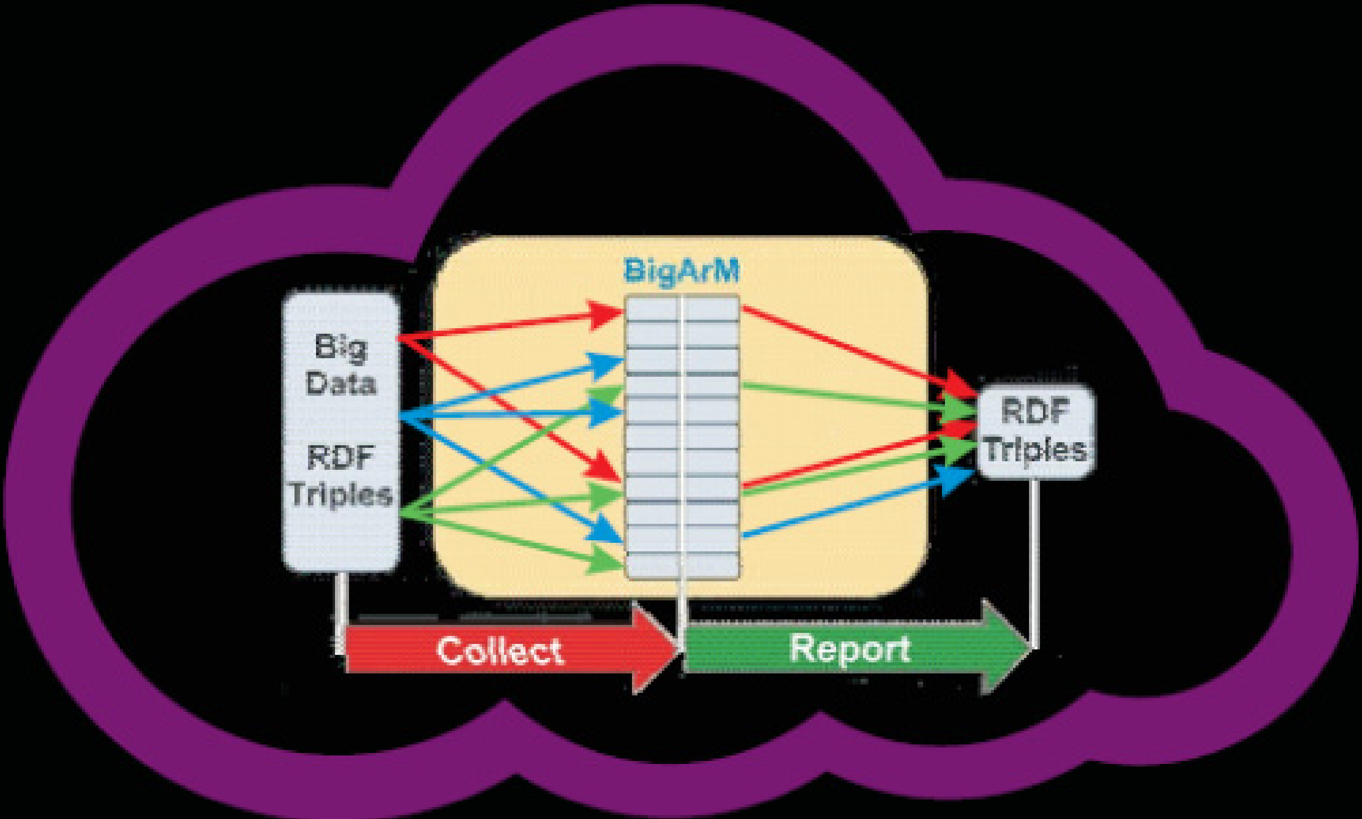
Krassimir Markov, Krassimira Ivanova, Vitalii Velychko
Koen Vanhoof, Juan Castellanos

Natural Language Addressing



ITHEA®

2015

**Krassimir Markov, Krassimira Ivanova, Vitalii Velychko,
Koen Vanhoof, Juan Castellanos**

Natural Language Addressing

ITHEA®
Sofia – Hasselt – Kyiv – Madrid
2015

Krassimir Markov, Krassimira Ivanova, Vitalii Velychko, Koen Vanhoof, Juan Castellanos
Natural Language Addressing

ITHEA® Sofia, Hasselt, Kyiv, Madrid
2015
ISBN: 978-954-16-0070-2 (printed)
ISBN: 978-954-16-0071-9 (online)
ITHEA IBS ISC No.: 33

First edition

Recommended for publication by The Scientific Council of the V.M.Glushkov Institute of Cybernetics of NAS, Ukraine (protocol No.: 7/28.04.2015), and The Scientific Council ITHEA Institute of Information Theories and Applications.

Reviewers:

Acad. Prof. DSci Alexander Palagin (V.M.Glushkov Institute of Cybernetics of NAS, Ukraine)
Prof. DSci Leonid Hulyanitskiy (V.M.Glushkov Institute of Cybernetics of NAS, Ukraine)
Prof. Assoc. Dr. Luis Fernando de Mingo Lopez (Universidad Politécnica de Madrid, Spain)
Prof. Assist. Dr. Benoit Depaire (Hasselt University, Belgium)

In this monograph, a new idea is proposed. It is called "Natural Language Addressing" (NLA). It is a possibility to access information using natural language words as paths to the information. For this purpose the internal encoding of the letters is used to generate corresponded path. This way it is possible to solve the problem of searching in big index structures by proposing a special kind of hashing, so-called "multi-layer hashing", i.e. by implementing recursively the same specialized hash function to build and resolve the collisions in hash tables.
Results presented in this work were implemented in practice for storing dictionaries, thesauruses, ontologies, and RDF-graphs.
It is represented that book articles will be interesting for experts in the field of information technologies as well as for practical users.

ISBN: 978-954-16-0070-2 (printed)
ISBN: 978-954-16-0071-9 (online)
C\o Jusautor, Sofia, 2015

Table of Contents

List of Figures

List of Tables

List of Acronyms

- ***Concepts***

ADT	Abstract data type
AIDA	Amdocs Intelligent Decision Automation
AKT	Advanced Knowledge Technologies
AM	Access Methods
ANSI SQL	American National Standards Institute standardized Structured Query Language
API	Application Programming Interface
ArM	Archive Manager based on MDIM
ArM32	32-bit realization of Archive Manager based on MDIM
ASCII	American Standard Code for Information Interchange
BSBM	Berlin SPARQL Benchmark
BSBM	Berlin SPARQL Bench Mark
BST	Binary Search Trees
BTC	Billion Triple Challenge
CI-BER	Cyber-infrastructure for a Billion Electronic Records
DAML+OIL	DARPA Agent Markup Language + Ontology Inference Layer
DB	DataBase
DB-models	Models for organization of the data in the DataBases
DBMS	Data Base Management System
DOD	Department of Defense
DVD	Dissociated Vertical Deviation
ebXML	Electronic Business using eXtensible Markup Language
EC	Electronic Collections
ER-model	Entity – Relationship model
GIS	Geographic Information Systems
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ID	IDentificator
IS	Information space
KIF	Knowledge Interchange Format
LDIF	Linked Data Integration Framework
MDAM	Multi-Dimensional Access Method

MDIM	Multi-Dimensional Information Model
MPGN	Multi-layer Pyramidal Growing Networks of information spaces
NARA	National Archives & Records Administration
NDM	Network Data Model
NLA	Natural Language Addressing
NL-access	Data Access using Natural Language Addressing
NL-Addressing	Natural Language Addressing
NL-archives	Data Base based on Natural Language Addressing
NLARead	Function of the module WordArM for accessing the information by Natural Language Addressing
NL-ArM	NL-addressing Archive Manager
NLAWrite	Function of the module WordArM for storing the information by Natural Language Addressing
NL-index	Index of Natural Language Addresses
NL-path	A path defined by natural language word or phrase
NL-storing	Storing based on Natural Language Addressing
NL-version	Realization based on Natural Language Addressing
NL-words	Natural Language Words
N-Quads	RDF quadruples
NSF	National Science Foundation
N-triples	RDF triples
ODBMS	Object DataBase Management Systems
OKBC	Open Knowledge Base Connectivity
OLAP	OnLine Analytical Processing
ORDBMS	Object-Relational DataBase Management Systems
OWL	Web Ontology Language
PC	Personal Computer
PHT	Perfect Hash Tables
RAM	Random Access Memory (main memory of the computer)
RDBMS	Relational DataBase Management Systems
RDF	Resource Description Framework
rdfDB	RDF Database
RDF-Read	Function of the module RDFArM for accessing the information by Natural Language Addressing
RDFS	RDF schema
RDF-Write	Function of the module RDFArM for accessing the information by Natural Language Addressing
RSO model	Relation-Subject-Object model
SD cards	Secure Digital cards
SGML	Standard Generalized Markup Language
SPARQL	SPARQL Protocol and RDF Query Language
SRO model	Subject-Relation-Object model
UML	Unified Modeling Language
UNICODE	Universal encoding standard
UNL model	Universal information model
URI	Uniform Resource Identifier

URLs	Uniform Resource Locators
USB	Universal Serial Bus
W3C	World Wide Web Consortium
XML	Extensible Markup Language

➢ ***Program tools***

AllegroGraph	High-performance persistent graph database
Apollo	Class system is modeled according to the OKBC
ArMSpeed	Specialized tool for NL-addressing of a document data base
Berkeley DB	A tool for storing RDF information
Chimaera	Software system that supports users in creating and maintaining distributed ontologies on the web
DOE	Differential Ontology Editor; simple ontology editor which allows the user to build ontologies according to the methodology
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering; the first module of the WonderWeb Foundational Ontologies Library (WFOL)
Firebird	A relational database that runs on Windows, Linux, and a variety of UNIX platforms
FrameNet	Lexical resource for English, based on frame semantics
GDM	Graph Data Model
GGL	Graph Database System for Genomics
GMOD	Graph-oriented Object Manipulation
GOAL	Graph-based Object and Association Language
GOOD	Graph Object Oriented Data Model
Gram	Graph Data Model and Query Language
Grinder	WordNet compiler for converting lexicographic format in to internal DB format
GROOVY	Graphically Represented Object-Oriented data model with Values
HyM	Hypernode Model
ICOM	Intelligent Conceptual Modeling
ICON	**I**nstrumental **C**omplex for **O**ntology designatio**N**
INFOS	**IN**telligence **FO**rmation **S**ystem
KAON	**Ka**rlsruhe **on**tology - open-source ontology management system targeted for business applications
K-Infinity	Knowledge editor broad support for object-oriented knowledge modeling
LDM	Logical Data Model
LinKFactory Workbench	Originally designed for very large medical ontologies
Medius Visual Ontology Modeller	UML-based ontology modeling tool
Mikrokosmos	Part of the Mikrokosmos knowledge-based machine translation system currently under development at the Computer Research Laboratory, New Mexico State University
OEM	Object Exchange Model
OilEd	Simple editor that allows the user to create and edit OIL ontologies
Omega	Terminological ontology constructed at USC ISI as the reorganization and

	synthesis of WordNet
OntoArM	Experimental module for storing ontologies (small RDF triple sets) using NL-addressing
OntoEdit	Built on top of a powerful internal ontology model
OntoIntegrator	Onto-linguistic research integrated environment for NLP using complicated structured ontological models
OntoLingua	Oriented toward the authoring of ontologies by assembling and extending ontologies obtained from the library
OSTP	Office of Science and Technology Policy
PaMaL	Object Oriented Pattern Matching Language
PROMT	Based on an extremely general knowledge model and therefore can be applied across various platforms
PropBank	Proposition Bank - focuses on the argument structure of verbs, and provides a complete corpus annotated with semantic roles, including roles traditionally viewed as arguments and as adjuncts
Protégé	Free, open source ontology editor and knowledge-base framework
RDFArM	Experimental module for storing large RDF triple or quadruple datasets
RDFArM-MP	Multi processor simulation of RDFArM work
RDFedt	Freeware module for creating complex Resource Description Framework (RDF) files
Sensus	Terminology taxonomy, as a framework into which additional knowledge can be placed
Sesame	An open-source framework for querying and analyzing RDF data
SMART	Self-Monitoring, Analysis and Reporting Technology; Based on an extremely general knowledge model and, therefore, can be applied across various platforms
SUMO	Suggested Upper Merged Ontology
WebODE	Extensible ontology-engineering suite based on an application server, whose development started in 1999 and whose support was discontinued in 2006
WebOnto	Java applet coupled with a customized web server which allows users to browse and edit knowledge models over the web
WordArM	Experimental module for storing thesauruses using NL-addressing
WordNet	Large lexical database of English
YARS	Yet Another RDF Store

This Page Intentionally Left Blank

Introduction

Large unstructured or semi-structured datasets require a high level of computational sophistication because operations that are easy at a small scale — such as moving data between machines or in and out of storage, visualizing the data, or displaying results —can all require substantial algorithmic ingenuity. As a data set becomes increasingly massive, it may be infeasible to gather it in one place and analyze it as a whole. Thus, there may be a need for algorithms that operate in a distributed fashion, analyzing subsets of the data and aggregating those results to understand the complete set. One aspect of this is the challenge of data assimilation, in which we wish to use new data to update model parameters without reanalyzing the entire data set. This is essential when new waves of data continue to arrive, or subsets are analyzed in isolation of one another, and one aims to improve the model and inferences in an adaptive fashion — for example, with streaming algorithms [NRC, 2013].

The White House Office of Science and Technology Policy (OSTP) — in concert with several Federal departments and agencies — created the "Big Data Research and Development Initiative" to:

- Advance state-of-the-art core technologies needed to collect, store, preserve, manage, analyze, and share huge quantities of data;
- Harness these technologies to accelerate the pace of discovery in science and engineering, strengthen national security, and transform teaching and learning;
- Expand the workforce needed to develop and use Big Data technologies.

By improving the ability to extract knowledge and insights from large and complex collections of digital data, the initiative promises to help solve some the Nation's most pressing challenges [BIG DATA INITIATIVE, 2012].

For instance, as it is introduced in [Big data, 2012], the USA Department of Defense (DOD) is "placing a big bet on big data" investing $250 million annually (with $60 million available for new research projects) across the Military Departments in a series of programs that will:

- Harness and utilize massive data in new ways and bring together sensing, perception and decision support to make truly autonomous systems that can maneuver and make decisions on their own;
- Improve situational awareness to help war fighters and analysts and provide increased support to operations. The Department is seeking a 100-fold increase in the ability of analysts to extract information from texts in any language, and a similar increase in the number of objects, activities, and events that an analyst can observe.

The ***XDATA*** program seeks to develop computational techniques and software tools for analyzing large volumes of data, both semi-structured (e.g., tabular, relational, categorical, meta-data) and unstructured (e.g., text documents, message traffic). Central challenges to be addressed include:

- Developing scalable algorithms for processing imperfect data in distributed data stores;
- Creating effective human-computer interaction tools for facilitating rapidly customizable visual reasoning for diverse missions.

The program envisions open source software toolkits for flexible software development that enable processing of large volumes of data for use in targeted defense applications.

The *Cyber-infrastructure for a Billion Electronic Records (CI-BER)* of the USA National Archives & Records Administration (NARA) is a joint agency sponsored testbed notable for its application of a multi-agency sponsored cyber infrastructure and the National Archives' diverse 87+ million file collection of digital records and information now active at the Renaissance Computing Institute. This testbed will evaluate technologies and approaches to support sustainable access to ultra-large data collections.

At the end, in the USA National Science Foundation (NSF), the *Information Integration and Informatics* addresses the challenges and scalability problems involved in moving from traditional scientific research data to very large, heterogeneous data, such as the integration of new data types models and representations, as well as issues related to data path, information life cycle management, and new platforms [Big data, 2012].

Worldwide Big Data technology and services are expected to grow. The challenge is to strengthen Europe’s position as provider of innovative multilingual products and services based on digital content and data, addressing well identified industry and consumer market needs. Research and Innovation activities in this challenge will provide professionals and citizens with new tools to model, analyze, and visualize vast amounts of data from which to extract more value, to make an intelligent use of data coming from different sources and to create, access, exploit, and re-use all forms of digital content in any language and with any device [HORIZON 2020, 2013].

In accordance with the actuality of these problems, this work is aimed to solve the problem of searching in big data structures by proposing a special kind of hashing, so-called “multi-layer hashing”, i.e. by implementing recursively the same specialized hash function to build and resolve the collisions in hash tables. In other words, the main idea consists in using the specialized hashing function in depth till it is needed.

This approach is called “Natural Language Addressing” (NLA) [Ivanova et al, 2012a; Ivanova et al, 2013a; Ivanova et al, 2013d]. In this work we will concern:

- structured data (dictionaries, thesauruses, ontologies);
- semi-structured data (big RDF triple or quadruple datasets),

and will provide corresponded experiments and experimental practical implementation.

The idea of Natural Language Addressing (NLA)

➢ *Variety and orderliness*

The world around us can be described in one word as "Variety". It is difficult to agree that the world has not so needed orderliness, created over millennia, developed and maintained constantly as oasises of ***order*** in the core of the chaos... It is strange for our perception of the world as a four-dimensional existence. It is strange, because our mind builds a completely different picture of ordered spatiality and extensity.

The concept “order” has many meanings but here it is used in the sense of a condition of logical or comprehensible arrangement among the separate elements of a group [AHD, 2009]; a state in which all components or elements are arranged logically, comprehensibly, or naturally; sequence (*alphabetical order)* [Collins, 2003]; arrangement of thoughts, ideas, temporal events [WordNet, 2012].

One very important aspect of the order is that every entity of the ordered set has its own location in it. The names of these locations are called addresses.

The common sense meaning of the concept “***address***” is such as a description of the location of a person or organization, as written or printed on mail as directions for delivery [AHD, 2009]; the conventional form by which the location of a building is described [Collins, 2003]; a sign in front of a house or business carrying the conventional form by which its location is described; [WordNet, 2012].

We will use the concept **“address”** in the sense accepted in the Computer Science: the code that identifies where a piece of information is stored [WordNet, 2012]; a name or number used in information storage or retrieval that is assigned to a specific memory location; the memory location identified by this name or number; a name or a sequence of characters that designates an e-mail account or a specific site on the Internet or other network [AHD, 2009].

It is important to take in account that the memory address may be of two kinds [Stably, 1970]:

- Physical location in any device (hard disk, main memory, flash memory);
- Logical (relative) location in a file given as an offset from the beginning of the file, i.e. the position of a byte in the file. In other words, it is the sequential number of the pointed byte in the file, starting from zero.

In this research we will use concept “*memory address*” only in the second sense, i.e. as ***offset in a file stored somewhere in the computer accessible local or global network environment.***

➢ *Name – Address - Route*

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].

Establishing and supporting the correspondence between logical and physical addresses is duty of the operating systems or, in general, of all service functions of the local or global networks. This correspondence is transparent for the end user programs which request the logical address and operating environment is responsible to locate and access concrete physical location.

Special kind of files are so called "*main memory mapped files*" which are accessible as files but during their processing are stored in the main computer memory and only updated their blocks are written on the external memory devices. Such kind of processing of files is useful for speeding the work of programs.

The concept "(logical) address" is closely connected with the term "***information model***" [Ivanova, 2013].

➢ *Information models*

We continuously build information models of the world and of ourselves in this world. The need of coordinating our actions with others humans or intelligent devices requires constant information exchange (interaction), the basis of which are the information models.

In the Computer Science, the term "***information model***" is popular.

"An information model is a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. The advantage of using an information model is that it can provide sharable, stable, and organized structure of information requirements for the domain context. An information modeling language is a formal syntax that allows users to capture data semantics and constraints" [Lee, 1999].

In other words, the modeled objects are information structures and the relations between them. The "***computer information models***" concern logical organization of storing the information and operations with it.

It is wrong to believe that the information models are a phenomenon only of humans. But only for humans there exist letters and accordingly ***textual information models*** (see for instance [Čech, 2012]).

➢ *Addressing in the textual information models*

The simplest textual information model is a linear structure of text elements – letters, words, sentences or more complicated structures like tags in XML.

Some models have continuous internal structure which may be divided on substructures, etc. For instance, the Brookshear's "Overview of the Computer Science" is such model. It is represented by a book with chapters [Brookshear, 2012]. It is a complex information model because contains non-textual elements: graphics and pictures.

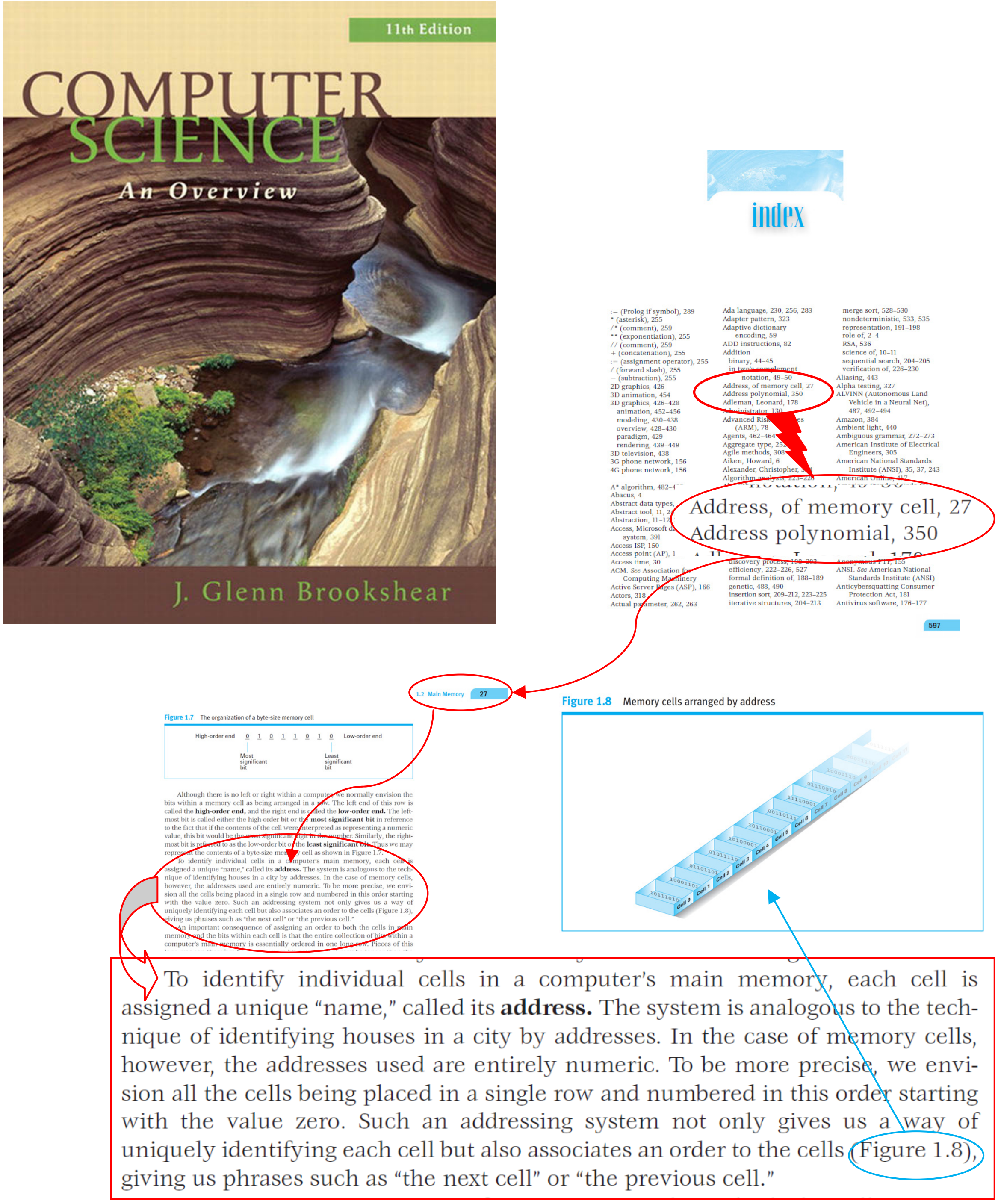
The nonlinear information models may be represented by graphs of interconnected textual elements. An example of such model is graphical representation of any ontology. Other examples are relational structures usually represented by sets of tables.

