.26
80400000	2.2	26.26
80500000	2.3	26.26
80600000	2.2	26.26
80700000	2.3	26.27
80800000	2.3	26.27
80900000	2.4	26.27
81000000	2.3	26.27
81100000	2.3	26.27
81200000	2.2	26.27
81300000	2.3	26.28
81400000	2.2	26.28
81500000	2.3	26.28
81600000	2.3	26.28
81700000	2.3	26.28
81800000	2.4	26.29
81900000	2.3	26.29

triples	ms	log n
82000000	2.2	26.29
82100000	2.3	26.29
82200000	2.3	26.29
82300000	2.2	26.29
82400000	2.2	26.30
82500000	2.3	26.30
82600000	2.2	26.30
82700000	2.3	26.30
82800000	2.3	26.30
82900000	2.3	26.30
83000000	2.3	26.31
83100000	2.2	26.31
83200000	2.3	26.31
83300000	2.3	26.31
83400000	2.3	26.31
83500000	2.3	26.32
83600000	2.3	26.32
83700000	2.3	26.32
83800000	2.2	26.32
83900000	2.2	26.32
84000000	2.2	26.32
84100000	2.3	26.33
84200000	2.3	26.33
84300000	2.3	26.33
84400000	2.3	26.33
84500000	2.3	26.33
84600000	2.2	26.33
84700000	2.3	26.34
84800000	2.3	26.34
84900000	2.4	26.34
85000000	2.3	26.34
85100000	2.3	26.34
85200000	2.3	26.34
85300000	2.2	26.35
85400000	2.2	26.35
85500000	2.2	26.35
85600000	2.3	26.35
85700000	2.3	26.35
85800000	2.3	26.35
85900000	2.2	26.36
86000000	2.3	26.36
86100000	2.3	26.36
86200000	2.2	26.36
86300000	2.3	26.36
86400000	2.2	26.36
86500000	2.2	26.37
86600000	2.3	26.37
86700000	2.3	26.37
86800000	2.2	26.37
86900000	2.2	26.37
87000000	2.2	26.37

triples	ms	log n
87100000	2.3	26.38
87200000	2.3	26.38
87300000	2.2	26.38
87400000	2.3	26.38
87500000	2.3	26.38
87600000	2.3	26.38
87700000	2.3	26.39
87800000	2.3	26.39
87900000	2.2	26.39
88000000	2.2	26.39
88100000	2.2	26.39
88200000	2.3	26.39
88300000	2.3	26.40
88400000	2.3	26.40
88500000	2.3	26.40
88600000	2.3	26.40
88700000	2.2	26.40
88800000	2.3	26.40
88900000	2.3	26.41
89000000	2.3	26.41
89100000	2.3	26.41
89200000	2.3	26.41
89300000	2.3	26.41
89400000	2.3	26.41
89500000	2.3	26.42
89600000	2.3	26.42
89700000	2.3	26.42
89800000	2.3	26.42
89900000	2.2	26.42
90000000	2.2	26.42
90100000	2.3	26.43
90200000	2.3	26.43
90300000	2.3	26.43
90400000	2.3	26.43
90500000	2.3	26.43
90600000	2.4	26.43
90700000	2.3	26.43
90800000	2.3	26.44
90900000	2.3	26.44
91000000	2.3	26.44
91100000	2.3	26.44
91200000	2.2	26.44
91300000	2.3	26.44
91400000	2.3	26.45

triples	ms	log n
91500000	2.2	26.45
91600000	2.3	26.45
91700000	2.3	26.45
91800000	2.3	26.45
91900000	2.3	26.45
92000000	2.4	26.46
92100000	2.3	26.46
92200000	2.3	26.46
92300000	2.3	26.46
92400000	2.2	26.46
92500000	2.3	26.46
92600000	2.2	26.46
92700000	2.3	26.47
92800000	2.3	26.47
92900000	2.3	26.47
93000000	2.2	26.47
93100000	2.3	26.47
93200000	2.2	26.47
93300000	2.3	26.48
93400000	2.3	26.48
93500000	2.3	26.48
93600000	2.3	26.48
93700000	2.3	26.48
93800000	2.3	26.48
93900000	2.2	26.48
94000000	2.3	26.49
94100000	2.3	26.49
94200000	2.3	26.49
94300000	2.3	26.49
94400000	2.3	26.49
94500000	2.4	26.49
94600000	2.2	26.50
94700000	2.3	26.50
94800000	2.3	26.50
94900000	2.3	26.50
95000000	2.3	26.50
95100000	2.4	26.50
95200000	2.3	26.50
95300000	2.4	26.51
95400000	2.3	26.51
95500000	2.3	26.51
95600000	2.3	26.51
95700000	2.3	26.51
95800000	2.2	26.51