When the definitions are placed randomly in a book, for the sake of convenience at the end of book is located an index with main concepts and numbers of the pages where the concepts are defined. One needs to follow simple algorithm to find a definition. This is illustrated at Figure 1 for the concept "address, of memory cell".

The important elements of the textual models may be defined by corresponded definitions located in different places of the text. If the concepts together with theirs definitions are ordered alphabetically, like in a dictionary (Figure 2), going through the text one may found the needed concept and its definition.

In other words, irrespective of the type of the textual information model, every text element has its own location in the text and, respectively, its own relative address in the text document (page, paragraph, number of word, etc.) or file (relative offset from the first position in the file). Some of the elements may be so important to be pointed by their relative positions in an *index*.

Index is a sequential arrangement of material, especially in alphabetical or numerical order, which serves to guide, point out or otherwise facilitate reference, especially: a more or less detailed alphabetized list of names, places, subjects, etc, treated in the text of a printed work. It usually appears at the end of the book and identifies page numbers on which information about each subject appears [AHD, 2009; Collins, 2003].



:– (Prolog if symbol), 289
* (asterisk), 255
/* (comment), 259
** (exponentiation), 255
// (comment), 259
+ (concatenation), 255
:= (assignment operator), 255
/ (forward slash), 255
– (subtraction), 255
2D graphics, 426
3D animation, 454
3D graphics, 426–428
animation, 452–456
modeling, 430–438
overview, 428–430
paradigm, 429
rendering, 439–449
3D television, 438
3G phone network, 156
4G phone network, 156

A* algorithm, 482–4
Abacus, 4
Abstract data types,
Abstract tool, 11, 2
Abstraction, 11–12
Access, Microsoft d
system, 391
Access ISP, 150
Access point (AP), 1
Access time, 30
ACM. *See* Association for
Computing Machinery
Active Server Pages (ASP), 166
Actors, 318
Actual parameter, 262, 263

Ada language, 230, 256, 283
Adapter pattern, 323
Adaptive dictionary
encoding, 59
ADD instructions, 82
Addition
binary, 44–45
in two's complement
notation, 49–50
Address, of memory cell, 27
Address polynomial, 350
Adleman, Leonard, 178
Administrator, 130
Advanced Risk es
(ARM), 78
Agents, 462–464
Aggregate type, 252
Agile methods, 308
Aiken, Howard, 6
Alexander, Christopher,
Algorithm analysis, 223–226

Address, of memory cell, 27
Address polynomial, 350

discovery process, 198–203
efficiency, 222–226, 527
formal definition of, 188–189
genetic, 488, 490
insertion sort, 209–212, 223–225
iterative structures, 204–213

merge sort, 528–530
nondeterministic, 533, 535
representation, 191–198
role of, 2–4
RSA, 536
science of, 10–11
sequential search, 204–205
verification of, 226–230
Aliasing, 443
Alpha testing, 327
ALVINN (Autonomous Land
Vehicle in a Neural Net),
487, 492–494
Amazon, 384
Ambient light, 440
Ambiguous grammar, 272–273
American Institute of Electrical
Engineers, 305
American National Standards
Institute (ANSI), 35, 37, 243
American Online, 417
Anonymous FTP, 155
ANSI. *See* American National
Standards Institute (ANSI)
Anticybersquatting Consumer
Protection Act, 181
Antivirus software, 176–177

1.2 Main Memory 27

Figure 1.7 The organization of a byte-size memory cell

Although there is no left or right within a computer, we normally envision the bits within a memory cell as being arranged in a row. The left end of this row is called the **high-order end,** and the right end is called the **low-order end.** The leftmost bit is called either the high-order bit or the **most significant bit** in reference to the fact that if the contents of the cell were interpreted as representing a numeric value, this bit would be the most significant digit in the number. Similarly, the rightmost bit is referred to as the low-order bit or the **least significant bit.** Thus we may represent the contents of a byte-size memory cell as shown in Figure 1.7.

To identify individual cells in a computer's main memory, each cell is assigned a unique "name," called its **address.** The system is analogous to the technique of identifying houses in a city by addresses. In the case of memory cells, however, the addresses used are entirely numeric. To be more precise, we envision all the cells being placed in a single row and numbered in this order starting with the value zero. Such an addressing system not only gives us a way of uniquely identifying each cell but also associates an order to the cells (Figure 1.8), giving us phrases such as "the next cell" or "the previous cell."

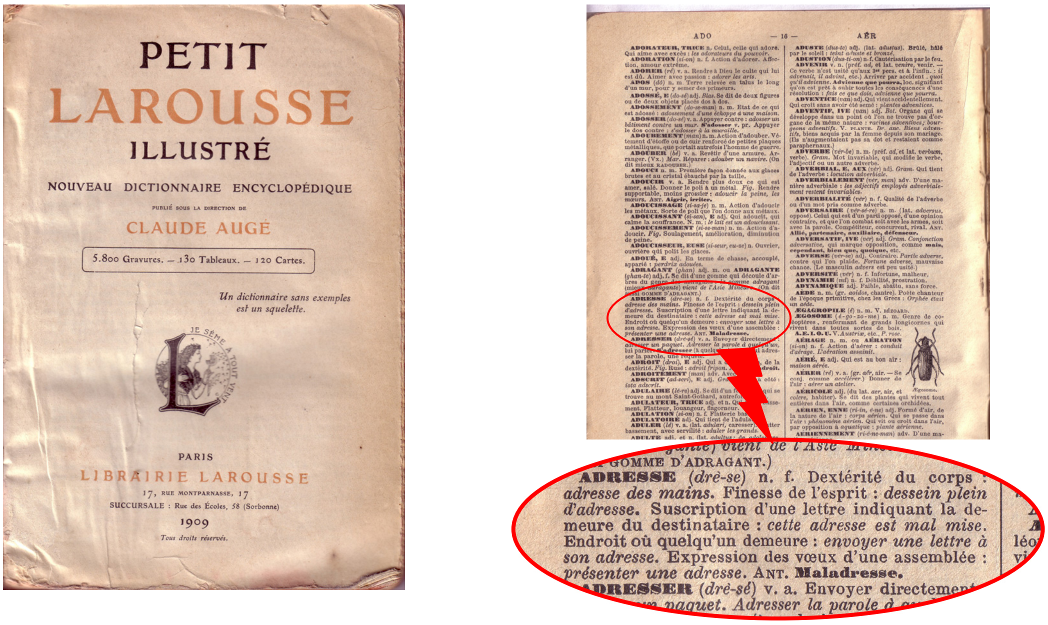
An important consequence of assigning an order to both the cells in main memory and the bits within each cell is that the entire collection of bits within a computer's main memory is essentially ordered in one long row. Pieces of this

Figure 1.8 Memory cells arranged by address

To identify individual cells in a computer's main memory, each cell is assigned a unique "name," called its **address.** The system is analogous to the technique of identifying houses in a city by addresses. In the case of memory cells, however, the addresses used are entirely numeric. To be more precise, we envision all the cells being placed in a single row and numbered in this order starting with the value zero. Such an addressing system not only gives us a way of uniquely identifying each cell but also associates an order to the cells (Figure 1.8), giving us phrases such as "the next cell" or "the previous cell."

Figure 1. Addressing by indexing [Brookshear, 2012]

Sets of concepts and their definitions, organized in dictionaries, are ordered alphabetically and this way location of every concept may be found easily.



PETIT
LARROUSSE
ILLUSTRÉ
NOUVEAU DICTIONNAIRE ENCYCLOPÉDIQUE
PUBLIÉ SOUS LA DIRECTION DE
CLAUDE AUGÉ
5.800 Gravures. — 130 Tableaux. — 120 Cartes.

Un dictionnaire sans exemples est un squelette.

PARIS
LIBRAIRIE LAROUSSE
17, RUE MONTPARNASSE, 17
SUCCURSALE : Rue des Écoles, 58 (Sorbonne)
1909
Tous droits réservés.

GOMME D'ADRAGANT.)

ADRESSE (*drè-se*) n. f. Dextérité du corps : *adresse des mains.* Finesse de l'esprit : *dessein plein d'adresse.* Suscription d'une lettre indiquant la demeure du destinataire : *cette adresse est mal mise.* Endroit où quelqu'un demeure : *envoyer une lettre à son adresse.* Expression des vœux d'une assemblée : *présenter une adresse.* ANT. **Maladresse.**

ADRESSER (*drè-sé*) v. a. Envoyer directement

Figure 2. Addressing by natural language order [Auge, 1909]

➢ *Computer indexes*

The text information models may be stored as files in the (internal or external) computer memory. Locating the concepts and definitions may be done by:

- Direct scanning the files;
- Indexing and based on it search of the pointer to the address (number of the first byte) of the text element (record in the file).

Scanning the files is convenient only for small volumes of concepts and definitions. Some rationalization is possible using some algorithms like binary search.

Indexing is creating tables (indexes) that point to the location of folders, files and records. Depending on the purpose, indexing identifies the location of resources based on file names, key data fields in a database record, text within a file or unique attributes in a graphics or video file [PC mag, 2013].

In database design, an index is a list or a reference table of keys (or keywords), each of which identifies a unique record or document and is used to locate a particular element within a data array or table. Indices make it faster to find specific records and to sort records by the index field - that is, the field used to identify each record [Webopedia, 2013; AHD, 2009; Collins, 2003].

For large volumes of concepts, the indexes became too large and additional, secondary indexing is needed. Such multi-level index structures are well-known B-trees of Rudolf Bayer [Bayer, 1971] as well as B$^+$ trees [Knuth, 1997] (Figure 3).

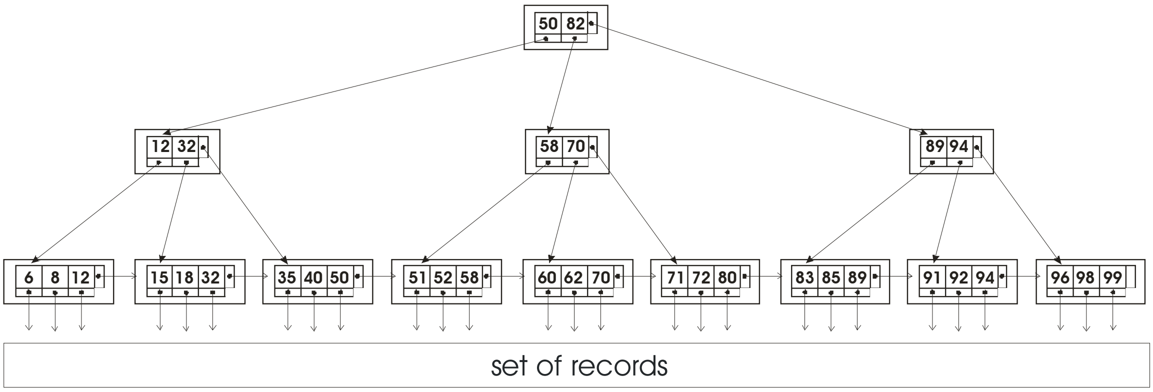


Figure 3. B-tree

Let repeat, *the main idea of indexing is to facilitate the search by search in the (multi-level) index* and after that to ensure the *direct access to the address given by the pointer*. The address is relative offset of the first byte of the record from the beginning (first byte) of the file. The value of the offset is just what the pointer consists.

In other words, the goal of data indexing is to ease the search of and access to data at any given time. This is done by creating a data structure called index and providing faster access to the data. Accessing data is determined by the physical storage device being used. Indexing could potentially provide large increases in performance for large-scale analysis of unstructured data. Additionally the implementation of the chosen index must be suitable in terms of index construction time and storage utilization [Faye et al, 2012].

Indexing needs resources: memory for storing additional information and time for processing, which may be quite a long, especially for updating of the indexes when new elements are added or some old ones are removed.

➢ *Naming the addresses*

Basic element of an index is couple: (name, address).

For instance such couples on Figure 1 are:

(“Address, of memory cell”, 27) (“Address polynomial”, 350)

In different sources the “name” is called “key”, “concept”, etc. The address usually is given by any “number”, “pointer”, “offset”, “location”, etc.

There are two interpretations of the couple (concept, address):

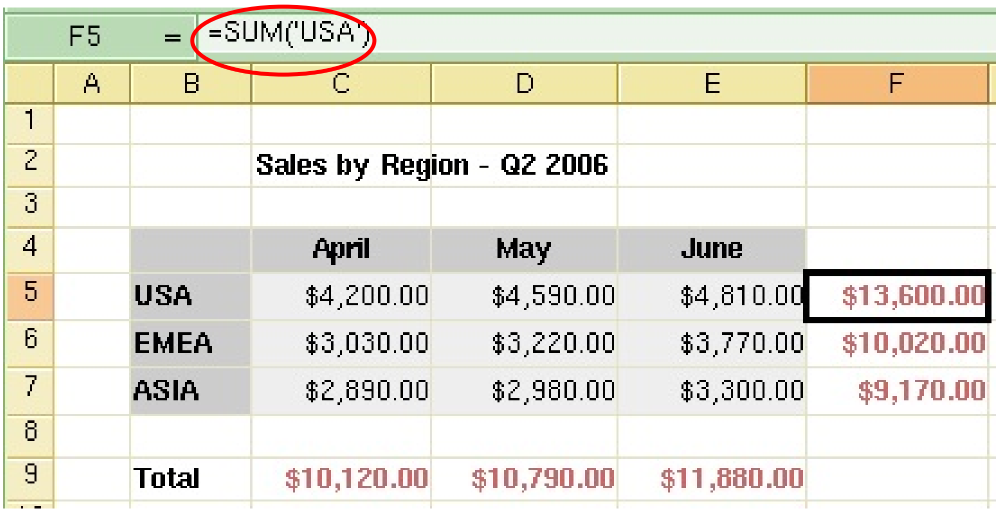
1) The address is *a connection of the concept with its definition in the text*, i.e. practically we have triple: ***(name, address, definition)***.

2) The concept is a name of a computer main memory address and may be used for user friendly style of programming and the third part (value) may be variable, i.e. practically we have triple: ***(name, address, value)***.

In the very beginning, replacing the address by name was used in the programming languages for pointing the addresses by names of identificators, like in Algol 60 [Naur, 1963]. (In this case, the address is real memory location but not offset in a file in the memory. We will not discuss all kinds of addressing in the computer memory used in processor’s registers. In this text, the address is relative offset from the first byte of a file, sometimes given together with information for file location (path to the) file.)

Later, the same idea was used in the Web navigation systems. Web navigation is mostly based on Uniform Resource Locators (URLs). URLs can be hard to remember and change constantly. For instance, in the International Human-Friendly Web Navigation System, the RealNames' Internet Keywords offer an alternative Web addressing scheme using natural language, replacing unfriendly URLs like http://www.fordvehicles.com/vehiclehome.asp?vid=12 with common names such as "Ford Mustang". Building a fully international system that provides a human-friendly naming infrastructure for the whole Web is a challenging task. By leveraging Unicode to represent names it is possible to build a global naming engine that, coupled with knowledge of local customs simplifies Web navigation through the use of natural language keywords [Arrouse, 1999].

Some of the electronic spreadsheets have possibility to point a group of cells and/or rows with any name and further to use this name in functions and other operations assuming all cells and/or rows named by this name [Zoho sheet, 2012] (Figure 4).



F5 = =SUM('USA')

	A	B	C	D	E	F
1						
2			Sales by Region - Q2 2006			
3						
4			April	May	June	
5		USA	$4,200.00	$4,590.00	$4,810.00	$13,600.00
6		EMEA	$3,030.00	$3,220.00	$3,770.00	$10,020.00
7		ASIA	$2,890.00	$2,980.00	$3,300.00	$9,170.00
8						
9		Total	$10,120.00	$10,790.00	$11,880.00	

Figure 4. Natural Language Addressing in a spreadsheet

For instance, Zoho Sheet can recognize and correlate names used in formulas with cells/cell ranges automatically. You have to just give the row/column header of a table as arguments to functions and Zoho Sheet will auto-recognize the cell range associated with the name. It is very convenient to quickly type in the formulas with these names instead of worrying about keying in the proper cell range.

Consider the sheet on Figure 4, available at http://zohosheet.com/public.do?fid=25835.

Look at the formulas in the cells F5:F7 and C9:E9. The formula =SUM (USA) will automatically add the cell values in the row with the header 'USA'. Earlier you had to use =SUM (C5:E5). Now the row header can directly be used. You do not even need to name/label the cell ranges. You can even copy and paste these formulas to adjacent rows or columns and they will automatically be adjusted relatively. In this case, copying the F5 cell and pasting it to F6, will result in the formula =SUM (EMEA) in F6. (Here the concept "address" is used as cell co-ordinates in the table

(C5, C9, E5, F6, etc.) but not as offset in a file.)

The approach of replacing cell addresses with names in Zoho Sheet was called "***Natural Language Addressing***".

➢ *Using encoding of the name both as address and as route*

In this research we follow a proposition of Krassimir Markov to use the computer encoding of name (concept) letters as logical address of connected to it information. This way no indexes are needed and high speed direct access to the text elements is available. It is similar to the natural order addressing in a dictionary shown at Figure 2 where no explicit index is used but the concept by itself locates the definition. Our approach is similar to one in the Zoho Sheet, too.

Because of this we will use the same term: "***Natural Language Addressing***".

Shoch's definition [Shoch, 1978a] 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 Natural Language Addressing (NLA) is to use encoding of the name *both as relative address and as route in a multi-dimensional information space* and this way to speed the access to stored information.

For instance, let have the next definition:

*"**London:** The capital city of England and the United Kingdom, and the largest city, urban zone and metropolitan area in the United Kingdom, and the European Union by most measures".*

In the computer memory, for example, it may be stored in a file at relative address "00084920" and the index couple is: ("London", "00084920")

At the memory address "00084920" the main text, "*The capital ... measures*." will be stored.

To read/write the main text, firstly we need to find name "*London*" in the index and after that to access memory address "00084920" to read/write the definition.

If we assume that name "London" in the computer memory is encoded by six numbers (letter codes), for instance by using ASCII encoding system London is encoded as (76, 111, 110, 100, 111, 110), than we may use these codes for direct address to memory, i.e.

("London", "76, 111, 110, 100, 111, 110")

Above we have written two times the same name and this is truth. Because of this we may omit this couple and index, and read/write directly to the address "76, 111, 110, 100, 111, 110".

For human this address will be shown as "London", but for the computer it will be "76, 111, 110, 100, 111, 110".

Now, what we need is a tool for storing and accessing information using Natural Language Addressing. At first glance, such tool may be the hash tables.

➢ *Hashing and natural language addressing*

The array "76, 111, 110, 100, 111, 110" may be assumed as an offset, i.e. as number "076111110100111110". This causes two main problems:

- We need a hypothetic file with unlimited length;

- The offset points to only one byte but the definition is 170 bytes long and will occupy the next addresses.

A possible solution is using hash tables.

Hash tables are used in a wide variety of applications. In networking systems, they are used for a number of purposes, including: load balancing, intrusion detection, TCP/IP state management, and IP address lookups. Hash tables are often attractive since sparse tables result in constant-time, O(1), query, insert and delete operations. However, as the table occupancy, or load, increases, collisions will occur which in turn places greater pressure on the collision resolution policy and often dramatically increases the cost of the primitive operations. In fact, as the load increases, the average query time increases steadily and very large worst case query times become more likely [Kumar & Crowley, 2005].

This means that we could not use encoding of names as keys for hash tables or direct offsets. We need a special organization of file internal structure and function that will transform the name in a unique location in the file where the definition will be stored without collisions with other texts.

This problem is already solved at the level of file system – every file has its own name and file system converts it (using file allocation table – FAT) in an address of the file's first block on the hard disk. This is convenient for information which is relatively long – whole documents, images, music files, etc. because every file occupies at least one cluster (2KB, 4KB or more hard disk space). We have to extend this idea to be used into the file. For this purpose we have to establish special kind of file internal organization with additional specialized indexing.

The idea presented in this work differs from the hashing by two characteristics:

- The function which juxtapose the letters to integer numbers is one-one mapping and this way no collisions exist;
- This mapping (hash function) may be used recursively for every symbol of a string to build hierarchical multi-layer set of hash tables and this way to speed the access to information.

For instance, the array "76, 111, 110, 100, 111, 110" may be assumed as a route to (co-ordinates of) a point in a multi-dimensional (in this case: six-dimensional) information space and the definition may be stored in this point.

In other words, our function may be used recursively for every symbol and this way we will create hierarchical multi-layer set of tables. For the case of word "London" we will have six layers.

The natural language does not contain words only of six letters long. The length of the words is variable and in addition there exist names as phrases like "Address polynomial" above. The set of all natural words and phrases defines a multi-dimensional logical address space with variable dimensions and unlimited size.

What we need are:

- A special algorithm which converts such multi-dimensional addresses in concrete routes to linear (relative) locations in the files (on the hard disk, for example);
- A program tool which will realize this algorithm.

A solution of this problem is presented in this monograph.

Brief overview of the content

Chapter 1. *Firstly in this chapter, we will remember the needed basic mathematical concepts. Special attention will be paid to the Names Sets – mathematical structure which is used further for building models needed for our research. We will use strong hierarchies of named sets to create a specialized mathematical model for new kind of organization of information bases called "Multi-Domain Information Model" (MDIM). The "information spaces" defined in the model are kind of strong hierarchies of enumerations (named sets).*

At the end, we will remember the main features of hashing and types of hash tables as well as the idea of "Dynamic perfect hashing" and "Trie", especially – the "Burst trie". Hash tables and tries give very good starting point. The main problem is that they are designed as structures in the main memory which has limited size, especially in small desktop and laptop computers.

Chapter 2 *presents state of the art in the storing models.*

This chapter is aimed to introduce the main data structures and storing technologies which we will use to compare our results. Mainly they are graph data models as well as Resource Description Framework (RDF) storage and retrieval technologies.

Firstly we shortly define concepts of storage model and data model.

Mapping of the data models to storage models is based on program tools called "access methods". Their main characteristics will be outlined.

During the eighties of the last century, the total growing of the research and developments in the computers' field, especially in image processing, data mining and mobile support cause impetuous progress of establishing convenient "spatial information structures" and "spatial-temporal information structures" and corresponding access methods. Important cases of spatial representation of information are Graph models. Because of this, Graph models and databases will be discussed more deeply and examples of different graph database models will be presented. The need to manage information with graph-like nature especially in RDF-databases has reestablished the relevance of this area.