triples	ms	log n
95900000	2.2	26.52
96000000	2.3	26.52
96100000	2.3	26.52
96200000	2.3	26.52
96300000	2.3	26.52
96400000	2.3	26.52
96500000	2.3	26.52
96600000	2.2	26.53
96700000	2.3	26.53
96800000	2.3	26.53
96900000	2.3	26.53
97000000	2.2	26.53
97100000	2.3	26.53
97200000	2.3	26.53
97300000	2.2	26.54
97400000	2.2	26.54
97500000	2.2	26.54
97600000	2.3	26.54
97700000	2.4	26.54
97800000	2.3	26.54
97900000	2.3	26.54
98000000	2.2	26.55
98100000	2.3	26.55
98200000	2.3	26.55
98300000	2.3	26.55
98400000	2.3	26.55
98500000	2.3	26.55
98600000	2.3	26.56
98700000	2.2	26.56
98800000	2.3	26.56
98900000	2.2	26.56
99000000	2.2	26.56
99100000	2.2	26.56
99200000	2.3	26.56
99300000	2.2	26.57
99400000	2.3	26.57
99500000	2.3	26.57
99600000	2.3	26.57
99700000	2.4	26.57
99800000	2.3	26.57
99900000	2.3	26.57
100000000	2.3	26.58
100000112	2.4	26.58

A6. Instruments for the programmers

Every access method is a brick in the whole program system building. Because of this it is important to ensure apparatus for using its possibilities by the programmers.

For natural language addressing there are several functions which serve its main features.

At the first place this is the function for converting the natural language text in the path. The sample code in Object Pascal is presented at Figure 101.

```
function string2coords(ssbeg : string) : TCoordArray;
var  ss1 : string;
      ll, ll2, ii, kk, coord : cardinal;
begin
  result[0] := 0;
  ss1 := ssbeg;
  ll := length(ssbeg);
  if ll = 0 then exit;
  ll2 := ll mod 4;   //  2;  or  4; for UNICODE or ASCII
  if ll2 > 0
    then for ii:=1 to (4-ll2) do ss1 := ss1 + ' ';
  ll2 := length(ss1) div 4;
  result[0] := ll2;
  ii := 0;
  while ii < ll2 do
    begin
      inc(ii);
      coord := 0;
      kk := (ii-1) * 4 + 1;
      coord := coord + ord(ss1[kk]);
      coord := coord shl 8;
      coord := coord + ord(ss1[kk+1]);
      coord := coord shl 8;
      coord := coord + ord(ss1[kk+2]);
      coord := coord shl 8;
      coord := coord + ord(ss1[kk+3]);
      result[ii] := coord;
    end;
 end;   {stgring2coords}
```

Figure 101. A sample function for converting the natural language text in path

The next step is procedure for writing information using NL-addressing. A sample code of such procedure is presented in Figure 102. It is based on the ArM function for storing information using a co-ordinate array (WriteA).

```
Procedure NLAWrite (const Name_arch_dat, Name_csv : shortstring);
 var ff : TextFile;
      ArmD : TArm;
      ss_line : string;
      concept, buffer, ss, ss1 : shortstring;
      xx, starttime, endtime, ii : cardinal;
      ccss : TCoordArray;

begin
  ArmD := TArm.Create(Name_arch_dat, false, 'wrkd');
  assignfile(ff, Name_csv);
  reset(ff);

  while not eof(ff) do
    begin
      readln(ff, buffer);
      inc (ii);
      xx := pos(';', buffer);
      if xx > 0
        then begin
          concept := shortstring(copy(buffer, 1, xx - 1));
          concept := del_sb(concept);
          delete(buffer, 1, xx);
          xx := length (buffer);
          if xx > 0 then
            begin
              ccss := string2coords(concept);
              ArmD.WriteA(@ccss, buffer, xx+1);
            end;
        end;
    end;
  closefile(ff);
  ArmD.Free;
end;
```