In accordance with this, the analyses of RDF databases as well as of the storage and retrieval technologies for RDF structures will be in the center of our attention. Storing models for several popular ontologies and summary of main types of storing models for ontologies and, in particular, RDF data, will be discussed.

Our attention will be paid to addressing and naming (labeling) in graphs with regards to introducing the Natural Language Addressing (NL-addressing) in graphs. A sample graph will be analyzed to find its proper 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 example, the advantages and disadvantages of the multi-layer representation of graphs will be outlined.

For practical implementation of NLA we need a proper model for database organization and corresponded specialized tools. To achieve such possibilities, we will use "Multi-Domain Information Model" (MDIM) and corresponded to it software tools to realize dynamic perfect hashing and burst tries as external memory structures.

***Chapter 3** introduces an Access method based on NL-addressing.*

This chapter is aimed to introduce a new access method based on the idea of NL-addressing.

For practical implementation of NLA we need a proper model for database organization and corresponded specialized tools. Hash tables and tries give very good starting point. The main problem is that they are designed as structures in the main memory which has limited size, especially in small desktop and laptop computers. Because of this we need analogous disk oriented database organization.

To achieve such possibilities, we decided to use "Multi-Domain Information Model" (MDIM) and corresponded to it software tools. MDIM and its realizations are not ready to support NL-addressing. We will upgrade them for ensuring the features of NL-addressing via new access method called NL-ArM.

The program realization of NL-ArM, based on specialized hash functions and two main functions for supporting the NL-addressing, access will be outlined. In addition, several operations aimed to serve the work with thesauruses and ontologies as well as work with graphs, will be presented.

***Chapter 4** is aimed to outline two basic experiments.*

In this chapter we will present two types "clear" experiments: with a text file and a relational database. The reason is that they are wide used for storing of semi-structured data.

***Chapter 5** contains description of experiments for NL-storing of small datasets.*

In this chapter we will present several experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of small size datasets which contain up to one hundred thousands of instances.

The goal of the experiments with different small datasets, like dictionary, thesaurus or ontology, is to discover regularities in the NL-addressing realization. More concretely, two regularities of time for storing by using NL-addressing will be examined:

— *It depends on number of elements in the instances;*

— *It not depends on number of instances in datasets.*

This chapter starts with introduction of the idea of knowledge representation. Further in the chapter three experiments with small size datasets are outlined: for NL-storing of dictionaries, thesauruses, and ontologies. Presentation of every experiment starts with introductory part aimed to give working definition and to outline state of the art in storing concrete structures.

We start with analyzing the easiest one: NL-storing dictionaries. After that, NL-storing of thesauruses will be analyzed. An experiment with WordNet thesaurus and program WordArM based on NL-addressing will be discussed.

At the end, a special attention will be given to NL-storing ontologies. This part of the chapter begins with introducing the basic ontological structures as well as the corresponded operations and tools for operating with ontologies. Further, NL-storing models for ontologies will be discussed and experiments with OntoArM program for storing ontologies based on NL-addressing will be outlined.

***Chapter 6** grounds on analyzing experiments for NL-storing middle-size and large RDF-datasets.*

In this chapter we will present results from series of experiments which are needed to estimate the storing time of NL-addressing for middle-size and large RDF-datasets.

The experiments for NL-storing of middle-size and large RDF-datasets are aimed to estimate possible further development of NL-ArM. We assume that its "software growth" will be done in the same grade as one of the known systems like Virtuoso, Jena, and Sesame. We will analyze what will be the place of NL-ArM in this environment. Our hypothesis is that NL-addressing will have good performance.

Chapter will start with describing the experimental storing models and algorithm used in this research. Further an estimation of experimental systems will be provided to make different configurations comparable. Special proportionality constants for hardware and software will be proposed. Using proportionality constants, experiments with middle-size and large datasets became comparable.

Experiments will be provided with both real and artificial datasets. Experimental results will be systematized in corresponded tables. For easy reading visualization by histograms will be given.

***Chapter** 7 contains analysis of experiments.*

In this chapter we will analyze experiments presented in previous chapters 4, 5, and 6, which contain respectively results from (1) basic experiments; (2) experiments with structured datasets; (3) experiments with semi-structured datasets. Special attention will be paid to analyzing of storing times of NL-ArM access method and its possibilities for multi-processing.

*In **Chapter 8** practical aspects will be discussed.*

Some practical aspects of implementation and using of NL-addressing will be discussed in this chapter.

NL-addressing is approach for building a kind of so called "post-relational databases". In accordance with this the transition to non-relational data models will be outlined.

The implementation has to be done following corresponded methodologies for building and using of ontologies. Such known methodology will be discussed in the chapter. It is called "METHONTOLOGY" and guides in how to carry out the whole ontology development through the specification, the conceptualization, the formalization, the implementation and the maintenance of the ontology.

Special case is creating of ontologies of text documents which are based on domain ontologies. It consists of Document annotation and Ontology population which we will illustrate following the OntoPop platform [Amardeilh, 2006].

The software realized in this research was practically tested as a part of an instrumental system for automated construction of ontologies "ICON" ("Instrumental Complex for Ontology designatioN") which is under development in the Institute of Cybernetics "V.M.Glushkov" of NAS of Ukraine.

In this chapter we briefly will present ICON and its structure. Attention will be paid to the storing of internal information resources of ICON realized on the base of NL-addressing and experimental programs WordArM and OntoArM.

***Conclusion** contains a short presentation of the next steps. Special attention is done on the area of so called "Big Data" and possible implementation of NLA for processing of large semi-structured data sets. A brief outline of the main achievements of this work is given.*

***Appendix A** outlines the program realizations of tools for storing information using NL-addressing: WordArM, OntoArM, and RDFArM for storing thesauruses, small ontologies, and large RDF triple datasets. Some illustrative tables and figures from experiments as well as other supporting information are given.*

***Appendix B** contains short information about tools analyzed in the monograph. Main attention will be paid to Protégé 4.2 and SPARQL. Storage characteristics of analyzed RDF triple stores will be presented shortly in two groups: (1) DBMS based approaches, (2) Multiple indexing frameworks.*

The monograph was written by:

- Krassimir Markov: Introduction, Ch. 2, 4. 7, Coclusion, and Appendix B2-B3 (125 pp.);

- Krassimira Ivanova: Ch. 1, 3, 5, 6, 8.1-8.3, and Appendix A1-A5 (153 pp.);

- Vitalii Velychko: Chapter 8.4, Appendix A6 - A8, and Appendix B1 (27 pp.)

- Koen Vanhoof and Juan Castellanos: consulting and editing.

1 Theoretical surroundings

Abstract

Firstly in this chapter, we will remember the needed basic mathematical concepts. Special attention will be paid to the Names Sets – mathematical structure which is used further for building models needed for our research. We will use strong hierarchies of named sets to create a specialized mathematical model for new kind of organization of information bases called "Multi-Domain Information Model" (MDIM). The "information spaces" defined in the model are kind of strong hierarchies of enumerations (named sets).

At the end, we will remember the main features of hashing and types of hash tables as well as the idea of "Dynamic perfect hashing" and "Trie", especially – the "Burst trie". Hash tables and tries give very good starting point. The main problem is that they are designed as structures in the main memory which has limited size, especially in small desktop and laptop computers. For practical implementation of NLA we need a proper model for database organization and corresponded specialized tools. To achieve such possibilities, we will use "Multi-Domain Information Model" (MDIM) and corresponded to it software tools to realize dynamic perfect hashing and burst tries as external memory structures.

1.1 Basic mathematical concepts

Let remember the basic mathematical concepts needed for this research [Bourbaki, 1960, Burgin, 2010].

$\varnothing$ is the *empty set*.

If X is a set, then $r \in X$ means that r belongs to X or r is a member of X.

If X and Y are sets, then $Y \subseteq X$ means that Y is a subset of X, i.e., Y is a set such that all elements of Y belong to X.

The *union* $Y \cup X$ of two sets Y and X is the set that consists of all elements from Y and from X.

The *intersection* $Y \cap X$ of two sets Y and X is the set that consists of all elements that belong both to Y and to X.

The *union* $\bigcup_{i \in I} X_i$ of sets X_i is the set that consists of all elements from all sets X_i, $i \in I$.

The *intersection* $\bigcap_{i\in I} X_i$ of sets X_i is the set that consists of all elements that belong to each set X_i, $i\in I$.

The *difference* $Y\backslash X$ of two sets Y and X is the set that consists of all elements that belong to Y but does not belong to X.

If X is a set, then 2^X is the *power set* of X, which consists of all subsets of X. The power set of X is also denoted by $\boldsymbol{P}(X)$.

If X and Y are sets, then $X \times Y = \{(x, y); x \in X, y \in Y\}$ is the direct or Cartesian product of X and Y, in other words, $X \times Y$ is the set of all pairs (x, y), in which x belongs to X and y belongs to Y.

Elements of the set X^n have the form $(x_1, x_2, \ldots, x_\mathrm{n})$ with all $x_i \in X$ and are called n-tuples, or simply, tuples.

A fundamental structure of mathematics is *function*. However, functions are special kinds of binary relations between two sets.

A *binary relation* T between sets X and Y is a subset of the direct product $X \times Y$. The set X is called the *domain* of T ($X = \mathrm{Dom}(T)$) and Y is called the *codomain* of T ($Y = \mathrm{CD}(T)$). The *range* of the relation T is $\mathrm{Rg}(T) = \{y; \exists\, x \in X\, ((x, y) \in T)\}$. The *domain of definition* of the relation T is $\mathrm{DDom}(T) = \{x; \exists\, y \in Y\, ((x, y) \in T)\}$. If $(x, y) \in T$, then one says that the elements x and y are in relation T, and one also writes $T(x, y)$.

Binary relations are also called multi valued functions (mappings or maps).

Y^X is the set of all mappings from X into Y.

$$X^n = \underbrace{X \times X \times \ldots X \times X}_{\mathrm{n}}.$$

A *preorder* (also called *quasiorder*) on a set X is a binary relation Q on X that satisfies the following axioms:

1. Q is reflexive, i.e. $x\mathrm{Q}x$ for all x from X.

2. Q is transitive, i.e., $x\mathrm{Q}y$ and $y\mathrm{Q}z$ imply $x\mathrm{Q}z$ for all $x, y, z \in X$.

A *partial order* is a preorder that satisfies the following additional axiom:

3. Q is antisymmetric, i.e., $x\mathrm{Q}y$ and $y\mathrm{Q}x$ imply $x = y$ for all $x, y \in X$.

A *strict partial order* is a preorder that is not reflexive, is transitive and satisfies the following additional axiom:

4. Q is asymmetric, i.e., only one relation $x\mathrm{Q}y$ or $y\mathrm{Q}x$ is true for all $x, y \in X$.

Equivalence on a set X is a binary relation Q on X that is reflexive, transitive and satisfies the following additional axiom:

5. Q is symmetric, i.e., $x\mathrm{Q}y$ implies $y\mathrm{Q}x$ for all x and y from X.

A *function* (also called a *mapping* or *map* or *total function* or *total mapping*) f from X to Y is a binary relation between sets X and Y in which:

- There are no elements from X which are corresponded to more than one element from Y;
- To any element from X, some element from Y is corresponded.

Often total functions are also called everywhere defined functions. Traditionally, the element $f(a)$ is called the image of the element a and denotes the value of f on the element a from X. At the same time, the function f is also denoted by f: $X \rightarrow$ Y or by $f(x)$. In the latter formula, x is a variable and not a concrete element from X.

A *partial function* (or *partial mapping*) f from X to Y is a binary relation between sets X and Y in which there are no elements from X which are corresponded to more than one element from Y.

Thus, any function is also a partial function. Sometimes, when the domain of a partial function is not specified, we call it simply a function because any partial function is a total function on its domain.

A *multi valued function* (or *mapping*) f from X to Y is any binary relation between sets X and Y.

$f(x) \equiv a$ means that the function $f(x)$ is equal to a at all points where $f(x)$ is defined.

Two important concepts of mathematics are the domain and range of a function. However, there is some ambiguity for the first of them. Namely, there are two distinct meanings in current mathematical usage for this concept. In the majority of mathematical areas, including the calculus and analysis, the term "domain of f" is used for the set of all values x such that $f(x)$ is defined. However, some mathematicians (in particular, category theorists), consider the domain of a function f: $X \rightarrow Y$ to be X, irrespective of whether $f(x)$ is defined for all x in X. To eliminate this ambiguity, we suggest the following terminology consistent with the current practice in mathematics.

If f is a function from X into Y, then the set X is called the *domain* of f (it is denoted by Dom$f$) and Y is called the *codomain* of T (it is denoted by Codom$f$). The *range* Rgf of the function f is the set of all elements from Y assigned by f to, at least, one element from X, or formally, Rg$f = \{y; \exists\, x \in X\,(f(x) = y)\}$. The *domain of definition* DDom$f$ of the function f is the set of all elements from X that related by f to, at least, one element from Y is or formally, DDom$f = \{x; \exists\, y \in Y\,(f(x) = y)\}$. Thus, for a partial function $f(x)$, its domain of definition DDom$f$ is the set of all elements for which $f(x)$ is defined.

Taking two mappings (functions) f: $X \rightarrow$ Y and g: $Y \rightarrow Z$, it is possible to build a new mapping (function) gf: $X \rightarrow Z$ that is called *composition* or *superposition* of mappings (functions) f and g and defined by the rule $gf(x) = g(f(x))$ for all x from X.

An n-ary relation R in a set X is a subset of the n^{th} power of X, i.e., $R \subseteq X^n$. If $(a_1, a_2, \ldots, a_n) \in R$, then one says that the elements $a_1, a_2, \ldots, a_n$ from X are in relation R.

➢ *Named sets*

Named set **X** is a triple $\mathbf{X} = (X, \mu, I)$ where:

- X is the *support* of **X** and is denoted by S(**X**);
- I is the *component of names* (also called *set of names* or *reflector*) of **X** and is denoted by N(**X**);
- μ: $X \rightarrow I$ is the *naming map* or *naming correspondence* (also called *reflection*) of the named set **X** and is denoted by n(**X**).

The most popular type of named sets is a named set $\mathbf{X} = (X, \mu, I)$ in which X and I are sets and μ consists of connections between their elements. When these connections are set theoretical, i.e., each connection is represented by a pair (x, a) where x is an element from X and a is its name from I, we have a *set theoretical named set*, which is binary relation.

A name $a \in I$ is called *empty* if $\mu^{-1}(a) = \varnothing$.

A named set **X** is called:

- *Normalized* if in **X** there are no empty names;
- *Conormalized* if in **X** there no elements without names;

Named sets as special cases include:

- Usual sets;
- Fuzzy sets;
- Multisets;
- Enumerations;
- Sequences (countable as well as uncountable);

etc.

A lot of examples of named sets we may find in linguistics studying semantical aspects that are connected with applying different elements of language (words, phrases, texts) to their meaning [Burgin & Gladun, 1989; Burgin, 2010].

A named set **Y** = (Y, η, J) is called *named subset* of named set **X** if $Y \subseteq X$, $J \subseteq I$, and $\eta = \mu\,|_{(Y,J)}$ $(\eta \subseteq \mu \cap (Y \times J))$. In this case **Y** and **X** are connected by the relation of the inclusion.

An ordered tuple of named sets $\Theta = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_k]$ where for all i=1, ..., k-1 the condition $N(\mathbf{X}_i) \cap S(\mathbf{X}_{i+1}) \neq \varnothing$ is fulfilled is called *chain of named sets*.

The number k is called a length of the chain Θ.

A tuple of named sets $\Xi_1 = [\mathbf{X}, \mathbf{Y}_1, \mathbf{Y}_2, ..., \mathbf{Y}_n]$ where for all i=1,...,n the condition $N(\mathbf{Y}_i) \cap S(\mathbf{X}) \neq \varnothing$ is fulfilled is called *one level hierarchy of named sets*.

If $N(\mathbf{Y}_i) \cap N(\mathbf{Y}_j) = \varnothing$ and $N(\mathbf{Y}_i) \subseteq S(\mathbf{X})$ for all i=1,...,n, j=1,...,n than Ξ is a *strong one level hierarchy of named sets*.

A tuple of named sets $\Xi_2 = [\mathbf{X}, \Xi_{1,1}, \Xi_{1,2}, ..., \Xi_{1,m}]$ where *sub-hierarchies* $\Xi_{1j} = [\mathbf{Y}_j, \mathbf{Z}_1, \mathbf{Z}_2, ..., \mathbf{Z}_k]$, j=1,...,m are one level hierarchy of named sets is called *second level hierarchy of named sets*.

If Ξ_{1j}, j=1,...,m, are strong one level hierarchies of named sets than Ξ_2 is a *strong second level hierarchy of named sets*.

A tuple of named sets $\Xi_n = [\mathbf{X}, \Xi_{n-1,1}, \Xi_{n-1,2}, ..., \Xi_{n-1,l}]$ where $\Xi_{n-1,i}$, i=1,...,l are n-1 level hierarchies of named sets than Ξ_n is a *n-th level hierarchy of named sets*..

If all sub-hierarchies of Ξ_n are strong hierarchies of named sets than Ξ_n is a *strong n-th level hierarchy of named sets*.

1.2 Hashing

A *set abstract data type* (set ADT) is an abstract data type that maintains a set ***S*** under the following three operations:

1. *Insert(x)*: Add the key *x* to the set.
2. *Delete(x)*: Remove the key *x* from the set.
3. *Search(x)*: Determine if *x* is contained in the set, and if so, return a pointer to *x*.

One of the most practical and widely used methods of implementing the set ADT is with *hash tables* [Morin, 2005].

The simplest implementation of such data structure is an ordinary array, where *k*-th element corresponds to key *k*. Thus, we can execute all operations in O(1). It is impossible to use this implementation, if the total number of keys is large [Kolosovskiy, 2009].

The main idea behind all hash table implementations is to store a set of $n = |\boldsymbol{S}|$ elements in an array (the hash table) A of length m. In doing this, we require a function that maps any element x to an array location. This function is called a *hash function h* and the value *h(x)* is called the *hash value of x*. That is, the element x gets stored at the array location $A[h(x)]$.

The occupancy of a hash table is the ratio $\alpha = n/m$ of stored elements to the length of A [Morin, 2005].

We have two cases: (1) m ≥ n and (2) m ≤ n:

- In the first case (m ≥ n) we may expect so called ***perfect hashing*** where every element may be stored in separate cell of the array. In other words, if we have a collection of n elements whose keys are unique integers in (1, m), where m ≥ n, then we can store the items in a direct address table, T[m], where T_i is either empty or contains one of the elements of our collection.
- In the second case (m ≤ n) we may expect so called "***collisions***" when two or more elements have to be stored in the same cell f the array.

If we work with two or more keys, which have the *same hash value*, these keys map to the same cell in the array. Such situations are called *collisions*. There are two basic ways to implement hash tables to resolve collisions:

- Chained hash table;
- Open-address hash table.

In ***chained hash table*** each cell of the array contains the linked list of elements, which have corresponding hash value. To add (delete, search) element in the set we add (delete, search) to corresponding linked list. Thus, time of execution depends on length of the linked lists.

In ***open-address hash table*** we store all elements in one array and resolve collisions by using other cells in this array. To perform insertion we examine some slots in the table, until we find an empty slot or understand that the key is contained in the table. To perform search we execute similar routine [Kolosovskiy, 2009].

The study of hash tables follows two very different lines: (1) integer universe assumption; (2) random probing assumption.

Integer universe assumption: All elements stored in the hash table come from the universe $U = \{0,...,u-1\}$. In this case, the goal is to design a hash function $h : U \rightarrow \{0,...,m-1\}$ so that for each $i \in \{0,...,m-1\}$, the number of elements $x \in \boldsymbol{S}$ such that $h(x) = i$ is as small as possible. Ideally, the hash function h would be such that each element of $\boldsymbol{S}$ is mapped to a unique value in $\{0,...,m-1\}$.

Historically, the **integer universe assumption** seems to have been justified by the fact that any data item in a computer is represented as a *sequence of bits that can be interpreted as a binary number*.

However, many complicated data items require a large (or variable) number of bits to represent and this make the size of the universe very large. In many applications *u* is much larger than the largest integer that can fit into a single word of computer memory. In this case, *the computations performed in number-theoretic hash functions become inefficient*. This motivates the second major line of research into hash tables, based on *Random probing assumption*.

Random probing assumption: Each element *x* that is inserted into a hash table is a black box that comes with an infinite random probe sequence $x_0, x_1, x_2, \ldots$ where each of the x_i is independently and uniformly distributed in $\{0, \ldots, m-1\}$.

Both the integer universe assumption and the random probing assumption have their place in practice.

When there is an easily computing mapping of data elements onto machine word sized integers then hash tables for integer universes are the method of choice.

When such a mapping is not so easy to compute (variable length strings are an example) it might be better to use the bits of the input items to build a good pseudorandom sequence and use this sequence as the probe sequence for some random probing data structure [Morin, 2005].

➢ *Perfect hash function*

We consider hash tables under the *integer universe assumption*, in which the key values *x* come from the universe $U = \{0, \ldots, u-1\}$. A hash function *h* is a function whose domain is *U* and whose level is the set $\{0, \ldots, m-1\}$, $m \leq u$.

A hash function h is said to be a ***perfect hash function*** for a set S ⊆ U if, ***for every x ∈ S, h(x) is unique.***

A *perfect hash function h* for *S* is ***minimal*** if $m = |S|$, i.e., *h* is a bisection between *S* and $\{0, \ldots, m-1\}$. Obviously a minimal perfect hash function for *S* is desirable since it allows us to store all the elements of *S* in a single array of length *n*. Unfortunately, perfect hash functions are rare, even for *m* much larger than *n* [Morin, 2005].

The set of elements, *S*, may be:

- *Static* (no updates);
- *Dynamic* where fast queries, insertions, and deletions must be made on a large set.

*"**Dynamic perfect hashing**"* is useful for the second type of situations. In this method, the entries that hash to the same slot of the table are organized as separate *second-level hash table*. If there are *k* entries in this set *S*, the second-level table is allocated with k^2 slots, and its hash function is selected at random from a universal hash function set so that it is *collision-free* (i.e. a perfect hash function). Therefore, the look-up cost is guaranteed to be O(1) in the worst-case [Dietzfelbinger et al, 1994].

Perfect hashing can be used in many applications in which we want to assign a unique identifier to each key without storing any information on the key. One of the most obvious applications of perfect hashing (or k-perfect hashing) is when we have a small fast memory in which we can store the perfect hash function while the keys and associated satellite data are stored in slower but larger memory. The size of a block or a transfer unit may be chosen so that *k* data items can be retrieved in one read access. In this case we can ensure that data associated with a key can be retrieved in a single probe to slower memory. This has been used for example in hardware routers. Perfect hashing has also been found to be competitive with traditional hashing in internal memory on standard computers. *Recently perfect hashing has been used* ***to accelerate algorithms on graphs*** *when the graph representation does not fit in main memory* [Belazzougui et al, 2009].

For the purposes of Natural Language Addressing (NLA) we need possibility to use *perfect hashing with dynamic and very large (practically – unlimited) set, S, of elements with variable length of strings*. In this case, the computing mapping of data elements onto machine word sized integers is not so easy to compute (we have long strings with variable length). In the same time, we could not use the bits of the input items to build a good pseudorandom sequence and use this sequence as the probe sequence for some random probing data structure, because of very large, unlimited, set, *S*, of elements.

1.3 Tries

"As defined by me, nearly 50 years ago, it is properly pronounced "tree" as in the word "retrieval". At least that was my intent when I gave it the name "Trie". The idea behind the name was to combine reference to both the structure (a tree structure) and a major purpose (data storage and retrieval)".

Edward Fredkin, July 31, 2008

Trie is a tree for storing strings in which there is one node for every common prefix. The strings are stored in extra leaf nodes.

A *trie* can be thought of as an *m*-ary tree, where *m* is the number of characters in the alphabet. A search is performed by examining the key one character at a time and using an *m*-way branch to follow the appropriate path in the trie, starting at the root. In other words, in the *multi-way trie* (Figure 5), each node has a potential child for each letter in the alphabet. Below is an example of a multi-way trie indexing the three words BE, BED, and BACCALAUREATE [Pfenning, 2012].

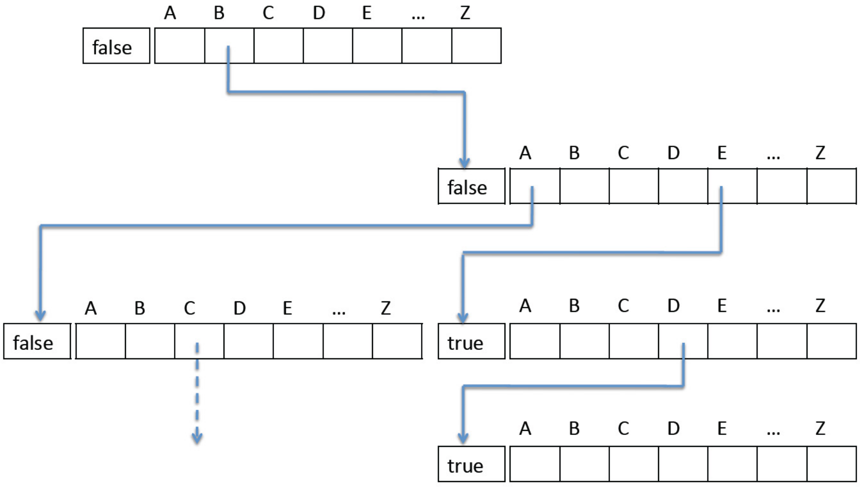


Figure 5. Example of multi-way trie [Pfenning, 2012]

Tries are distinct from the other data structures because they explicitly assume that the keys are a sequence of values over some (finite) alphabet, rather than a single indivisible entity. Thus tries are particularly well-suited for handling variable-length keys. Also, when appropriately implemented, tries can provide compression of the set represented, because common prefixes of words are combined together; words with the same prefix follow the same search path in the trie [Sahni, 2005].

To illustrate trie [Liang, 1983] had used the set of 31 most common English words (Figure 6):

A
AND
ARE
AS
AT
BE
BUT
BY
FOR
FROM
HAD
HAVE
HE
HER
HIS
I
IN
IS
IT
NOT
OF
ON
OR
THAT
THE
THIS
TO
WAS
WHICH
WITH
YOU

Figure 6. The 31 most common English words [Liang, 1983]

Figure 7 shows a linked trie representing this set of words. In a linked trie, the m-way branch is performed using a sequential series of comparisons.

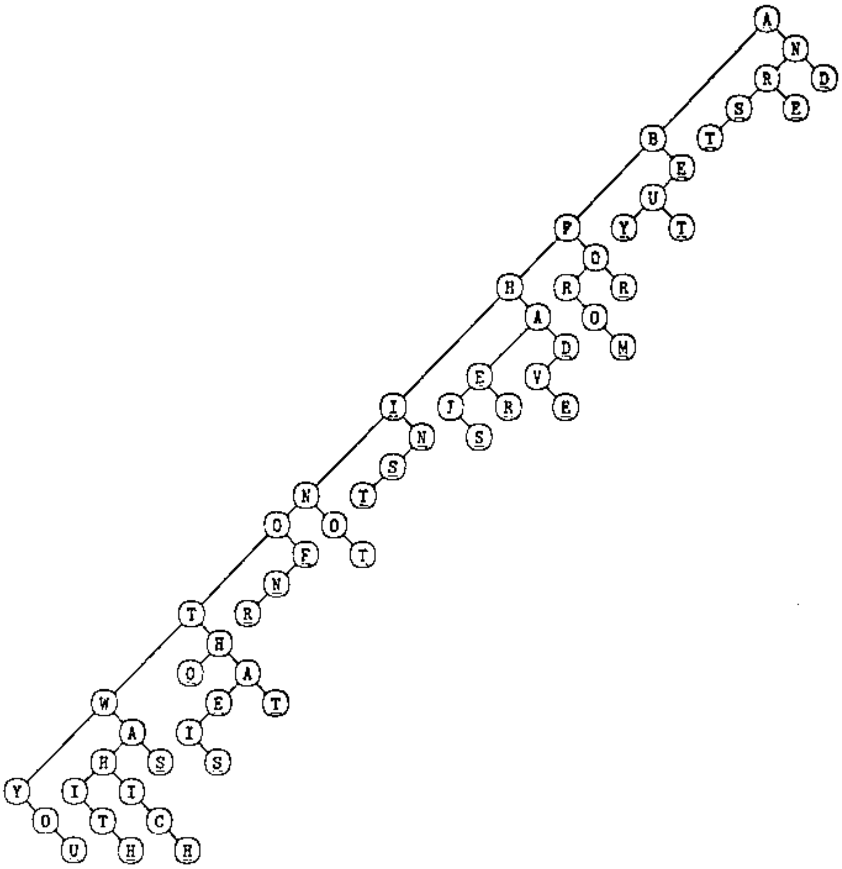


Figure 7. Linked trie for the 31 most common English words [Liang, 1983].

Suppose that the elements in our dictionary are student records that contain fields such as student name and social security number (SS#) [Sahni, 2005]. The key field is the social security number, which is a nine digit decimal number. To keep the example manageable, assume we have only five elements.

Table 1 shows the name and SS# fields for each of the five elements in our dictionary.

Table 1. Five students' records [Sahni, 2005]

Name	Social Security Number (SS#)
Jack	951-94-1654
Jill	562-44-2169
Bill	271-16-3624
Kathy	278-49-1515
April	951-23-7625

To obtain a trie representation for these five elements, we first select a radix that will be used to decompose each key into digits. If we use the radix 10, the decomposed digits are just the decimal digits shown in Table 1. We shall examine the digits of the key field (i.e., SS#) from left to right. Using the first digit of the SS#, we partition the elements into three groups–elements whose SS# begins with 2 (i.e., Bill and Kathy), those that begin with 5 (i.e., Jill), and those that begin with 9 (i.e., April and Jack). Groups with more than one element are partitioned using the next digit in the key. This partitioning process is continued until every group has exactly one element in it (Figure 8) [Sahni, 2005].

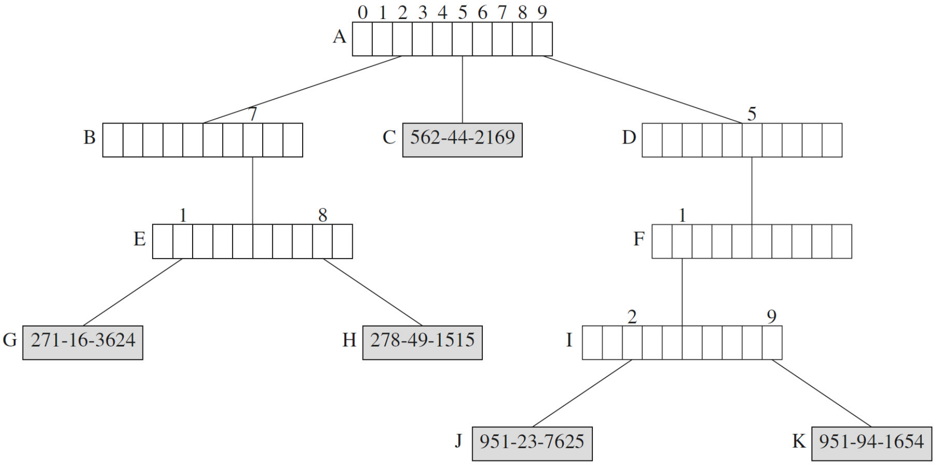


Figure 8. Trie for the elements of Table 1 [Sahni, 2005]

The partitioning process described above naturally results in a tree structure that has 10-way branching as is shown in Figure 8. The tree employs two types of nodes:

- Branch nodes;
- Element nodes.

Each branch node has 10 children (or pointer/reference) fields. These fields, child[0 : 9], have been labeled 0, 1, ..., 9 for the root node of Figure 8 ***root.child[i]*** points to the root of a sub-trie that contains all elements whose first digit is ***i***.

In Figure 8, nodes A, B, D, E, F, and I are branch nodes.

The remaining nodes, nodes C, G, H, J, and K are element nodes. Each element node contains exactly one element. In Figure 8, only the key field of each element is shown in the element nodes.

➢ *Burst Tries*

The tree data structures compared to hashing have three sources of inefficiency [Heinz et al, 2002]:

- First, the average search lengths is surprisingly high, typically exceeding ten pointer traversals and string comparisons even on moderate-sized data sets with highly skew distributions. In contrast, a search under hashing rarely requires more than a string traversal to compute a hash value and a single successful comparison;
- Second, for structures based on Binary Search Trees (BSTs), the string comparisons involved redundant character inspections, and were thus unnecessarily expensive. For example, given the query string "middle" and given that, during search, "Michael" and "midfield" have been encountered, it is clear that all subsequent strings inspected must begin with the prefix "mi";
- Third, in tries the set of strings in a sub-trie tends to have a highly skew distribution: typically the vast majority of accesses to a sub-trie are to find one particular string. Thus use of a highly time-efficient, space-intensive structure for the remaining strings is not a good use of resources [Heinz et al, 2002].

These considerations led to the burst trie. A ***burst trie*** is an *in-memory* data structure, designed for sets of records that each has a unique string that identifies the record and acts as a key. Formally, a string ***s*** with length *n* consists of a series of symbols or characters c_i for *i*=0;...;*n*, chosen from an alphabet A of size |A|. It is assumed that |A| is small, typically no greater than 256 [Heinz et al, 2002].

A ***burst trie*** consists of three distinct components (Figure 9): a set of records, a set of containers, and an access trie:

- ***Records***. A record contains a string; information as required by the application using the burst trie (that is, for information such as statistics or word locations); and pointers as required to maintain the container holding the record. *Each string is unique*;
- ***Containers***. A container is a small set of records, maintained as a simple data structure such as a list or a *binary search tree* (BST). For a container at depth *k* in a burst trie, all strings have length at least k and the first k characters of all strings are identical. It is not necessary to store these first k characters. Each container also has a header, for storing the statistics used by heuristics for bursting. Thus a particular container at depth 3 containing "author" and "automated" could also contain "autopsy" but not "auger";
- ***Access trie***. An access trie is a trie whose leaves are containers. Each node consists of an array *p*, of length |A|, of pointers, each of which may point to either a trie node or a container, and a single empty-string pointer to a record. The |A| array locations are indexed by the characters *c*∈A. The remaining pointer is indexed by the empty string.

The depth of the root is defined to be 1. Leaves are at varying depths.

A burst trie can be viewed as a generalization of other proposed variants of trie.

Figure 9 shows an example of a burst trie storing ten records whose keys are “came”, “car”, “cat”, “cave”, “cy”, “cyan”, “we”, “went”, “were”, and “west” respectively. In this example, the alphabet A is the set of letters from A to Z, and in addition an empty string symbol ⊥ is shown; the container structure used is a BST. In this figure, the access trie has four nodes, the deepest at depth 3. The leftmost container has four records, corresponding to the strings “came”, “car”, “cat”, and “cave”. One of the strings in the rightmost container is “⊥”, corresponding to the string “we”. The string “cy” is stored wholly within the access trie, as shown by the empty-string pointer to a record, indexed by the empty string [Heinz et al, 2002].

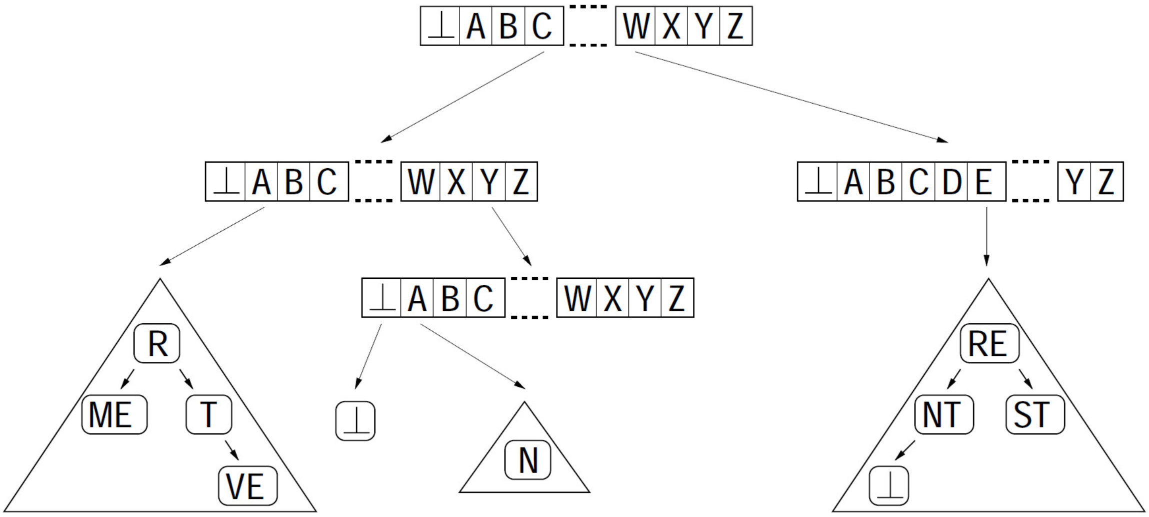


Figure 9. Burst trie with BSTs used in containers [Heinz et al, 2002]

Conclusion of Chapter 1

This chapter was aimed to introduce the theoretical surroundings of our work.

Firstly in this chapter, we remembered the needed basic mathematical concepts. Special attention was paid to the Names Sets – mathematical structure which we implemented in our research. We used strong hierarchies of named sets to create a specialized mathematical model for new kind of organization of information bases called “Multi-Dmain Information Model” (MDIM). The “information spaces” defined in the model are kind of strong hierarchies of enumerations (named sets).

We will realize MDIM via special kind of hashing. Because of this, we remembered the main features of hashing and types of hash tables as well as the idea of “Dynamic perfect hashing” and “Trie”, especially – the “Burst trie”. A ***burst trie*** *is an in-memory data structure, designed for sets of records that each has a unique string that identifies the record and acts as a key. Burst trie consists of three distinct components: a set of records, a set of containers, and an access trie.*

2 Storing models

Abstract

This chapter is aimed to introduce the main data structures and storing technologies which we will use to compare our results. Mainly they are graph data models as well as Resource Description Framework (RDF) storage and retrieval technologies.

Firstly we shortly define concepts of storage model and data model.

Mapping of the data models to storage models is based on program tools called "access methods". Their main characteristics will be outlined.

During the eighties of the last century, the total growing of the research and developments in the computers' field, especially in image processing, data mining and mobile support cause impetuous progress of establishing convenient "spatial information structures" and "spatial-temporal information structures" and corresponding access methods. Important cases of spatial representation of information are Graph models. Because of this, Graph models and databases will be discussed more deeply and examples of different graph database models will be presented. The need to manage information with graph-like nature especially in RDF-databases has reestablished the relevance of this area.

In accordance with this, the analyses of RDF databases as well as of the storage and retrieval technologies for RDF structures will be in the center of our attention. Storing models for several popular ontologies and summary of main types of storing models for ontologies and, in particular, RDF data, will be discussed.

At the end of this chapter, our attention will be paid to addressing and naming (labeling) in graphs with regards to introducing the Natural Language Addressing (NL-addressing) in graphs. A sample graph will be analyzed to find its proper 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 example, the advantages and disadvantages of the multi-layer representation of graphs will be outlined.

The specialized mathematical model for new kind of organization of information bases called "Multi-Dmain Information Model" (MDIM) and its realizations will be presented.

2.1 Storage model and Data model

Let remember that the "***data storage***" is a part of a computer that stores information for subsequent use or retrieval [AHD, 2009]. It is a device consisting of electronic, electrostatic, electrical, hardware, or other elements into which data may be entered, and from which data may be obtained as desired. For instance it may be magnetic tapes, hard drive storage, network storage, removable media (USB devices, flash drives, SD cards, DVDs), and online storage (Cloud storage [Mell & Grance, 2011]) [Greenwood, 2012].

The "***storage model***" is a model that captures key *physical* aspects of data structure in a data store. The ***storage schema*** (internal schema) is a specification of how the data relationships and rules specified in the logical schema of a database will be mapped to the physical storage level in terms of the available constructs, such as aggregation into records, clustering on pages, indexing, and page sizing and caching for transfer between secondary and primary storage. Storage schema facilities vary widely between different **D**ata**B**ase **M**anagement **S**ystems (DBMS) [Daintith, 2004].

On the other hand, a **data model** is a model that captures key *logical* aspects of data structure in a database, i.e. underlying the structure of a database is a *data model*. A data model is a collection of conceptual tools for describing the real-world entities to be modeled in the database and the relationships among these entities. Data models differ in the primitives available for describing data and in the amount of semantic detail that can be expressed. The various data models that have been proposed fall into three different groups: *object-based* logical models, *record-based* logical models, and *physical* data models. Physical data models are used to describe data at the lowest level. [Silberschatz et al, 1996].

There is multitude of reviews and taxonomies of data models [Silberschatz et al, 1996; Navathe, 1992; Beeri, 1988; Kerschberg et al, 1976]. An evolutionary scheme of the most important and widely accepted DataBase (DB) models is outlined in Figure 10. Rectangles denote database models (db-models), arrows indicate influences, and circles denote theoretical developments. A time-line in years is shown on the left [Angles & Gutierrez, 2008].

From a database point of view, the conceptual tools that make up a db-model should at least address data structuring, description, maintenance, and a way to retrieve or query the data. According to these criteria, a db-model consists of three components [Codd, 1980]:

- A set of data structure types;
- A set of operators or inference rules;
- A set of integrity rules.

Several proposals for db-models only define the data structures, sometimes omitting operators and/or integrity rules [Angles & Gutierrez, 2008].

In addition, each db-model proposal is based on certain theoretical principles, and serves as base for the development of related models.

The short overview of Database Models (db-models) below follows one given in [Angles & Gutierrez, 2008].

Before the advent of the relational model, most db-models focused essentially on the specification of data structures on actual file systems (Figure 10).

At this time the main information structure is the "record". Let remember that the "record" is a logical sequence of fields which contain data eventually connected to unique identifier (a "key"). The identifier (key) is aimed to distinguish one sequence from another [Stably, 1970]. The records are united in the sets, called "files". There exist three basic formats of the records – with *fixed*, *variable* and *undefined* length.

In the *context-free file models*, storing of the records is not connected to their content and depends only on external factors – the sequence, disk address or position in the file. The main idea of *the context-depended file models* is that the part of the record is selected as a key which is used for making decision where to store the record and how to search it. This way the content of the record influences on the access to the record [Markov et al, 2008].

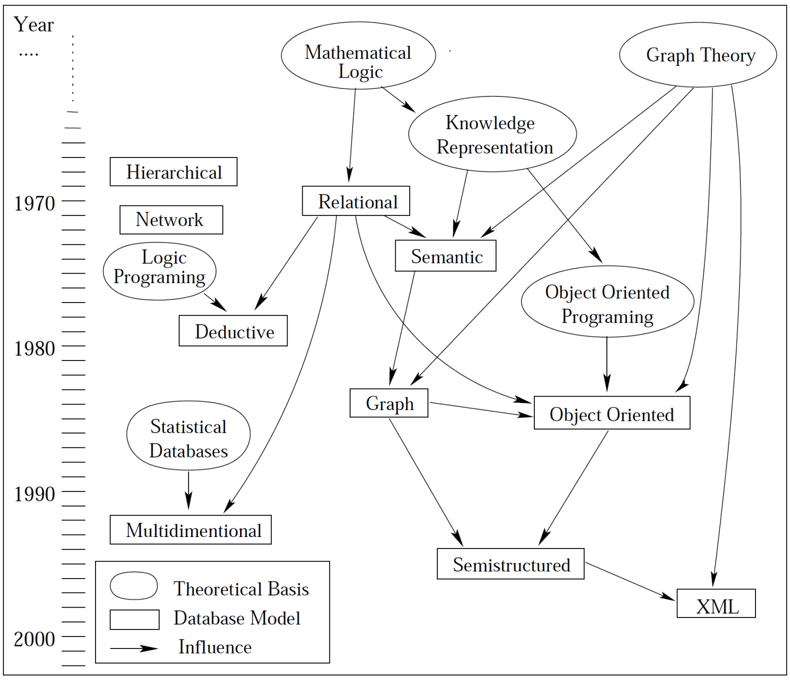


Figure 10. Evolutionary scheme of DB-models [Angles & Gutierrez, 2008]

Modern DataBase Management Systems (DBMS) are built using context-depended file models such as: unsorted sequential files with records with keys; sorted files with fixed record length; static or dynamic hash files; index file and files with data; clustered indexed tables [Connolly & Begg, 2002].

Two representative database models are the *hierarchical* [Tsichritzis & Lochovsky, 1976] and the *network* [Taylor & Frank, 1976] models, both of which place emphasis on the physical level.

The *relational db-model* was introduced by Codd [Codd, 1980] and highlights the concept of abstraction levels by introducing the idea of separation between physical and logical levels. It is based on the notions of sets and relations.

As opposed to previous models, *semantic db-models* [Peckham & Maryanski, 1988] allow database designers to represent objects and their relations in a natural and clear manner, providing users with tools to faithfully capture the desired domain semantics. A well-known example is the entity-relationship model [Chen, 1976].

Object-oriented db-models [Kim, 1990] appeared in the eighties, when most of the research was concerned with so-called "advanced systems for new types of applications [Beeri, 1988]". These db-models are based on the object-oriented paradigm and their goal is to represent data as a collection of objects, which are organized into classes, and are assigned complex values.

Graph db-models made their appearance alongside object-oriented db-models. These models attempt to overcome the limitations imposed by traditional db-models with respect to capturing the inherent graph structure of data appearing in applications such as hypertext or geographic information systems, where the interconnectivity of data is an important aspect. This type of models is outlined further in this text.

Semi-structured db-models [Buneman, 1997] are designed to model data with a flexible structure, for example, documents and Web pages. Semi-structured data is neither raw nor strictly typed, as in conventional database systems. These db-models appeared in the nineties. Further in this chapter we will outline such type model called Resource Description Framework (RDF).