Figure 102. A sample code of procedure for storing information using NL-addressing

A sample code of the reverse procedure for reading information using NL-addressing is presented in Figure 103. It is based on the ArM function for reading information using co-ordinate array (ReadA).

```
Procedure NLARead (const Name_arch_dat, Name_words, Name_csv : shortstring);
var ffw, ffcsv : TextFile;
    concept, concept_work, buffer : shortstring;
    ArmD : TArm;
    ss, ss1, ssq, ss_line : string;
    xx, yy, starttime, endtime, ii : cardinal;
    ccss : TCoordArray;
    ind_doc : array of cardinal;
```

```
begin
  ArmD := TArm.Create(Name_arch_dat, false, 'rwrkd');
  assignfile(ffw, Name_words);
  reset(ffw);
  assignfile(ffcsv, Name_csv);
  rewrite(ffcsv);

  while not eof(ffw) do
    begin
      readln(ffw, concept);
      yy := length(concept);
      inc (ii);
      if yy > 0
        then begin
          concept_work := del_sb(concept);
          if concept_work <> '' then
            begin
              ccss := string2coords(concept_work);
              xx := 255;
              ArmD.ReadA (@ccss, buffer, xx, 0);
              writeln (ffcsv, concept, ' ; ', buffer);
            end;
        end;
    end;

  closefile(ffw);
  closefile(ffcsv);
  ArmD.Free;
end;
```

Figure 103. A sample code of procedure for reading information using NL-addressing

The main function for NL-storing and accessing are built in separate executive files. The programmers need a function for executing these so-called ".exe" files. A sample such function is presented in Figure 104.

```
function CreateProcessAndWait(AppPath, AppParams: string; Visibility: word): DWord;
var
  SI: TStartupInfo;
  PI: TProcessInformation;
  Proc: THandle;
begin
  FillChar(SI, SizeOf(SI), 0);
  SI.cb := SizeOf(SI);
  SI.wShowWindow := Visibility;
  SI.dwFlags := STARTF_USEPOSITION;
  SI.dwX := 30;
  SI.dwY := 500;
  if not CreateProcess(Nil, PChar(AppPath+' '+ AppParams), Nil, Nil, False,
```

```
                  Normal_Priority_Class, Nil, Nil, SI, PI)
        then showmessage('Failed to execute program.' + inttostr(GetLastError));

       Proc := PI.hProcess;
       CloseHandle(PI.hThread);
       if WaitForSingleObject(Proc, Infinite) <> Wait_Failed then
         GetExitCodeProcess(Proc, Result);
       CloseHandle(Proc);
     end;
```

Figure 104. A sample function for executing a program

At the end, the same function can be realized using other programming languages like C++, Java, etc. For instance, in the Institute of Cybernetics V.M.Glushkov in Kiev, Ukraine were prepared Java interface modules (see Figure 105, Figure 106, and Figure 107).

```
//NLA-Write

private void NLAWriteActionPerformed(java.awt.event.ActionEvent evt)
{
  this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
  try
  {       // TODO add your handling code here:
    FileOutputStream fos1 = null;
          //CSVFileWrite = new File("write_doc.csv");
    try { fos1 = new FileOutputStream(CSVFileWrite); }
    catch (FileNotFoundException ex)
      { Logger.getLogger(SemanticMapping.class.getName()).log(Level.SEVERE, null, ex); }
    BufferedWriter writer1 = null;
    try  { writer1 = new BufferedWriter(new OutputStreamWriter(fos1, "windows-1251")); }
    catch (UnsupportedEncodingException ex)
       { Logger.getLogger(SemanticMapping.class.getName()).log(Level.SEVERE, null, ex); }
    for (int k = 0; k < ArrayTermsForNLAWrite.size(); k++)
    {     //System.out.println(model_.get(k).toString());
      try  { writer1.append(SelectedFileCanonicalPath + ";" + ArrayTermsForNLAWrite.get(k).toString() + ";");
             writer1.append("\n"); }
      catch (IOException ex) {Logger.getLogger(SemanticMapping.class.getName()).log(Level.SEVERE, null, ex);}
    }
    writer1.close();
    fos1.close();
  }
  catch (IOException ex) {  Logger.getLogger(SemanticMapping.class.getName()).log(Level.SEVERE, null, ex); }
```