Closely related to them is the *XML (eXtensible Markup Language)* [Bray et al, 1998] model, which did not originate in the database community. Although originally introduced as a document exchange standard, it soon became a general purpose model, focusing on information with tree-like structure [Angles & Gutierrez, 2008].

Mapping of the data models to storage models is based on program tools called "*access methods"*.

2.2 Memory management and access methods

Memory management is a complex field of computer science. Over the years, many techniques have been developed to make it more efficient [Ravenbrook, 2010]. Memory management is usually divided into three areas: *hardware*, *operating system*, and *applications*, although the distinctions are a little fuzzy. In most computer systems, all three are present to some extent, forming layers between the user's program and the actual memory hardware:

- **Memory management at the hardware level** is concerned with the electronic devices that actually store data. This includes things like RAM, Associative memory, and memory caches [Mano, 1993];
- **Memory in the operating system** must be allocated to user programs, and reused by other programs when it is no longer required. The operating system can pretend that the computer has more memory than it actually does, and that each program has the machine's memory to itself. Both of these are features of *virtual memory* systems;
- **Application memory management** involves supplying the memory needed for a program's objects and data structures from the limited resources available, and recycling that memory for reuse when it is no longer required. Because in general, application programs cannot predict in advance how much memory they are going to require, they need additional code to handle their changing memory requirements.

Application memory management combines two related tasks:

- **Allocation**: when the program requests a block of memory, the memory manager must allocate that block out of the larger blocks it has received from the operating system. The part of the memory manager that does this is known as the *allocator*;
- **Recycling**: when memory blocks have been allocated, but the data they contain is no longer required by the program, the blocks can be recycled for reuse. There are two approaches to recycling memory: either the programmer must decide when memory can be reused (known as *manual memory management*); or the memory manager must be able to work it out (known as *automatic memory management*).

The progress in memory management gives the possibility to allocate and recycle not directly blocks of the memory but structured regions or fields corresponding to some types of data. In such case, we talk about corresponded "*access methods*".

The **Access Methods (AM)** had been available from the beginning of the development of computer peripheral devices. As many devices so many possibilities for developing different AM there exist. Our attention is focused only to the access methods for devices for permanently storing the information with direct access such as magnetic discs, flash memories, etc. [Markov et al, 2008].

In the beginning, the AM were functions of the Operational Systems' Core or so called Supervisor, and were executed via corresponding macro-commands in the assembler languages [Stably, 1970] or via corresponding input/output operators in the high level programming languages like FORTRAN, COBOL, PL/I, etc.

The establishment of the first databases in the sixties of the previous century caused gradually accepting the concepts "physical" as well as "logical" organization of the data [CODASYL, 1971; Martin, 1975]. In 1975, the concepts "access method", "physical organization" and "logical organization" became clearly separated. In the same time Christopher Date [Date, 1977] wrote:

"The DataBase Management System (DBMS) does not know anything about:

a) Physical records (blocks);
b) How the stored fields are integrated in the records (nevertheless that in many cases it is obviously because of their physical disposition);
c) How the sorting is realized (for instance it may be realized on the base of physical sequence, using an index or by a chain of pointers);
d) How is realized the direct access (i.e. by index, sequential scanning or hash addressing).

This information is a part of the structures for data storing but it is used by the access method but not by the DBMS".

Every access method presumes an exact organization of the file, which it is operating with and is not related to the interconnections between the files, respectively, – between the records of one file and that in the others files. These interconnections are controlled by the physical organization of the DBMS [Date, 2004].

Therefore, in the DBMS we may distinguish four levels:

- Basic access methods of the core (supervisor) of the operation system;
- Specialized access methods realized using basic access methods;
- Physical organization of the DBMS;
- Logical organization of the DBMS.

During the eighties of the last century, the total growing of the research and developments in the computers' field, especially in image processing, data mining and mobile support cause impetuous progress of establishing convenient "spatial information structures" and "spatial-temporal information structures" and corresponding access methods. From different points of view, this period has been presented in [Ooi et al, 1993; Gaede & Günther, 1998; Arge, 2002; Mokbel et al, 2003; Moënne-Loccoz, 2005]. Usually, the "one-dimensional" (linear) AM are used in the classical applications, based on the alphanumerical information, whereas the "multi-dimensional" (spatial) methods are aimed to serve the work with graphical, visual, multimedia information [Markov et al, 2013].

2.3 Interconnections between raised access methods

Maybe one of the most popular analyses of the genesis of the access methods is given in [Gaede & Günther, 1998]. The authors presented a scheme of the genesis of the basic multi-dimensional AM and theirs modifications. This scheme firstly was proposed in [Ooi et al, 1993] and it was expanded in [Gaede & Günther, 1998]. An extension in direction to the multi-dimensional spatio-temporal access methods was given in [Mokbel et al, 2003].

The survey [Markov et al, 2008] presents a new variant of this scheme, where the new access methods, created after 1998, are added. A comprehensive bibliography of corresponded articles, where the methods are firstly presented, is given.

The access methods, presented on Figure 11 [Markov et al, 2008] may be classified as follow:

- One-dimensional AM:
    - Context free;
    - Context depended;
- Multidimensional Spatial AM:
    - Point AM:
        - Multidimensional Hashing;
        - Hierarchical Access Methods;
        - Space Filling Curves for Point Data;
    - Spatial AM:
        - Transformation;
        - Overlapping Regions;
        - Clipping;
        - Multiple Layers;
- Metric Access Methods;
- High Dimensional Access Methods:
    - Data Approximation;
    - Query Approximation:
        - Clustering of the database;
        - Splitting the database;
- Spatio-Temporal Access Methods:
    - Indexing the past;
    - Indexing the present;
    - Indexing the future.

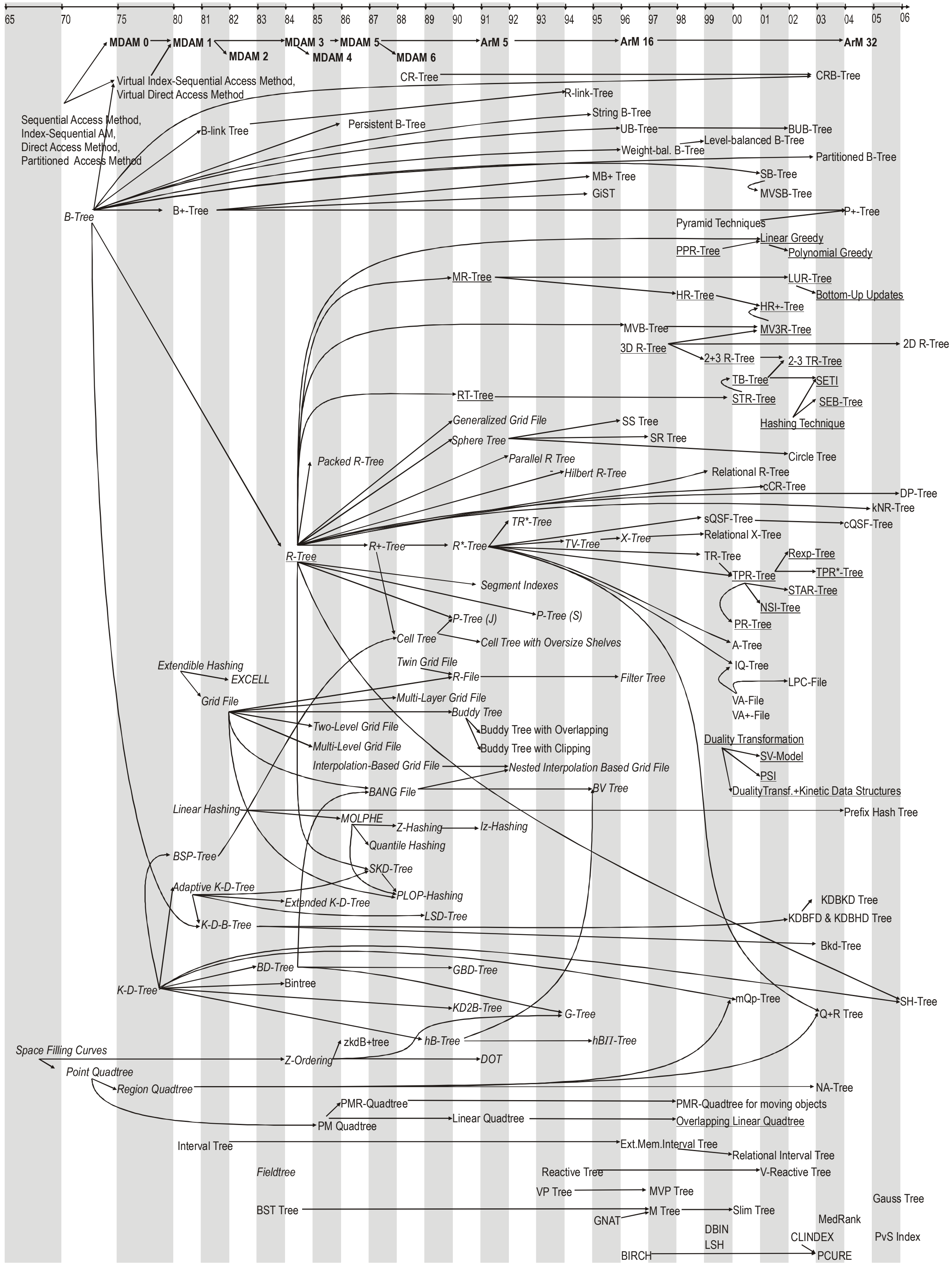


Figure 11. Genesis of the Access Methods and their modifications extended variant of [Gaede & Günther, 1998; Mokbel et al, 2003] presented in [Markov et al, 2008]

➢ *One-dimensional access methods*

One-dimensional AM are based on the concept "*record*". The "record" is a logical sequence of fields, which contain data eventually connected to unique identifier (a "key"). The identifier (key) is aimed to distinguish one sequence from another [Stably, 1970]. The records are united in the sets, called "*files*". There exist three basic formats of the records – with fixed, variable, and undefined length.

In the ***context-free methods***, the storing of the records is not connected to their content and depends only on external factors – the sequence, disk address, or position in the file. The necessity of stable file systems in the operating systems does not allow a great variety of the context-free AM. There are three main types well known from sixties and seventies: *Sequential Access Method* (SAM); *Direct Access Method* (DAM) and *Partitioned Access Method* (PAM) [IBM, 1965-68].

The main idea of the ***context-depended AM*** is that a part of the record is selected as a key, which is used for making decision where to store the record and how to search it. This way, the content of the record influences the access to the record.

Historically, from the sixties of the previous century on, the attention is directed mainly to the second type of AM. Modern DBMS are built using context-depended AM such as: unsorted sequential files with records with keys; sorted files with fixed record length; static or dynamic hash files; index files and files with data; clustered indexed tables [Connolly & Begg, 2002].

➢ *Multidimensional spatial access methods*

Multidimensional Spatial Access Methods are developed to serve information about spatial objects, approximated with points, segments, polygons, polyhedrons, etc. The implementations are numerous and include traditional multi-attributive indexing, geographical and/or information systems for global monitoring for environment and security, spatial databases, content indexing in multimedia databases, etc.

From the point of view of the spatial databases, the access methods can be split into two main classes of access methods – *Point Access Methods* and *Spatial Access Methods* [Gaede & Günther, 1998].

Point Access Methods are used for organizing multidimensional point objects. Typical instances are traditional records, where every attribute of the relation corresponds to one dimension. These methods can be separated in three basic groups:

- Multidimensional Hashing (for instance Grid File and its varieties, EXCELL, Twin Grid File, MOLPHE, Quantile Hashing, PLOP-Hashing, Z-Hashing, etc);
- Hierarchical Access Methods (includes such methods as KDB-Tree, LSD-Tree, Buddy Tree, BANG File, G-Tree, hB-Tree, BV-Tree, etc.);
- Space Filling Curves for Point Data (like Peano curve, N-trees, Z-Ordering, etc).

Spatial Access Methods are used for working with objects, which have an arbitrary form. The main idea of the spatial indexing of non-point objects is to use an approximation of the geometry of the examined objects as more simple forms. The most used approximation is Minimum Bounding Rectangle (MBR), i.e. minimal rectangle, which sides are parallel of the coordinate axes and completely include the object. There exist approaches for approximation with Minimum Bounding Spheres (SS Tree) or other polytopes (Cell Tree), as well as their combinations (SR-Tree) [Gaede & Günther, 1998].

The usual problem when one operates with spatial objects is their overlapping. There are different techniques to avoid this problem. From the point of view of the techniques for the organization of the spatial objects, Spatial Access Methods can be split in four main groups:

- **Transformation** – this technique uses transformation of spatial objects to points in the space with more or less dimensions. Most of them spread out the space using space filling curves (Peano Curves, z-ordering, Hibert curves, Gray ordering, etc.) and then use some point access method upon the transformed data set;
- **Overlapping Regions** – here the data sets are separated in groups; different groups can occupy the same part of the space, but every space object is associated with only one of the groups. The access methods of this category operate with data in their primary space (without any transformations) eventually in overlapping segments. Methods which use this technique includes R-Tree, R-link-Tree, Hilbert R-Tree, R*-Tree, Sphere Tree, SS-Tree, SR-Tree, TV-Tree, X-Tree, P-Tree of Schiwietz, SKD-Tree, GBD-Tree, Buddy Tree with overlapping, PLOP-Hashing, etc.;
- **Clipping** – this technique uses the clipping of one object to several sub-objects, which will be stored. The main goal is to escape overlapping regions. However this advantage can lead to the tearing of the objects, extending the resource expenses, and decreasing the productivity of the method. Representatives of this technique are R+-Tree, Cell-Tree, Extended KD-Tree, Quad-Tree, etc.;
- **Multiple Layers** – this technique can be considered as a variant of the techniques of Overlapping Regions, because the regions from different layers can overlap. Nevertheless there exist some important differences: first – the layers are organized hierarchically; second – every layer splits the primary space in a different way; third – the regions of one layer never overlaps; fourth – the data regions are separated from the space extensions of the objects. Instances for these methods are Multi-Layer Grid File, R-File, etc.

➢ *Metric access methods*

Metric Access Methods deal with relative distances of data points to chosen points, named anchor points, vantage points or pivots [Moënne-Loccoz, 2005]. These methods are designed to limit the number of distance computation, calculating first distances to anchors, and then finding the searched point in a narrowed region. These methods are preferred when the distance is highly computational, as e.g. for the dynamic time warping distance between time series. Representatives of these methods are: Vantage Point Tree (VP Tree), Bisector Tree (BST-Tree), Geometric Near-Neighbor Access Tree (GNNAT), as well as the most effective from this group – Metric Tree (M Tree) [Chavez et al, 2001].

➢ *High dimensional access methods*

Increasing the dimensionality strongly aggravates the qualities of the multidimensional access methods. Usually, these methods exhaust their possibilities at dimensions around **15**. Only X-Tree reaches the boundary of **25** dimensions, after which this method gives worse results then sequential scanning [Chakrabarti, 2001].

The exit of this situation is based on the data approximation and query approximation in sequential scan. These methods form a new group of access methods – High Dimensional Access Methods.

Data approximation is used in VA-File, VA+-File, LPC-File, IQ-Tree, A-Tree, P+-Tree, etc.

For **query approximation**, two strategies can be used:

- Examine only a part of the database, which is more probably to contain the resulting set – as a rule these methods are based on the clustering of the database. Some of these methods are: DBIN, CLINDEX, PCURE;
- Splitting the database to several spaces with fewer dimensions and searching in each of them. Here two main methods are used:
  - Random Lines Projection. Representatives of this approach are MedRank, which uses B+-Tree for indexing every arbitrary projection of the database, and PvS Index, which consist of combination of iterative projections and clustering.
  - Locality Sensitive Hashing, which is based on the set of local-sensitive hashing functions [Moënne-Loccoz, 2005].

➢ *Spatio-temporal access methods*

The Spatio-Temporal Access Methods have additional defined time dimensioning [Mokbel et al, 2003]. They operate with objects, which change their form and/or position during the time. According to position of time interval in relation to present moment, the Spatio-Temporal Access Methods are divided to:

- ***Indexing the past,*** i.e. these methods operate with historical spatio-temporal data. The problem here is the continuous increase of the information over time. To overcome the overflow of the data space two approaches are used – sampling the stream data at certain time position or updating the information only when data is changed. Spatio-temporal indexing schemes for historical data can be split in three categories:
  - The first category includes methods that manage spatial and temporal aspects into already existing spatial data;
  - The second category can be explained as snapshots of the spatial information in each time instance;
  - The third category focuses on trajectory-oriented queries, while spatial dimension lag on second priority.

  Representatives of this group are: RT-Tree, 3DR-Tree, STR-Tree, MR-Tree, HR-Tree, HR+-Tree, MV3R-Tree, PPR-Tree, TB-Tree, SETI, SEB-Tree;

- **Indexing the present**. In contrast to previous methods, where all movements are known, here the current positions are neither stored nor queried. Some of the methods, which answer the questions of the current position of the objects are 2+3R-Tree, 2-3TR-Tree, LUR-Tree, Bottom-Up Updates, etc.;
- **Indexing the future**. These methods have to answer the questions about the current and future position of a moving object – here are embraced the methods like PMR-Quadtree for moving objects, Duality Transformation, SV-Model, PSI, PR-Tree, TPR-Tree, TPR*-tree, NSI, VCIR-Tree, STAR-Tree, REXP-Tree.

2.4 Structured data models

Traditional database systems rely on the relational data model.

When it was proposed in the early 1970's by Codd, a logician, the relational model generated a true revolution in data management. In this simple model data is represented as relations in first order structures and queries as first order logic formulas. It enabled researchers and implementers to separate the logical aspect of the data from its physical implementation. Thirty years of research and development followed, and they led to today's mature and highly performance relational database systems [Mendelzon et al, 2001].

2.5 Semi-structured data models

The age of the Internet brought new data management applications and challenges. Data is now accessed over the Web, and is available in a variety of formats, including HTML, XML, as well as several applications specific data formats. Often data is mixed with free text, and the boundary between data and text is sometimes blurred. The way the data can be retrieved also varies considerably: some instances can be downloaded entirely; others can only be accessed through limited capabilities. To accommodate all forms and kinds of data, the database research community has introduced the ***"semi-structured data model"***, ***where data is self-describing, irregular***, ***and graph-like***. The new model captures naturally Web data, such as HTML, XML, or other application specific formats [Mendelzon et al, 2001].

The topic of semi-structured data is relatively recent [Buneman, 2001]. Applications that manage semi-structured data are becoming increasingly commonplace. Current approaches for storing semi-structured data use existing storage machinery - they either map the data to ***relational databases, or use a combination of flat files and indexes*** [Bhadkamkar et al, 2009].

In semi-structured data, the information that is normally associated with a schema contained within the data, which is sometimes called "self-describing". In some forms of semi-structured data, there is no separate schema, in others it exists but only places loose constraints on the data, Semi-structured data has recently emerged as an important topic of study for a variety of reasons. First, there are data sources such as the Web, which we would like to treat as databases but which cannot be constrained by a schema. Second, it may be desirable to have an extremely flexible format for data exchange between disparate databases. Third, even when dealing with structured data, it may be helpful to view it as semi-structured for the purposes of browsing [Buneman, 2001].

The importance of semi-structured models which are “graph-like” revived the interest to *Graph models*.

2.6 Graph models and databases

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:

- Data structures;
- Transformation language;
- 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].

➢ *Examples of graph database models*

In these examples we illustrate the most representative graph database models and other related models that do not fit properly as graph database models, but use graphs, for example, for navigation, for defining views, or as language representation.

To give a flavor of the modeling in each proposal, we will use as a running example the toy genealogy from [Angles & Gutierrez, 2008] (Figure 12). 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).

The examples below are divided into two categories:

- Graph models with explicit schema (Table 2);
- Graph models with implicit schema (Table 3).

In the both tables, on the left side of the rectangle the corresponded schema is presented and on the right side of the rectangle the examples of instances are given. For all examples in both tables a brief explanation is added.