Figure 105. A sample JAVA interface for NLAWrite program)

```
//NLA-Read

private void NLAReadActionPerformed(java.awt.event.ActionEvent evt) {
     // TODO add your handling code here:
     String S_NLAread;
```

```
        ArrayList ArrayNLAread = new ArrayList<String>();
        this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        Runtime r = Runtime.getRuntime();
        Process p = null;
        try {
           p = r.exec("NLAread.exe " + DataBaseFile.getAbsolutePath() + " rrr_doc.csv " + jTextField1.getText());
           // p = r.exec("wine NLAread.exe " + DataBaseFile.getAbsolutePath() + " rrr_doc.csv " +
jTextField1.getText());
           try { p.waitFor(); }
          catch (InterruptedException ex)
            { Logger.getLogger(SemanticMapping.class.getName()).log(Level.SEVERE, null, ex); }
        }
        catch (IOException ex)
          { Logger.getLogger(SemanticMapping.class.getName()).log(Level.SEVERE, null, ex);  }
        CSVFileRead = new File("rrr_doc.csv");
        try { br_NLAred = new BufferedReader(new InputStreamReader(
                                  new FileInputStream(CSVFileRead.getAbsolutePath()), ENCODING_WIN1251));  }
        catch (IOException ex)
            { Logger.getLogger(SemanticMapping.class.getName()).log(Level.SEVERE, null, ex); }
        try { while ((S_NLAread = br_NLAred.readLine()) != null)
             { ArrayNLAread.add(S_NLAread); ArrayNLAread.add("\n"); }   }
    catch (IOException ex) { Logger.getLogger(SemanticMapping.class.getName()).log(Level.SEVERE, null, ex); }
    jTextArea1.setText(ArrayNLAread.toString());
    ArrayNLAread.clear();
    this.setCursor(Cursor.getDefaultCursor());
 }
```

Figure 106. A sample JAVA interface for NLARead program

```
Runtime r = Runtime.getRuntime();
  Process p = null;
  try
  {      // NLAwrite.exe
    p = r.exec("NLAwrite.exe ArmIndDoc.Dat write_doc.csv");
        //PI Linux, Mac OS
        //p = r.exec("wine NLAwrite.exe ArmIndDoc.Dat write_doc.csv");
    StatusBar.setText("xxx ");
    try  { p.waitFor();  this.setCursor(Cursor.getDefaultCursor());  }
    catch (InterruptedException ex) { Logger.getLogger(exe_run.class.getName()).log(Level.SEVERE, null, ex); }
  }
  catch (IOException ex)  { Logger.getLogger(exe_run.class.getName()).log(Level.SEVERE, null, ex);  }
}

 ==========================================================================

String OSver = System.getProperty("os.name");
System.out.println("OS Version -->" + OSver);
if (OSver.startsWith("Win"))   { p = r.exec("NLAwrite.exe ArmIndDoc.Dat write_doc.csv"); }
else   { p = r.exec("wine NLAwrite.exe ArmIndDoc.Dat write_doc.csv"); }
```

Figure 107. A sample JAVA interface for executing a program

A7. ICON Ontological editor

Creating and editing domain ontologies in ICON is supported by its original ontological editor [Velychko & Prihodnyuk, 2013]. It is able to read and store ontologies in OWL, XML and NL-addressing formats.

Internal representation of ontologies in ICON ontological editor is based on Growing pyramidal networks [Gladun, 2003]. A visualization of such network is given on Figure 108.

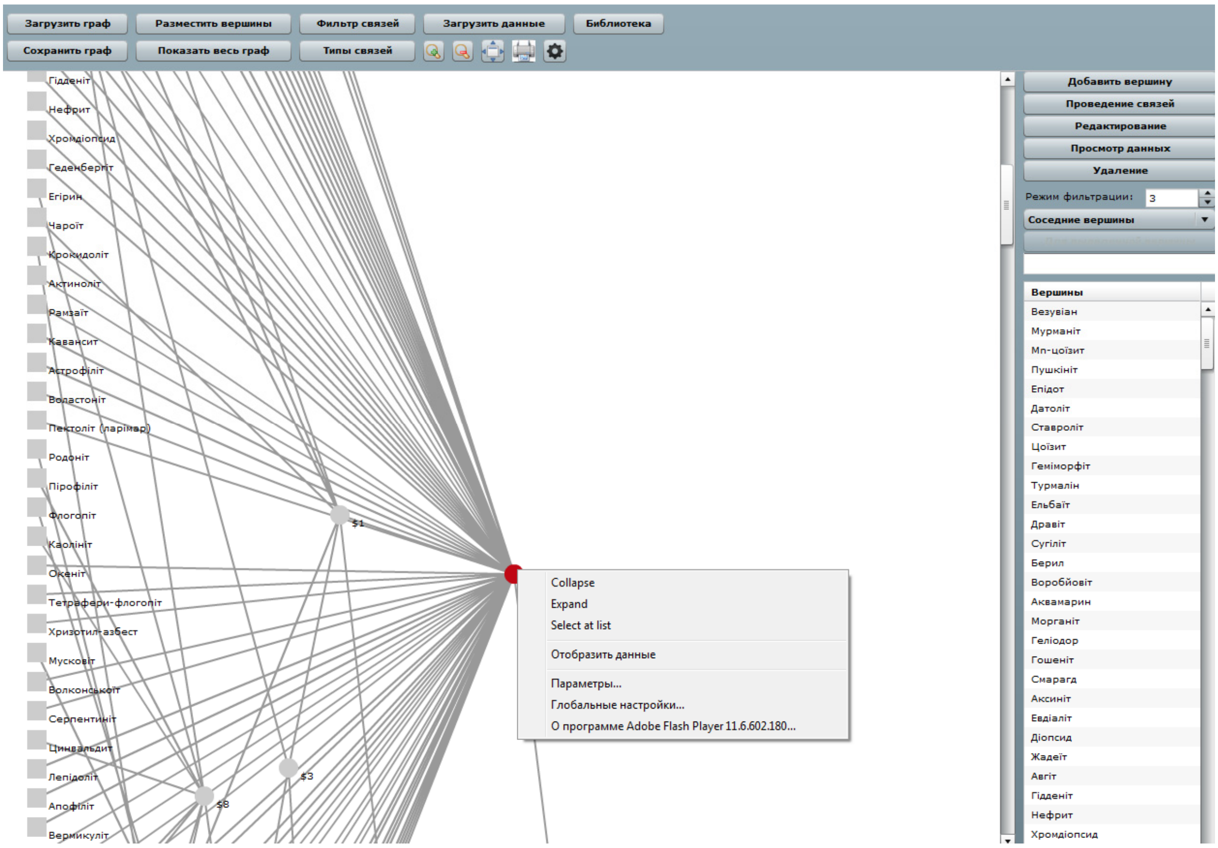

Figure 108. A visualization of a Growing pyramidal network

An example of the ontological graph generated by the ICON Ontological Editor is presented on Figure 109. This visualization of our sample graph (Figure 13) is created by this editor.

In Table 72 the corresponded ICON XML description of sample graph is given. It is generated automatically.