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

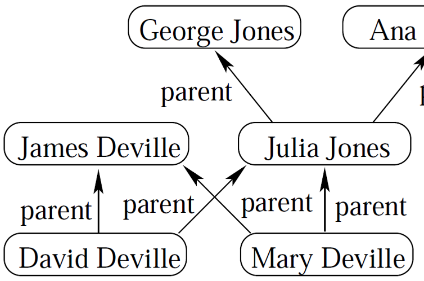
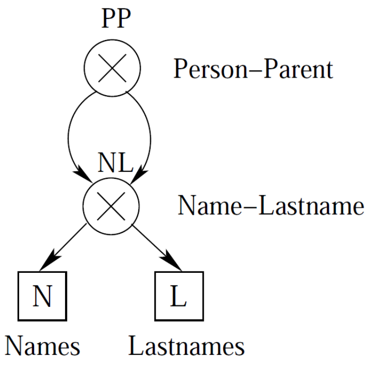


Figure 12. Running example: the toy genealogy

Table 2. Examples of the graph models with explicit schema

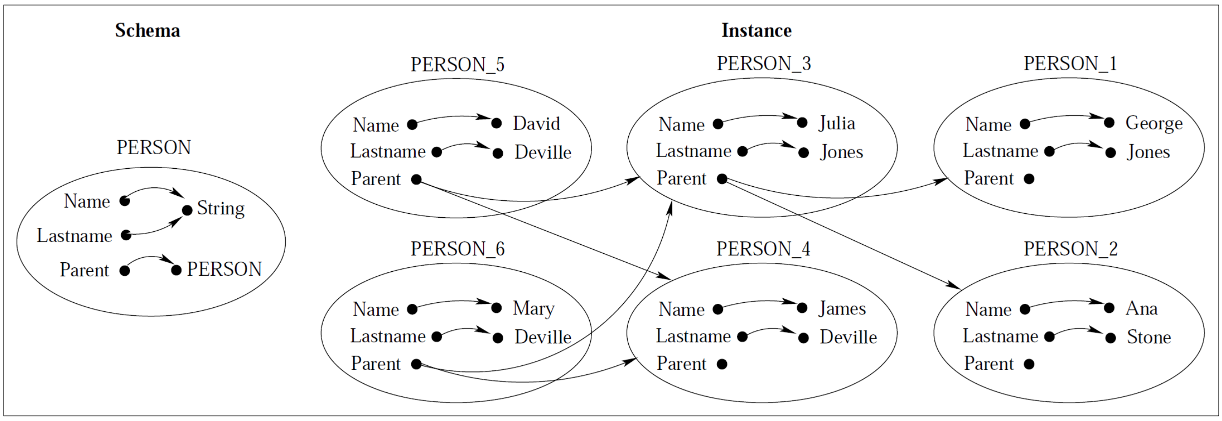
Schema



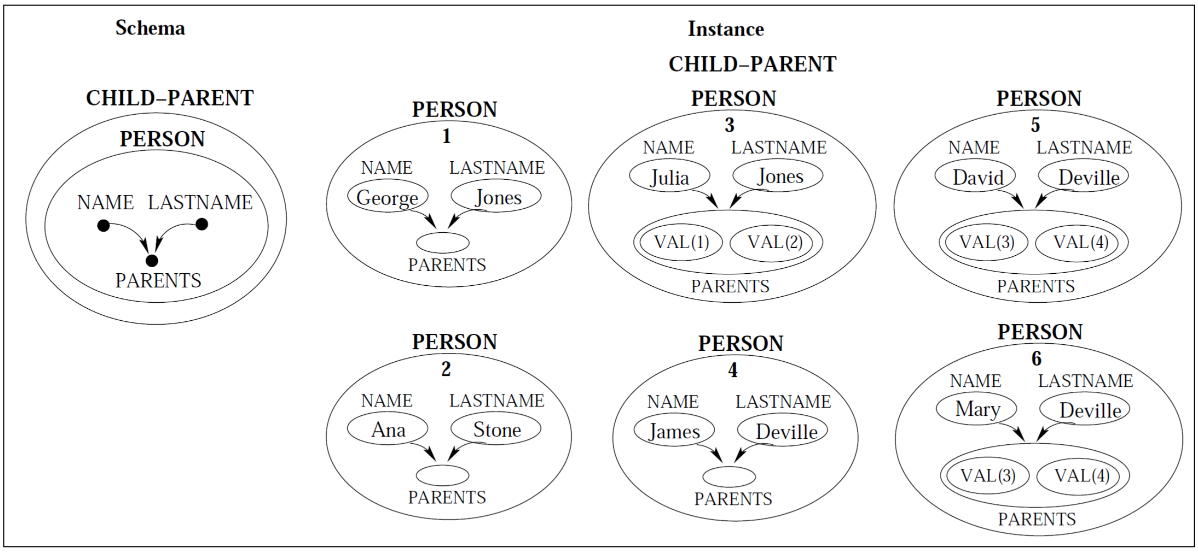
Instance

I (N)		*I (L)*		*I (NL)*		*I (PP)*	
I	*val (1)*	*I*	*val (1)*	*I*	*val (1)*	*I*	*val (1)*
1	George	7	Jones	10	(1,7)	16	(12,10)
2	Ana	8	Stone	11	(2,8)	17	(12,11)
3	Julia	9	Deville	12	(3,7)	18	(14,13)
4	James			13	(4,9)	19	(14,12)
5	David			14	(5,9)	20	(15,13)
6	Mary			15	(6,9)	21	(15,12)

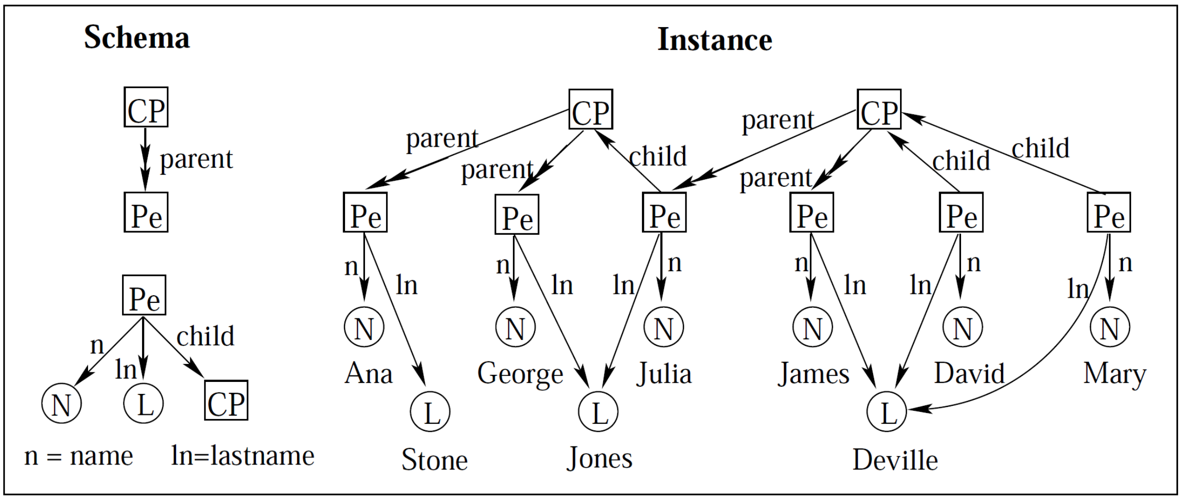
Logical Data Model (LDM). The schema (on the left) uses two basic type nodes for representing data values (N and L), and two product type nodes (NL and PP) to establish relations among data values in a relational style. The instance (on the right) is a collection of tables, one for each node of the schema. Internal nodes use pointers (names) to make reference to basic and set data values defined by other nodes [Kuper & Vardi, 1984, 1993].



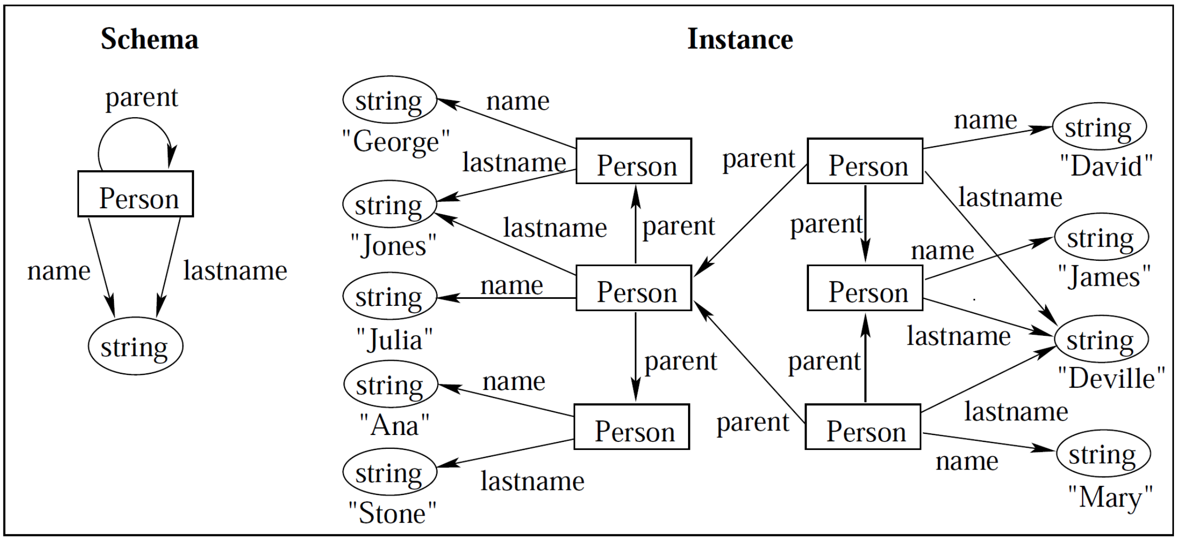
Hypernode Mode (HyM). The schema (left) defines a person as a complex object with the properties name and last name of type string, and parent of type person (recursively defined). The instance (on the right) shows the relations in the genealogy among different instances of person [Levene & Poulovassilis, 1990; Poulovassilis & Levene, 1994; Levene & Loizou, 1995].



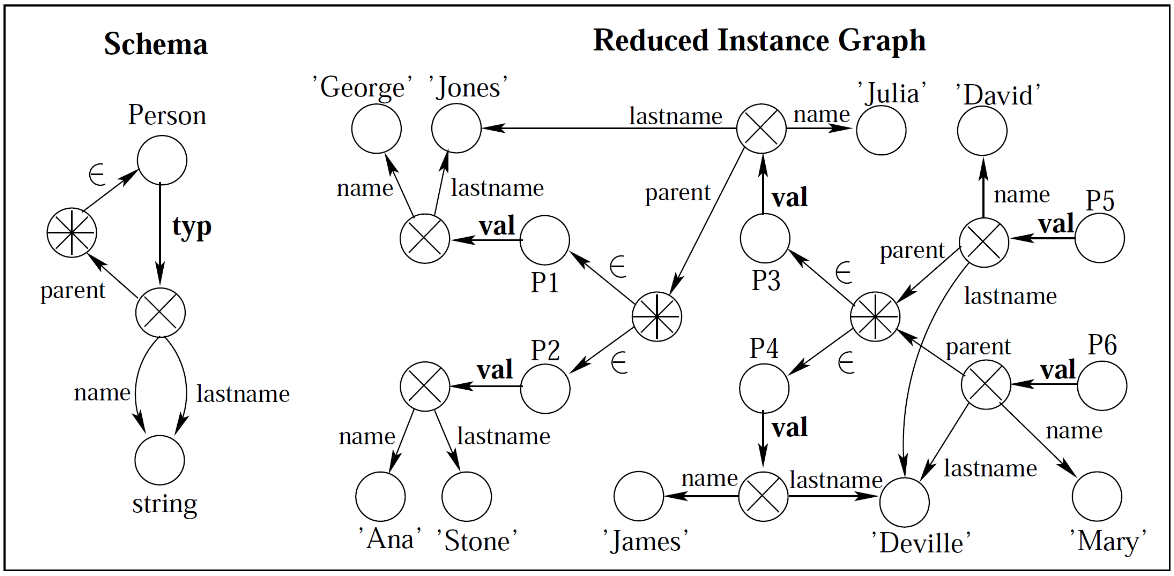
GROOVY. At the schema level (left), we model an object PERSON as a hypergraph that relates the attributes NAME, LASTNAME and PARENTS. Note the value functional dependency (VDF) NAME, LASTNAME → PARENTS logically represented by the directed hyperedge ({NAME, LASTNAME} {PARENTS}). This VFD asserts that NAME and LASTNAME uniquely determine the set of PARENTS [Levene & Poulovassilis, 1991].



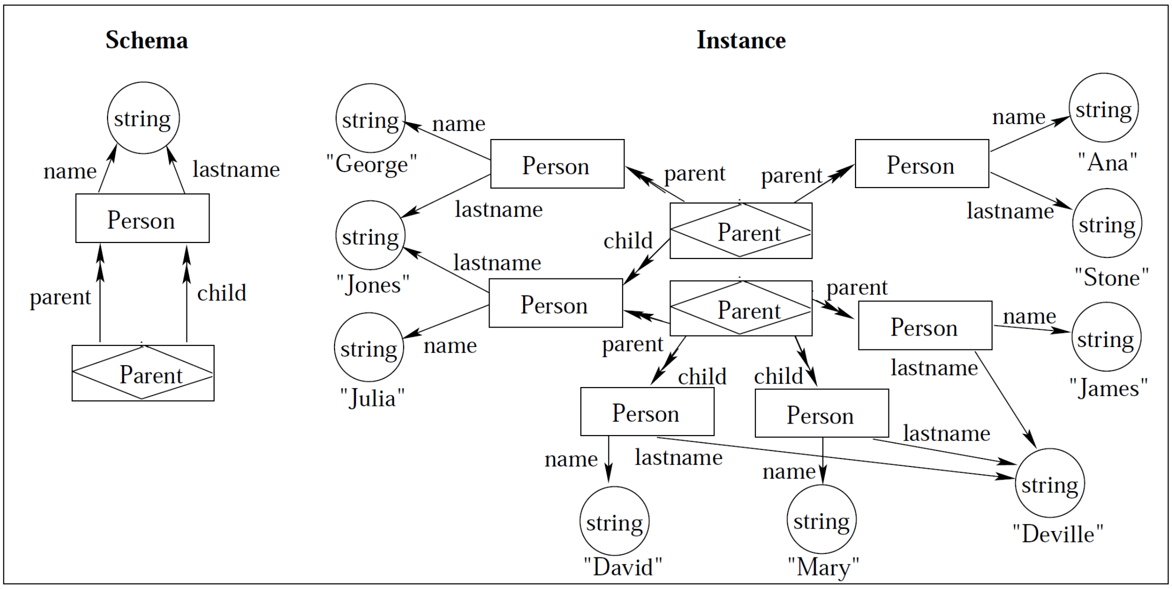
GOOD. In the schema, we use printable nodes N and L to represent names and last names respectively, and nonprintable nodes, Pe(rson) and CP, to represent relations Name-Lastname, and Child-Parent, respectively. A double arrow indicates a nonfunctional relationship, and a simple arrow indicates a functional relationship. The instance is obtained by assigning values to printable nodes and instantiating the CP and PE nodes [Gyssens et al, 1990; Gemis & Paredaens, 1993].



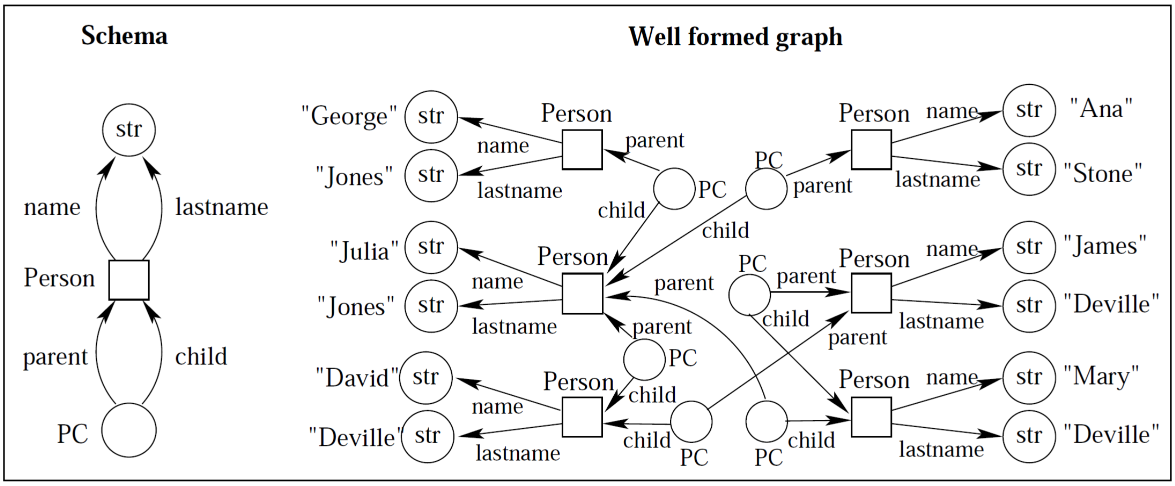
GMOD. In the schema, nodes represent abstract objects (Person) and labeled edges establish relations with primitive objects (properties name and last name), and other abstract objects (parent relation). For building an instance, we instantiate the schema for each person by assigning values to oval nodes [Andries et al, 1992].



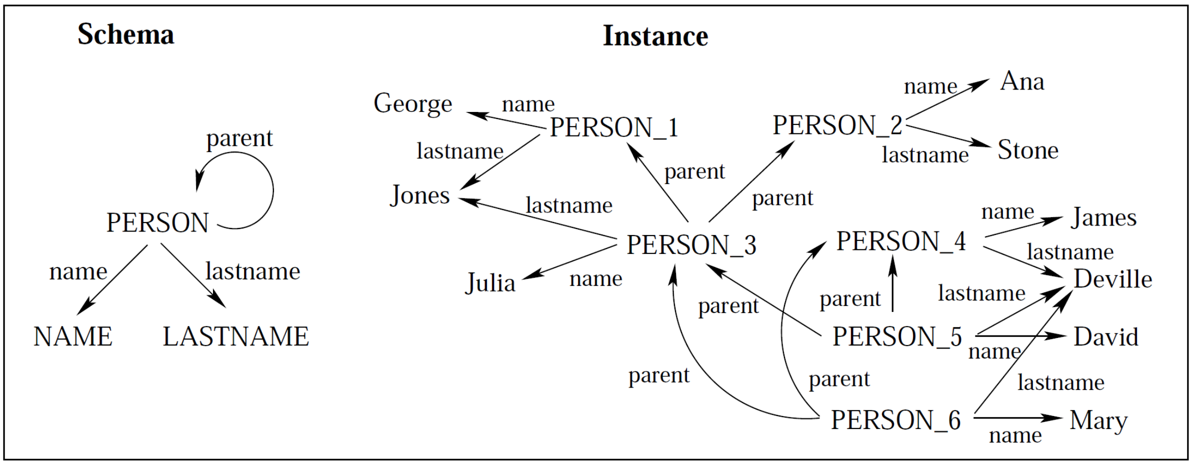
PaMaL. The example shows all the nodes defined in PaMaL: basic type (string), class (Person), tuple (⊗), set (⊛) nodes for the schema level, and atomic (George, Ana, etc.), instance (P1, P2, etc), tuple and set nodes for the instance level. Note the use of edges ∈ to indicate elements in a set, and the edge type to indicate the type of class Person (these edges are changed to val in the instance level) [Gemis & Paredaens, 1993].



GOAL. The schema presented in the example shows the use of the object node Person with properties Name and Lastname. The association node Parent and the double headed edges parent and child allow expression of the relation Person-Parent. At the instance level, we assign values to value nodes (string) and create instances for object and association nodes. Nodes with same value were merged (e.g. Deville) [Hidders & Paredaens, 1993].



GDM. In the schema each entity Person (object node represented as a square) has assigned the attributes name and last name (basic value nodes represented round and labeled str.). We use the composite value node PC to establish the relationship Parent-Child. Note the redundancy introduced by the node PC. The instance is built by instantiating the schema for each person [Hidders, 2001, 2002].



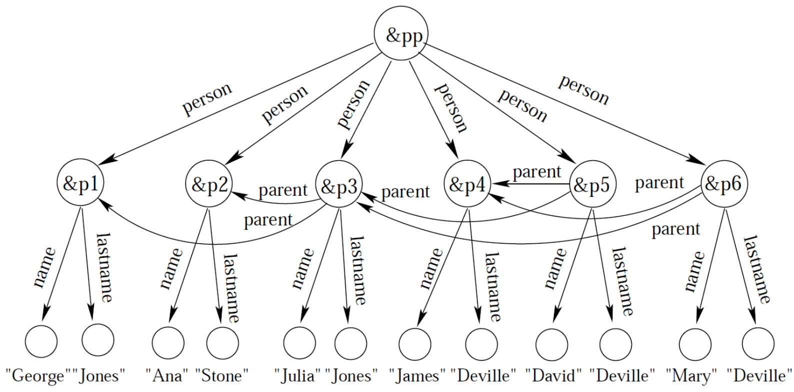
Gram. At the schema level we use generalized names for definition of entities and relations. At the instance level, we create instance labels (e.g. PERSON 1) to represent entities, and use the edges (defined in the schema) to express relations between data and entities [Amann & Scholl, 1992].

Table 3. Examples of the graph models with implicit schema

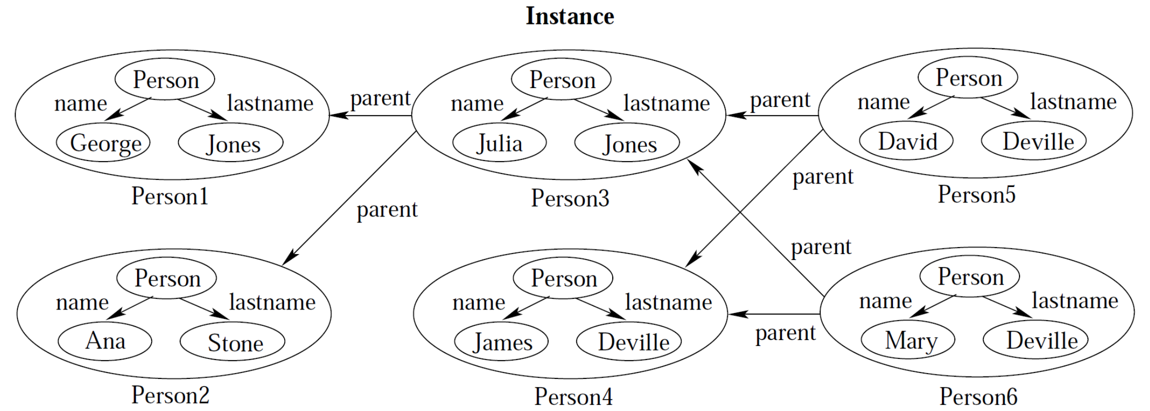
OEM Syntax

```
{ person : &p1 { name : "George" ,
                 lastname : "Jones" }
  person : &p2 { name : "Ana" ,
                 lastname : "Stone" }
  person : &p3 { name : "Julia" ,
                 lastname : "Jones" ,
                 parent : &p1 ,
                 parent : &p2 }
  person : &p4 { name : "James" ,
                 lastname : "Deville" }
  person : &p5 { name = "David",
                 lastname : "Deville" ,
                 parent : &p3 ,
                 parent : &p4 }
  person : &p6 { name = "Mary" ,
                 lastname : "Deville" ,
                 parent : &p3 ,
                 parent : &p4 } }
```

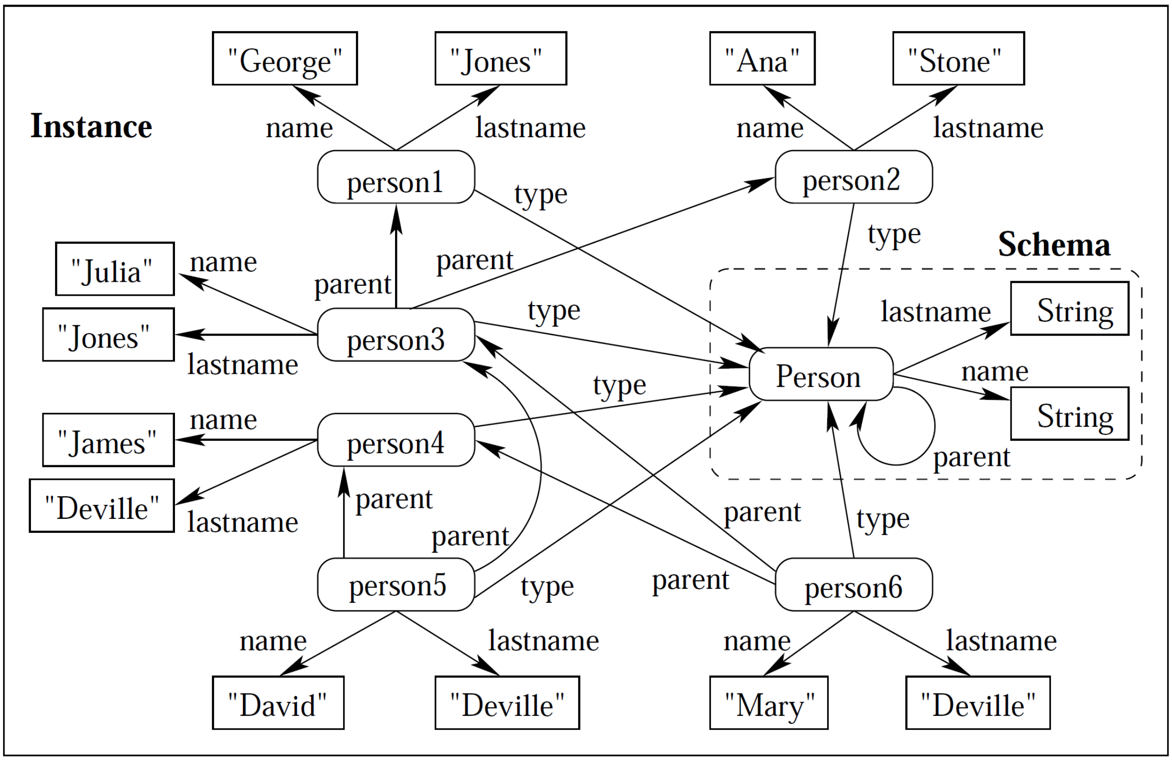
OEM Graph


&pp
person
&p1
&p2
&p3
&p4
&p5
&p6
parent
name
lastname
"George" "Jones" "Ana" "Stone" "Julia" "Jones" "James" "Deville" "David" "Deville" "Mary" "Deville"

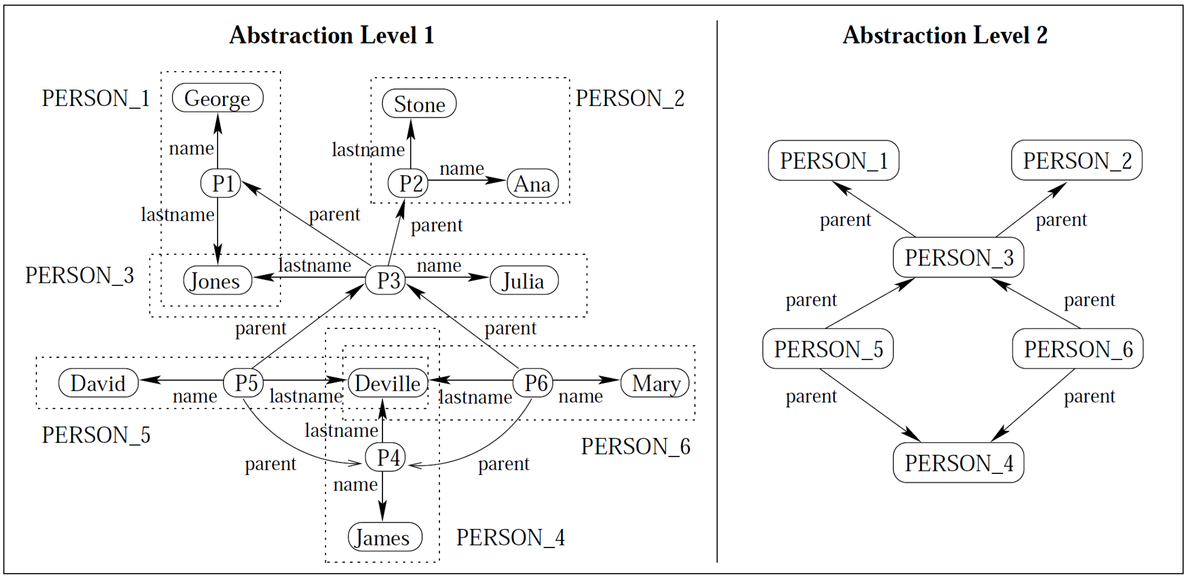
Object Exchange Model (OEM). Schema and instance are mixed. The data is modeled beginning in a root node &pp, with children person nodes, each of them identified by an Object-ID (e.g. &p2). These nodes have children that contain data (name and last name) or references to other nodes (parent). Referencing permits establishing relations between distinct hierarchical levels. Note the tree structure obtained if one forgets the pointers to OIDs, a characteristic of semi structured data [Papakonstantinou et al, 1995].



GGL. Schema and instances are mixed. Packaged graph vertices (Person1, Person2, ...) are used to encapsulate information about the graph defining a Person. Relations among these packages are established using edges labeled with *parent* [Graves, 1993; Graves et al, 1994; Graves et al, 1995a; Graves et al, 1995b].



RDF. Schema and instance are mixed together. In the example, the edges labeled type disconnect the instance from the schema. The instance is built by the subgraphs obtained by instantiating the nodes of the schema, and establishing the corresponding parent edges between these subgraphs [Klyne & Carroll, 2004; Hayes & Gutierrez, 2004; Angles & Gutierrez, 2005].



Simatic-XT. This model does not define a schema. The relations Name-Lastname and Person-Parent are represented in two abstraction levels. In the first level (the most general), the graph contains the relations Name and Lastname to identify people (P1, ..., P6). In the second level we use the abstraction of Person, to compress the attributes Name and Lastname and represent only the relation Parent between people [Mainguenaud, 1992].

➢ *Advantages of Graph database models*

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].

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 especially in *RDF-databases* has reestablished the relevance of this area [Angles & Gutierrez, 2008].

2.7 RDF databases

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 design of a traditional database is guided by the discovery of regularity or uniformity. The principle of regularity is a standardization of design relying on an abstract view of the world, where exceptions to the rule are not taken into account, since they are considered as insignificant in the design of an advantageous structured schema. The popularity of relational database management systems (RDBMS) is due to their ability to support many data management problems dealt by applications. However, a priori uniformity required by relational model can lead to hardness when modeling a not static world such as Semantic Web data [Faye et al, 2012].

The primary goal of RDF is to handle ***non regular or semi-structured data***. The research community has early recognized that there is an increasing amount of data that is insufficiently structured to support traditional database techniques, but does contain a sufficiently regular structure exploitable in the formulation and execution of queries [Muys, 2007].

It is widely acknowledged that information access can benefit from the use of ontologies. For this purpose, available data has to be linked to concepts and relations in the corresponding ontology and access mechanisms have to be provided that support the integrated model consisting of ontology and data. The most common approach for linking data to ontologies is via RDF representation of available data that describes the data as instances of the corresponding ontology that is represented in terms of RDF Schema. Due to the practical relevance of data access based on RDF and RDF Schema, a lot of effort has been spent on the development of corresponding storage and retrieval infrastructures [Hertel et al, 2009].

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 RDF graph [RDF, 2013]. This can be illustrated by a node and directed-edge diagram, in which each triple is represented as a “node-edge-node” link (hence the term "graph") (Figure 13).

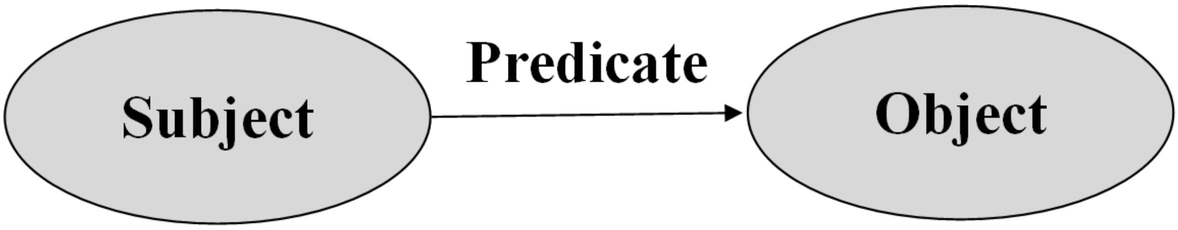


Figure 13. RDF triple

Each triple represents a statement of a relationship between the things denoted by the nodes that it links. It has three parts:

- Subject;
- A predicate (also called a property) that denotes a relationship;
- Object.

The direction of the edge (predicate) is significant: it always points toward the object. The nodes of RDF graph are its subjects and objects.

The assertion of 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 RDF graph amounts to asserting all the triples in it, so the meaning of 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].

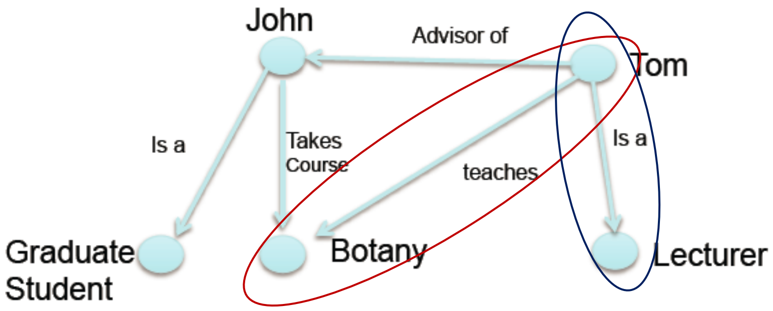
In other words, RDF provides a general method to decompose any information into pieces called triples [Briggs, 2012]:

- Each triple is of the form “Subject”, “Predicate”, “Object”;
- Subject and Object are the names for two things in the world. Predicate is the relationship between them;
- Subject, Predicate, Object are given as URI’s (stand-ins for things in the real world);
- Object can additionally be raw text.

In technical terms the RDF-triples’ set form labeled directed graph, where each edge is a triple, for instance, the triples:

Subject	**Predicate**	**Object**
<Tom>	<is a>	<Lecturer>
<Tom>	<teaches>	<Botany>

define some of the elements of the next graph [Briggs, 2012]:



The research community has early recognized the natural flexibility and expressivity of triples. Indeed, triples consider both objects and relationships as first-class citizens; thus, allowing on-the-fly generation of data. The power of RDF relies on the flexibility in representing arbitrary structure without a priori schemas. Each edge in the graph is a single fact, a single statement, similar to the relationship between a single cell in a relational table and its row’s primary key. RDF offers the ability to specify concepts and link them together into a graph of data [Faye et al, 2012].

➢ *RDF advantages*

As a storage language, RDF has several advantages [Owens, 2009]. First, it is possible to link different data sources together by adding a few additional triples specifying relationships between the concepts. This would be more difficult in the case of an RDBMS in which schema realignment or matching may be necessary. Then, RDF offers a great deal of flexibility due to the variety of the underlying graph-based model (i.e. almost any type of data can be expressed in this format with no needs for data to be present). There is no restriction on the graph size, as opposed to RDBMS field where schema must be concise. This a significant gains when the structure of the data is not well known in advance. Last, any kind of knowledge can be expressed in RDF, authorizing extraction and reuse of knowledge by various applications.

Consequently RDF offers a very useful data format, for which efficient management is needed. This becomes a hard issue for application dealing with RDF and known as ***RDF (or Triple) Stores***, due to the irregularity of the data. RDF Stores must allow the following fundamental operations on repository of RDF data: performing a query, updating, inserting (assertion), and deleting (retraction) triples [Owens, 2009]. In addition, there are issues that may require an extension of the triple-based schemas and thus are affecting the design of the database tables:

- Storing multiple ontologies in one database;
- Storing statements from multiple documents in one database.

Both points are concerning the aspect of provenance, which means keeping track of the source an RDF statement is coming from.

When storing multiple ontologies in one database it should be considered that classes, and consequently the corresponding tables, can have the same name. Therefore, either the tables have to be named with a prefix referring to the source ontology or this reference is stored in an additional attribute for every statement [Pan & Heflin, 2004].

A similar situation arises for storing multiple documents in one database. Especially, when there are contradicting statements it is important to know the source of each statement. Again, an additional attribute denoting the source document helps solving the problem [Pan & Heflin, 2004].

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

➢ *RDF disadvantages*

Different authors report different and specific RDF disadvantages. For instance, in [Costello & Jacobs, 2003] is noted that disadvantages of using the RDF format are:

- RDF uses namespaces to uniquely identify types (classes), properties, and resources. Thus, one must have a solid understanding of namespaces;
- Constrained: the RDF format constrains one on how he design his XML (i.e., one can't design his XML in any arbitrary fashion);
- Another XML vocabulary to learn: to use the RDF format one must learn the RDF vocabulary.

Other point of view we see in [Baidu, 2013]. RDF disadvantages are:

- Generic triple storage often (but not always) implies less efficient lookups (special indexes can still be built, but this moves away from schema flexibility);
- Certain data cannot easily be represented in RDF;
- Practical disadvantages with respect to (relatively) immature RDF storage systems and tools and porting over existing systems;
- High overhead for developers to get the necessary expertise to do a good job;

- Only non-standard solutions available for declaratively specifying (common types of CWA) constraints;
- The RDF triple is ontology based, always need the same schema;
- Not easy to do some complex reasoning;
- Low efficient to query data in the RDF triples, compared against RDBMS.

From our point of view, it is important to discuss the problem of ***numbering*** large RDF triple's elements (strings). Developers generally make special provisions for storing RDF resources efficiently. Indeed, rather than storing each Internationalized Resource Identifier (IRI) or literal value directly as a string, implementations usually associate a *unique numerical identifier* to each resource and store this identifier instead [Yongming et al, 2012].

There are two motivations for this strategy. First, since there is no a priori bound on the length of the IRIs or literal values that can occur in RDF graphs, it is necessary to support variable-length records when storing resources directly as strings. By storing the numerical identifiers instead, fixed-length records can be used. Second, and more importantly, RDF graphs typically contain very long IRI strings and literal values that, in addition, are frequently repeated in the same RDF graph.

Unique identifiers can be computed in two general ways [Yongming et al, 2012]:

- ***Hash-based approaches*** obtain a unique identifier by applying a hash function to the resource string, where the hash function used for IRIs may differ from the hash function used for literal values. Of course, care must be taken to deal with possible hash collisions. In the extreme, the system may reject addition of new RDF triples when a collision is detected. To translate hash values back into the corresponding IRI or literal value when answering queries, a distinguished dictionary table is constructed;
- ***Counter-based approaches*** obtain a unique identifier by simply maintaining a counter that is incremented whenever a new resource is added. To answer queries, dictionary tables that map from identifiers to resources and vice versa are constructed. Typically, these dictionary tables are stored as B-Trees for efficient retrieval. A variant on this technique that is applicable when the RDF graph is static is to first sort the resource strings in lexicographic order, and to assign the identifier n to the nth resource in this order. In such a case, a single dictionary table suffices to perform the mapping from identifiers to resources and vice versa [Yongming et al, 2012].

Various optimizations can be devised to further improve storage space. For example, when literal values are small enough to serve directly as unique identifiers (e.g., literal integer values), there is no need to assign unique identifiers, provided that the storage medium can distinguish between the system-generated identifiers and the small literal values. Also, it is frequent that many IRIs in an RDF graph share the same namespace prefix. By separately encoding this namespace prefix, one can further reduce the storage requirements [Yongming et al, 2012].

In other words, ***the bottleneck problem for RDF is numbering of very great amount of strings from RDF triples, sometimes up to several billion instances***.

For goal of this research we chose the second approach for solving the problem, i.e. to use counters. The new idea is that the process of numbering does not use B-Trees or any variant of traditional hashing. We use NL-addressing to assign numbers and co-ordinate access to restore string which corresponds to given number. The algorithm will be presented in Chapter 6. It has constant complexity which is important for very large datasets.

➢ *Storage and retrieval technologies for RDF*

The state of the art with respect to existing storage and retrieval technologies for RDF data is given in [Hertel et al, 2009] as well as in [Faye et al, 2012]. Different repositories are imaginable, e.g. main memory, files or databases.

RDF schemas and instances can be efficiently accessed and manipulated in main memory. Storing everything *in-memory* cannot be a serious method for storing extremely large volumes of data. However, they can act as useful benchmark and can be used for performing certain operations like caching data from remote sites or for performing inference. Most of the in-memory stores have efficient reasoners available and can help solve the problem of performing inference in persistent RDF stores, which otherwise can be very difficult to perform [CTS, 2012].

For persistent storage, the data can be serialized to files, but for large amounts of data the use of database management system is more reasonable. Examining currently existing RDF stores we found that they have used 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:

- Generic schemas, i.e. schemas that do not depend on the ontology and run on third party databases (Jena SDB which can be coupled with almost all relational databases like MySQL, PostsgreSQL, and Oracle);
- Ontology specific schemas, for instance, the native triple stores which provide persistent storage with their own implementation of the databases (Virtuoso, Mulgara, AllegroGraph, and Garlik JXT).

Main characteristics of several known RDF triple stores and our experimental program RDFArM are presented in Table 77 of the Appendix B.

In the following we will discuss the NL-Addressing (Natural Language Addressing) as an approach to be used for organizing middle-size or large RDF triple or quadruple stores or other kind of graph data bases.

➢ *Storing ontology generic schemas*

✓ *Vertical representation*

The simplest RDF generic schema is triple store with only one table required in the database.

The table contains three columns named *Subject*, *Predicate* and *Object*, thus reflecting the triple nature of RDF statements. Indexes are added for each of the columns in order to make joins less expensive. This corresponds to the *vertical representation* for storing objects in a table [Agrawa et al, 2001].

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

In other words, since the collections of triples are stored in one single RDF table, *the queries may be very slow to execute*. Indeed, when the number of triples scales, the RDF table may exceed main memory size. Additionally, simple statement-based queries can be satisfactorily processed by such systems, although they do not represent the most important way of querying RDF data. Nevertheless, RDF triples store scales poorly because complex queries with multiple triple patterns require many self-joins over this single large table as pointed out in [Faye et al, 2012].

The triple table approach has been used by systems like Oracle [oracledb, 2012; Chong et al, 2005], 3store [Harris & Gibbins, 2003], Redland [Beckett, 2001], RDFStore [RDFStore, 2012] and rdfDB [Guha, 2013].

✓ *Normalized triple store (vertical partitioning)*

The triple store can be used in its pure form [Oldakowski et al, 2005], but most existing systems add several modifications to improve performance or maintainability. A common approach, the so-called *normalized triple store*, is adding two further tables to store resource URIs and literals separately as shown in Figure 14, which requires significantly less storage space [Harris & Gibbins, 2003]. Furthermore, a hybrid of the simple and the normalized triple store can be used, allowing storing the values themselves either in the triple table or in the resources table [Jena2, 2012].

Triples:

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

Resources:

ID	URI
r1	*...#1*
r2	*...#2*
…	…

Literals:

ID	Value
l1	*Value1*
…	…
…	…

Figure 14. Normalized triple store

In a further refinement, the Triples table can be *split horizontally* into several tables, each modeling an RDF(S) property. These tables need only two columns for *Subject* and *Object*. The table names implicitly contain the predicates. This schema separates the ontology schema from its instances, explicitly models class and property hierarchies and distinguishes between class-valued and literal-valued properties [Broekstra, 2005; Gabel et al, 2004].

To realize the vertical partitioning approach, the tables have to be stored by using a *column-oriented DBMS* (i.e., a DBMS designed especially for the vertically partitioned case, as opposed to a row oriented DBMS, gaining benefits of compressibility and performance), as collections of columns rather than collections of rows. The goal is to avoid reading entire row into memory from disk, like in row-oriented databases, if only a few attributes are accessed per query. Consequently, in column oriented databases only those columns relevant to a query will be read. The approach creates materialized views for frequent joins. Furthermore, the object columns of tables in their scheme can also be optionally indexed (e.g., using an unclustered B+ tree), or a second copy of the table can be created clustered on the object column. One of the primary benefits of vertical partitioning is the support for rapid subject joins. This benefit is achieved by sorting the tables via subject. The tables being sorted by subject, one has a way to use fast merge joins to reconstruct information about multiple properties for subsets of subjects.

Index-all approach is a poor way to simulate a column-store. The vertical partitioning approach offers a support for multi-valued attributes. Indeed, if a subject has more than one object value for a given property, each distinct value is listed in a successive row in the table for that property. For a given query, only the properties involved in that query need to be read and no clustering algorithm is needed to divide the triples table into two-column tables.

Inserts can be slow in vertically partitioned tables since multiple tables need to be accessed for statement about the same subject. With a larger number of properties, the triple store solution manages to outperform the vertically partitioned approach [Faye et al, 2012].

➢ *Storing ontology specific schemas*

✓ *Horizontal representation*

Ontology specific schemas are changing when the ontology changes, i.e. when classes or properties are added or removed. The basic schema consists of one table with one column for the instance identificator (ID), one for the class name and one for each property in the ontology. Thus, one row in the table corresponds to one instance. This schema is corresponding to the *horizontal representation* [Agrawal et al, 2001] and obviously has several drawbacks:

- Large number of columns;
- High sparsity;
- Inability to handle multi-valued properties;
- The need to add columns to the table when adding new properties to the ontology,

etc.

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

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

✓ *Decomposition storage model*

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

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

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

In practice, a hybrid schema is used to benefit from advantages of combining both the table-per-class and table-per property schemas. This schema contains one table for each class, only storing there a unique ID for the specific instance. This replaces the modeling of the rdf:type property. For all other properties tables are created as described in the table-per-property approach (Figure 15) [Pan & Heflin, 2004]. Thus, changes to the ontology do not require changing existing tables, as adding a new class or property results in creating a new table in the database.

ClassA:

ID
…#1
…

Property1:

Subject	Object
…#1	…#3
…	…

ClassB:

ID
…#3
…

Figure 15. RDF Hybrid schema (the table-per-property approach)

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

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

To reduce the number of tables, single-valued properties with a literal as range can be stored in the class tables [Wilkinson, 2006; Broekstra et al, 2002]. Adding new attributes would then require changing existing tables. Another variation is to store all class instances in one table called Instances. This is especially useful for ontologies where there are many classes with only few or no instances [Alexaki et al, 2001; Wilkinson, 2006; Inseok et al, 2005].

The property table technique has the drawback of generating many NULL values since, for a given cluster, not all properties will be defined for all subjects. This is due to the fact that RDF data may not be very structured. A second disadvantage of property table is that multi-valued attributes, that are furthermore frequent in RDF data, are hard to express. In a data model without a fixed schema like RDF, it's common to seek for all defined properties of a given subject, which, in the property table approach, requires scanning all tables.

In this approach, including new properties requires also adding new tables; which is clearly a limitation for applications dealing with arbitrary RDF content. Thus schema flexibility is lost and this approach limits the benefits of using RDF. Moreover, queries with triples patterns that involve multiple property tables are still expensive because they may require many union clauses and joins to combine data from several tables. This consequently complicates query translation and plan generation. In summary, property tables are rarely used due to their complexity and inability to handle multi-valued attributes [Faye et al, 2012].

This approach has been used by tools like Sesame [Sesame, 2012; Broekstra et al, 2002], Jena2 [Jena2, 2012; Wilkinson et al, 2003], RDFSuite [Alexaki et al, 2001] and 4store [Harris et al, 2009].

✓ *Multiple indexing frameworks*

The idea of multi-indexing is based on the fact that queries bound on property value are not necessarily the most interesting or popular type of queries encountered in real world Semantic Web applications.

Due to the triple nature of RDF data, the goal is to handle equally the following type of queries:

- Triples having the same subject;
- Triples having the same property;
- List of subjects or properties related to a given object.

For achieving this goal, these approaches maintain a set of six indices covering all possible access schemes an RDF query may require. These indexes are PSO, POS, SPO, SOP, OPS, and OSP (P stands for property, O for object and S for subject). These indices materialize all possible orders of precedence of the three RDF elements. At first sight, such a multiple-indexing would result into a combinatorial explosion for an ordinary relational table. Nevertheless, it is quite practical in the case of RDF data [Weiss et al, 2008; RDF, 2013]. The approach does not treat property attributes specially, but pays equal attention to all RDF items [Faye et al, 2012].

This approach has been used by tools like Kowari system [Wood et al, 2005], Virtuoso [Erling & Mikhailov, 2007], RDF-3X [Neumann & Weikum, 2008], Hexastore [Weiss et al, 2008], RDFCube [Matono et al, 2007], BitMat [Atre et al, 2009], BRAHMS [Janik & Kochut, 2005], RDFJoin [McGlothlin & Khan, 2009], RDFKB [McGlothlin & Khan, 2009a], TripleT [Fletcher & Beck, 2009], iStore [Tran et al, 2009], Parliament [Kolas et al, 2009].