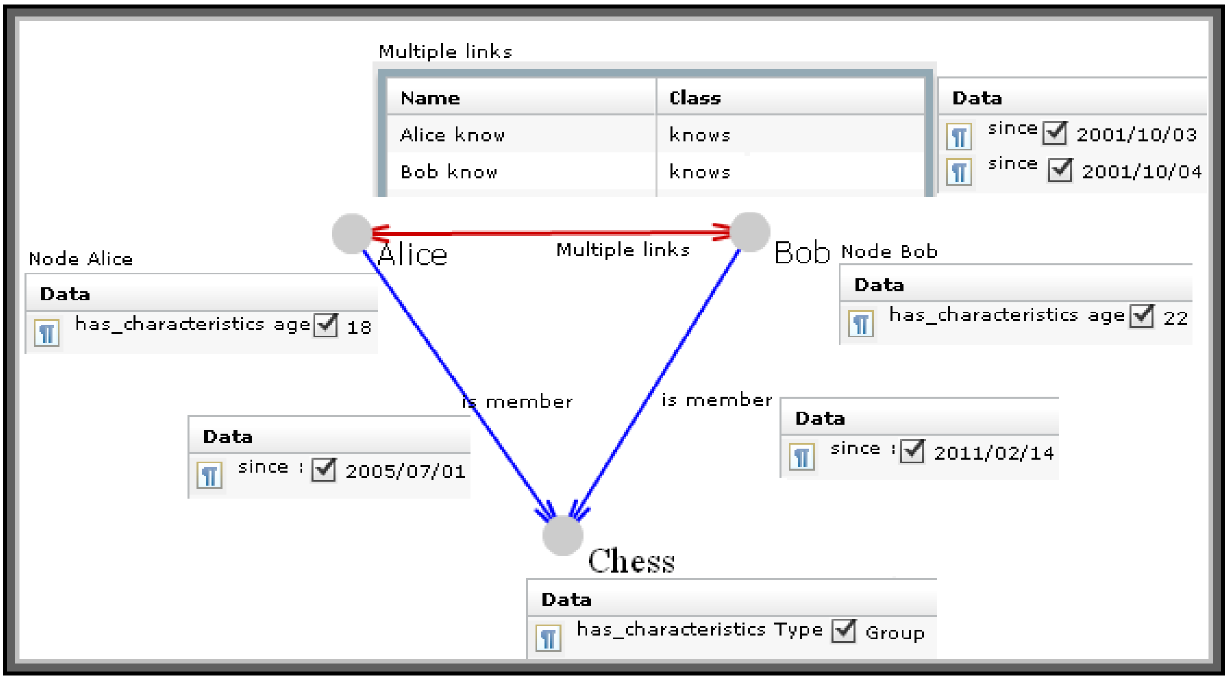

Figure 109. Screenshot from the ICON Ontological Editor

Table 72. XML description of the sample graph by ICON Ontological Editor

<Graph guid="31FFCF43-06A9-2F48-C2BC-CA5471D54392" >
<datagroups>
<datagroup>18</datagroup>
<datagroup>age</datagroup>
<datagroup>22</datagroup>
<datagroup>Group</datagroup>
<datagroup>2005/07/01</datagroup>
<datagroup>2001/10/03</datagroup>
<datagroup>2001/10/04</datagroup>
<datagroup>2011/02/14</datagroup>
</datagroups>
<Nodes>
<Node guid="CA5736C12F6A" nodeName="Alice" nclass="" shape="circle" color="13421772" xPos="455" yPos="223" font="Verdana" fontsize="16">
<data tclass="18" type="text">has_characteristics age</data>
</Node>
<Node guid="CA5E0ED51C35" nodeName="Bob" nclass="" shape="circle" color="13421772" xPos="681" yPos="220" font="Verdana" fontsize="16">
<data tclass="22" type="text">has_characteristics age</data>

```
    </Node>
    <Node guid="CA5F846D209F" nodeName="Chess" nclass="" shape="circle" color="13421772" xPos="552" yPos="397" font="Times New Roman" fontsize="20">
      <data tclass="Group" type="text">has_characteristics Type</data>
    </Node>
  </Nodes>
  <Linkgroups>
    <Group name="is member" color="255"/>
    <Group name="Default" color="10066329"/>
    <Group name="members" color="255"/>
    <Group name="knows" color="10092441"/>
  </Linkgroups>
  <Edges>
    <Edge guid="CA6CEE826F6F" edgeName="Alice know" node1="Alice" node2="Bob" group="knows" istwoway="true">
      <data tclass="2001/10/03" type="text">since</data>
    </Edge>
    <Edge guid="CA6E36B2C117" edgeName="Bob know" node1="Alice" node2="Bob" group="knows" istwoway="false">
      <data tclass="2001/10/04" type="text">since</data>
    </Edge>
    <Edge guid="CB6FF08FA087" edgeName="is member" node1="Alice" node2="Chess" group="is member" istwoway="false">
      <data tclass="2005/07/01" type="text">since :</data>
    </Edge>
    <Edge guid="CB70FD6BD805" edgeName="is member" node1="Bob" node2="Chess" group="is member" istwoway="false">
      <data tclass="2011/02/14" type="text">since :</data>
    </Edge>
  </Edges>
</Graph>
```

A8. Sample layers in ICON

Storing model chosen in ICON is multi-layer storing of ontology graph based on Natural Language Addressing. A sample list of layers used for storing common and local ontologies in ICON is presented in Table 73. It permits a preliminary evaluation of the number of layers needed for ICON at the project's first stage (about 50 up to 100).

Table 73. List of sample layers in ICON

Types	Layers (Relations)
Classification relations	• Class - Subclass. (genus-species) ("Organic compound - alcohol")
	• Element - Class. (element-set) ("Pet - cow")
	• Part - Whole. ("The wheel of the tractor")
	• Above - Below. ("Rector - Dean")
Attributive relations	• Object - Property
	• Object - Function
Comparison relations	• Association (object-object)
	• Incomparable. ("The weight of the object and the object's color are incomparable")
	• Comparable. ("The weight of the object and the weight of all parts of the object")
	• Equal. (Synonyms) ("All sides of an equilateral triangle are equal")
	• Greater than ("Turkey is greater than chicken")
	• Less than ("The density of ice is less than that of water").
Arrangement relations	• Be the following ("Ann came after John")
	• Be the next ("In the spring, it was the turn of the summer")
	• Be the nearest ("Zelenodolsk is the nearest town to the city of Kazan")
Modal relations	• Existence
	• Possibility ("The plane may take off ")
	• Necessity ("Five lorries are needed for the export of the crop")
	• Modifiers. ("It is desirable that you are not late for the start of the session")
Causal relations	• Purpose ("We want to climb the mountain")
	• Reason ("He violated his oath")

Types	Layers (Relations)
	• Cause - effect. ("Hot coal burned material")
Temporal relations	• Be at the same time. ("Jane and Elan came to the beginning of classes")
	• Be earlier ("The building was finished a month early.")
	• Be later on ("He come to studio an hour later on usual")
	• During the time interval ("During your stay in London we will visit the Royal Theater")
	• Start simultaneously. ("They start speaking at the same time")
	• Finish simultaneously. ("We finish our work at the same time you finish yours")
	• Coincide in time. ("Time of departure of aircraft and the train to Brussels – 19:00")
	• Overlap in time. ("The conferences overlap each other in two days")
Spatial relations	• Be on the left. ("The car stopped on the left of the tree")
	• Be on the right. ("On the right of the car there was a green tree")
	• Be in front. ("In front the teacher were two students")
	• Be at the back. ("The car stopped at the back of the house")
	• At the side. ("At the side of the road there is a lake.")
	• Touch. ("Clouds floated touching the roofs of houses")
	• To be on. ("The table is on the floor")
	• Be on top. ("They put the books on top of the bookshelf.")
	• Be below. ("Under the ice river flowed peacefully")
	• Be in. ("There were five people in the crew cabin.)
	• Intersect in space. ("The road intersects the forest.")
	• Coincide in space. ("Two conferences coincide in this building.")
Quantifiers	• Universal quantifier. ("All first-year students passed the exam on the programming.")
	• Existential quantifier. ("There exists at least one student who is able to solve the quadratic equation.")
Information relations	• Be sender. ("They submit the paper to the journal.”)
	• Be recipient. ("The editorial board received the paper.")
	• Be source of information. ("He told me that the order is ready.”)