✓ *Storing models for popular ontologies*

Storing models for nine popular linguistic, conceptual or mixed ontologies are outlined in Table 4. These models are similar and practically are based on the well-known file systems or relational databases (RDBMS). In the case of RDBMS, orientation is mainly toward SPARQL. The ontologies are described by high-level languages (e.g. KIF, CycL, SubL, RDF, XML), which can be interpreted and/or stored in relational structures (e.g. MySQL), ER-model (e.g. FreeBase) and others.

In general, the systems for storing ontologies and, in particular, RDF data are based on (see also [Magkanaraki et al, 2002]):

- Structures in memory (e.g. TRIPLE [Sintek & Decker, 2001]);
- Popular relational databases (e.g. ICS-FORTH RDF Suite [Alexaki et al, 2001; 2001a], Semantics Platform 2.0 of Intellidimension Inc. [ISP2.0, 2012], Ontopia Knowledge Suite [Ontopia, 2012]);
- Non-relational file systems, indexed by key B-trees using systems such as Oracle Berkeley DB (e.g. rdfDB [Dumbill, 2000], RDF Store [RDFStore, 2012], Redland [Beckett, 2001], Jena [McBride, 2001]).

Table 4. Methods for storing data in nine ontologies

	ontology name	developer	quantity of terms	storing models	integration with other ontologies
1	**WordNet** [Fellbaum et al, 1998; Miller, 1995]	Princeton University	about 100 000	files	SUMO, FrameNet
2	**Sensus** [ISI, 2012]	ISI USC	more than 70 000	files, relational databases	subset of the WordNet
3	**Omega** [Philpot et al, 2005]	ISI USC	about 120 000	relational databases (MySQL)	WordNet, Mikrokosmos
4	**Mikrokosmos** [Beale et al, 1996; Mikr, 2012]	CLR UNMS	more than 7 000	relational database	WordNet, Omega
5	**OpenCyc** [OpenCyc, 2012]	Cycorp	more than 100 000	files (CycL, SubL, RDF)	WordNet
6	**DOLCE** [Masolo et al, 2003]	LAO ICST	about 4 000	files (KIF)	No
7	**PropBank** [Giuglea & Moschitti, 2004; Kingsbury & Palmer, 2003]	University PennState	more than 4 300	frame files	FrameNet, VerbNet

8	**FrameNet**
	[Fillmore, 1976;
	Baker et al, 1998;
	FrameNet, 2012]
		ISI, Berkeley, CA	about 900 frames	files (XML)	WordNet,
					PropBank,
					SUMO

9	**SUMO** [SUMO, 2012]	Teknowledge Corporation, SUO WG	more than 1 000	SUO-KIF files	FrameNet, WordNet, EMELD

2.8 Multi-layer representation of graphs

➢ *Names and locations 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 16)

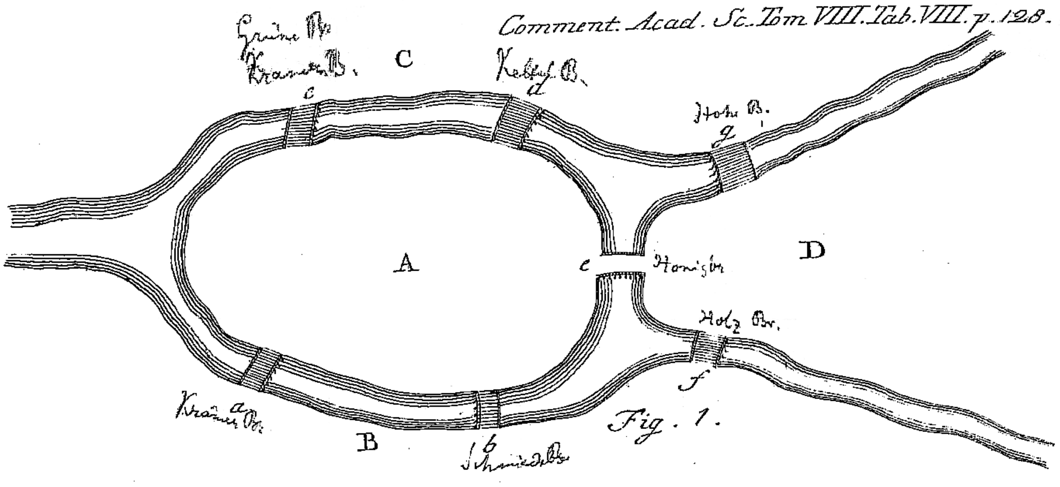


Figure 16. Illustration of Königsberg bridge problem [Euler, 1736]

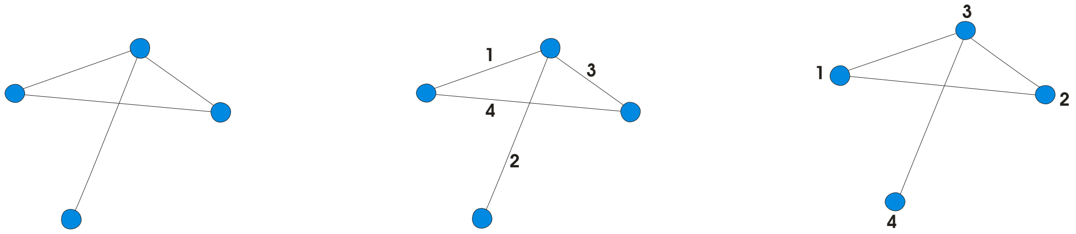
What is interesting in this schema is that every bridge has two kinds of identification - every bridge has:

- 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.), Hohe Br. (High Br.);
- Its own location, given by another names (addresses): a, b, c, d, e, f, g.

In this case the names are unique and different one from other. Because of this, the names of locations may seem redundant. In the general case the names may coincide, but the addresses of locations must be unique.

This way we come to the idea 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 17).



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

Figure 17. Labeled graphs

If set of names is a multi-set [Knuth, 1998], i.e. if it may contain more than one instance of the same name, than mapping the names to locations is not one-one. To make it one-one, additional qualifiers are used like "building" and "street" in two definitions below:

- **"Queen Victoria Building"**: It is a late nineteenth-century building designed by the architect George McRae in the central business district of Sydney, Australia;
- **"Queen Victoria Street"**: It is a street named after the British monarch who reigned from 1837 to 1901, located in the city of London which runs east by north from its junction with New Bridge Street and Victoria Embankment in Castle Baynard ward, along a section that divides the wards of Queenhithe and Bread Street, then lastly through the middle of Cordwainer ward, until it reaches Mansion House Street at Bank junction.

Another approach is numbering like the street numbers of houses.

At the end, the case with multi-sets of names may be resolved by the analyzing current context, for instance:

- **"King Street"**: It is a major east-west commercial thoroughfare in Toronto, Ontario, Canada. It was named after King George III, the reigning British monarch at the time the street was being built in early Toronto (then called the Town of York);
- **"King Street"**: It is a cross street in the Central Business District of Sydney, New South Wales, Australia. It stretches from King Street Wharf and Lime Street near Darling Harbour in the west, to Queens Square at St. James railway station in the east.

Depending where we are or what we talk about (Canada or Australia), saying "King Street" we will understand one or another meaning (definition) without collisions.

If set of names does not contain more than one instance of the same name, than set of locations 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 name of location (numbers, symbols, letters) are preferable. If the interpreter is an end-user (human), the natural language names (words or phrases) are preferable.

Now, one may put a question: “Is it possible, taking in account the context, one and the same string of letters to be used as both name and point to a location (address) of the definition or other information connected to it?” The positive answer is given by NL-addressing presented in this monograph.

To illustrate this, a simple example is presented in the next section.

➢ *Multi-layer representation of graphs*

Consider a sample graph (Figure 18) [GraphDB, 2012], which contains three nodes named (Alice, Bob, Chess) with identificators (Id.1, Id.2 and Id.3), six labeled edges connecting them (knows, knows, is_member, members, members, is_member) with identificators (Id.100,..., Id.105), and some additional features and their values.

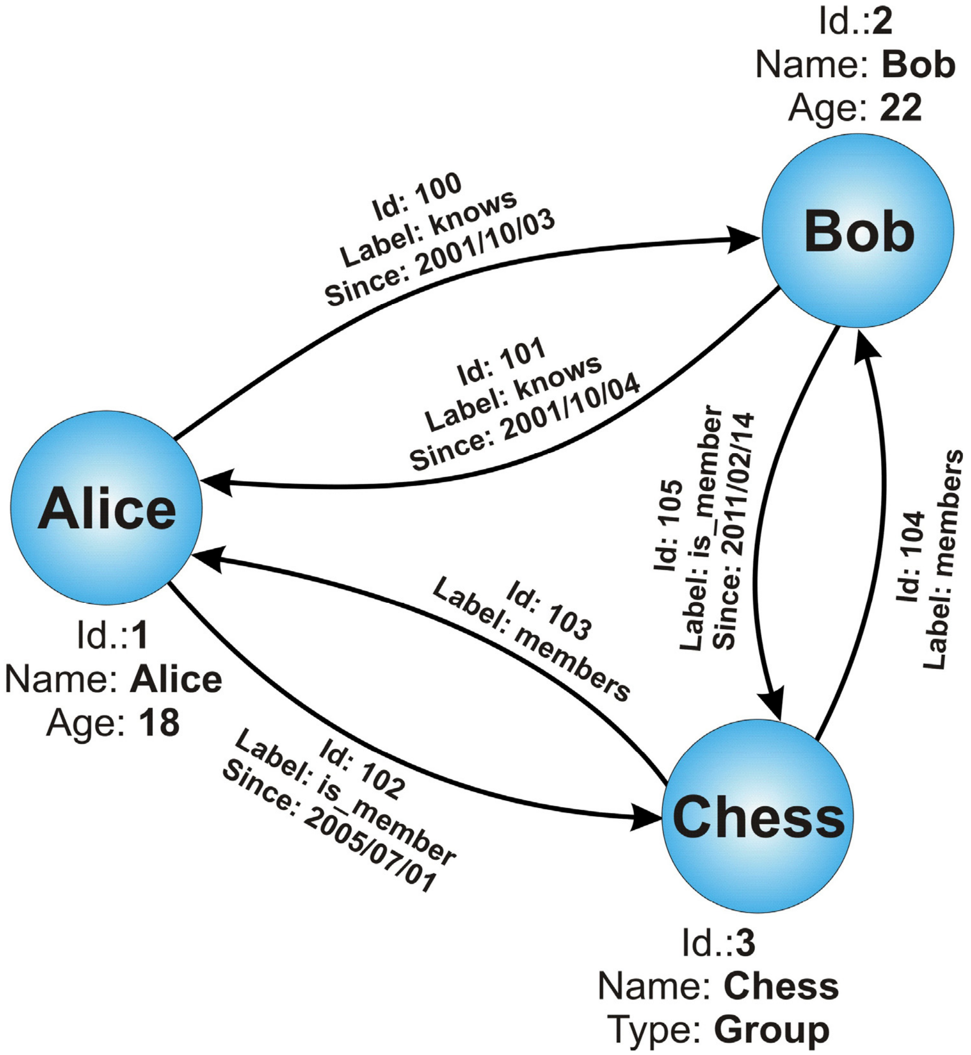


Figure 18. A sample graph

The set of nodes does not contain repeated members but the set of edges is a multi-set – every member is repeated two times and we need additional identification to separate different edges.

It is possible to present the information about graph in two tables – one for nodes (Table 5) and another for the edges (Table 6) [Ivanova et al, 2013b].

The names of columns and values of the corresponded cells are easy understandable. In this case we may use relational database and the additional information of locations will be the identification couples (key, column). It is well-known that in the relational databases there exits special algorithm for computing real offset in the plain file of records using the definition of the table where:

- Columns are clearly described by their names, width, type, etc.;
- Rows are identified by keys.

For Table 5 keys may be the names of the nodes or identifiers because these sets do not contain repeated elements. In this case the identifiers may be avoided.

For Table 6 the identifiers could not be avoided because the set of edges' names is multi-set, i.e. it contains repeated elements (every element is included two times). In this case, as keys we may use identifiers as single keys or couple (identifier, edge name) as complex key.

Table 5. Description of nodes of the sample graph and index

index			information about nodes					
key	offset		Node Id.	Name	from-edges	to-edges	Age	Type
1	0	→	1	Alice	101; 103	100; 102	18	-
2	100	→	2	Bob	100; 104	101; 105	22	-
3	200	→	3	Chess	102; 105	103; 104	-	Group

Table 6. Description of edges of the sample graph and index

index			information about edges				
key	offset		Edge Id.	Label	from node	to node	Since
100	0	→	100	knows	1	2	2001/10/03
101	50	→	101	knows	2	1	2001/10/04
102	100	→	102	is_member	1	3	2005/07/01
103	150	→	103	members	3	1	-
104	200	→	104	members	3	2	-
105	250	→	105	is_member	2	3	2011/02/14

For both tables to reach needed information we may scan sequentially records of the file till find the right key.

Usually in relation databases, an index of keys and their offsets (locations of the rows' records) in the file is built and search is provided firstly in the index and taking the value of the offset (location) the row information is accessed directly in the main file.

For easy reading we assume that rows of the nodes' table are 100 bytes long, and rows of the edges' table are 50 bytes long.

Indexes may be of different types – B-trees, hash tables, etc. In all cases, additional resources are needed (memory to be stored and time to be processed). Note that the values of the keys are duplicated in the indexes.

At the end, the identifiers play important role for easy representing inter-relations of the graph. Because of this they could not be avoided.

If we will take in account the interrelations between nodes and edges, which define "context" of the graph, we will see that another (“***multi-layer***”) representation is possible and the identifiers of nodes and edges can be avoided (Table 7 and Table 8).

Table 7 contains a multi-layer representation of the sample graph with nodes as columns and edges as layers. This is not exactly relational table because the layers may be stored in different files. If we will use the possibility for NL-addressing, the names of the columns will define locations in files of layers.

Table 7. Multi-layer representation of the sample graph with nodes as locations.

		locations (nodes)		
		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**	

To find all edges from given node we have to take node name (column) as NL-address, for instance "Bob", and read all information from different layers (rows) stored at location determined by "Bob" as NL-address.

If the information about edges is more important than we may transpose the matrix from Table 7. This will give us a multi-layer representation of the sample graph with edges as locations (Table 8). Here, the same considerations as for the Table 7 may be done.

Table 8. Multi-layer representation of the sample graph with edges as locations.

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

What is important is that NL-addressing reduces the information to be stored on the disc - only the cells with text in bold of Table 7 or Table 8 will be stored. At a rough estimate we have:

- In the "relational" case (Table 5 + Table 6): at least 18 + 30 = **48** cells for the two tables and real need of additional indexing to speed up the access. Note that empty cells in relational tables really take place on the disk (they contain space symbols);
- In the "multi-layer" case (Table 7 or Table 8): only **8** filled cells and no need of additional indexing.

In both cases the same information is stored, but in the second case the filled cells contain larger information which may be accessed by single operation and this way may speed the access.

2.9 Multi-domain information model (MDIM)

Below we will use strong hierarchies of named sets to create a specialized mathematical model for new kind of organization of information bases. The "information spaces" defined in the model are kind of strong hierarchies of enumerations (named sets).

The independence of dimensionality limitations is very important for developing new software systems aimed to process large volumes of high-dimensional data. To achieve this, we need information models and corresponding access methods to cross the boundary of the dimensional limitations and to obtain the possibility to work with large information spaces with variable and practically unlimited number of dimensions. A step in developing such methods is the **Multi-domain Information Model (MDIM)** introduced in [Markov, 1984; Markov, 2004].

2.9.1 Basic structures of MDIM

Main structures of MDIM are *basic information elements, information spaces, indexes* and *meta-indexes,* and *aggregates*. The definitions of these structures are given below:

➢ *Basic information elements*

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

➢ *Information spaces*

Let the universal set ***UBIE*** be the set of all ***BIE***.

Let $\boldsymbol{E}_1$ be a set of basic information elements. Let $\boldsymbol{\mu}_1$ be a function, which defines a biunique correspondence between elements of the set $\boldsymbol{E}_1$ $\mathbf{E_1}$ and elements of the set $\boldsymbol{C}_1$ $\boldsymbol{C_1}$of positive integer numbers, i.e.:

$$\boldsymbol{E}_1 = \{e_i \mid e_i \in \boldsymbol{UBIE}\ , i=1,\dots, m_1\}.$$

$$\boldsymbol{C}_1 = \{c_1 \mid c_i \in \mathrm{N}, i=1,\dots,m_1\}$$

$$\boldsymbol{\mu}_1\ E_1 \leftrightarrow C_1$$

The elements of $\boldsymbol{C}_1$ are said to be numbers (co-ordinates) of the elements of E_1.

The triple $\boldsymbol{S}_1 = (\boldsymbol{E}_1, \boldsymbol{\mu}_1, \boldsymbol{C}_1)$ is said to be a ***numbered information space of level 1*** (one-dimensional or one-domain information space).

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

$$\boldsymbol{E}_2 = \{e_i \mid e_i \in \boldsymbol{NIS_1}, i=1,\ldots, m_2\}.$$

$$C_2 = \{c_i \mid c_i \in N, i=1,\ldots,m_2\}$$

$$\mu_2 : E_2 \leftrightarrow C_2$$

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

$$\boldsymbol{E}_n = \{e_j \mid e_j \in \boldsymbol{NIS_{n-1}}, j=1,\ldots, m_n\}.$$

$$C_n = \{c_j \mid c_j \in N, j=1,\ldots,m_n\}$$

$$\mu_n : E_n \leftrightarrow C_n$$

Every basic information element "e" is considered as an ***information space*** $\boldsymbol{S}_0$ of level 0. It is clear that the information space $\boldsymbol{S}_0 = (\boldsymbol{E}_0, \boldsymbol{\mu}_0, \boldsymbol{C}_0)$ is constructed in the same manner as all others:

- The machine codes (bytes) b_i, i=1,…,m_0 are considered as elements of $\boldsymbol{E}_0$;
- The position p_i (natural number) of b_i in the string e is considered as co-ordinate of b_i, i.e.

$$C_0 = \{p_k \mid p_k \in N, k=1,\ldots,m_0\} ,$$

- Function μ_0 is defined by the physical order of b_i in e and we have $\mu_0 : E_0 \leftrightarrow C_0$.

This way, the string $\boldsymbol{S}_0$ may be considered as a set of ***sub-elements (sub-strings).*** The number and length of the sub-elements may be variable. This option is very helpful but it closely depends on the concrete realizations and it is not considered as a standard characteristic of MDIM.

The information space $\boldsymbol{S}_n$, which contains all information spaces of a given application is called ***information base*** of level **n**. The concept information base without indication of the level is used as generalized concept to denote all available information spaces. For instance every relation data base may be represented as an ***information base of level 3*** which contains set of two dimensional tables.

➢ *Indexes and meta-indexes*

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

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

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

➢ *Aggregates*

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

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

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

It is clear, we can build **m** aggregates using the set G because every information space $S_J \in G, j=1, \ldots, n$, may be chosen to be a main information space.

2.9.2 Operations in the MDIM

After defining the information structures, we need to present the operations, which are admissible in the model.

In MDIM, we assume that **all** information elements of **all** information spaces **exist**.

If for any $S_i : E_i = \emptyset \wedge C_i = \emptyset$, than it is called ***empty***.

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

➢ *Operations with basic information elements*

Because of the rule that all structures exist, we need only two operations with a BIE:

- Updating;
- Getting the value.

For both operations, we need two service operations:

- Getting the length of a BIE;
- Positioning in a BIE.

Updating, or simply – ***writing*** the element, has several modifications with obvious meaning:

- Writing as a whole;
- Appending/inserting;
- Cutting/replacing a part;
- Deleting.

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

➢ *Operations with spaces*

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

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

- Copy/move with clearing the recipient space;
- Copy/move with merging the spaces.

The first modifications first clear the recipient space and after that provide a copy or move operation.

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

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

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

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

➢ *Operations with indexes, meta-indexes and aggregates*

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

Every information space is built by two sets: the set of co-ordinates and the set of information elements. Because of this, the operations with indexes, meta-indexes, and aggregates may be classified in two main types:

- Operations based only on co-ordinates, regardless of the content of the structures;
- Operations, which take in account the content of the structures.

The operations based only on the co-ordinates are aimed to support information processing of analytically given information structures. For instance, such structure is the table, which may be represented by an aggregate. Aggregates may be assumed as an extension of the relations in the sense of the model of Codd [Codd, 1970]. The relation may be represented by an aggregate if the aggregation mapping is one-one mapping. Therefore, the aggregate is a more universal structure than the relation and the operations with aggregates include those of relation theory. What is the new is that the mappings of aggregates may be not one-one mappings.

In the second case, the existence and the content of non empty structures determine the operations, which can be grouped corresponding to the main information structures: elements, spaces, indexes, and meta-indexes. For instance, such operation is the **projection**, which is the analytically given space index of non-empty structures. The projection is given when some coordinates (in arbitrary positions) are fixed and the other coordinates vary for all possible values of coordinates, where non-empty elements exist. Some given values of coordinates may be omitted during processing.

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

2.10 Multi-domain access method “ArM32”

For practical implementation aimed to store very large perfect hash tables and burst tries *in the external memory* (hard disks) we need realization in accordance to the real possibilities. The existing models, analyzed in this research, do not support such structures. Because of this, we decide to make experiments with “Multi-Domain Information Model” (MDIM) [Markov, 1984] and corresponded to it software tools. We will use MDIM as a model for database organization and corresponded specialized tools we will upgrade to our case.

During the last three decades, MDIM has been discussed in many publications. See for instance [Markov et al, 1990; Markov, 2004; Markov et al, 2013].

The program realizations of MDIM are called Multi-Domain Access Method (MDAM) or Archive Manager (ArM) (Table 9).

Table 9. Realizations of MDAM:

no.	name	year	machine	type	language and	operating system
0	**MDAM0**	1975	MINSK 32	37 bit	Assembler	Tape OS
1	**MDAM1**	1981	IBM 360	32 bit	FORTRAN	DOS 360
2	**MDAM2**	1983	PDP 11	16 bit	FORTRAN	DOS 11
3	**MDAM3**	1985	PDP 11	16 bit	Assembler	DOS 11
4	**MDAM4**	1985	Apple II	8 bit	UCSD Pascal	Disquette OS
5	**MDAM5**	1986	IBM PC	16 bit	Assembler, C	MS DOS
6	**MDAM6**	1988	SUN	32 bit	C	SUN UNIX
7	**ArM7**	1993	IBM PC	16 bit	Assembler	MS DOS 3
8	**ArM8**	1998	IBM PC	16 bit	Object Pascal	MS Windows 16 bit
9	**ArM32**	2003	IBM PC	32 bit	Object Pascal	MS Windows 32 bit
10	**NL-ArM**	2012	IBM PC	32 bit	Object Pascal	MS Windows 32 bit
11	**BigArM**	2015 ... under developing		64 bit	Pascal, C, Java	MS Windows, Linux, Cloud

All projects of MDAM and ArM had been done by Krassimir Markov. The first program realizations had been done by Krassimir Markov (MDAM0, MDAM1, MDAM2, MDAM3);

The next program realizations had been done by Krassimir Markov and:

- Dimitar Guelev (MDAM4);
- Todor Todorov (MDAM5 written on Assembler with interfaces to PASCAL and C, MDAM5 rewritten on C for IBM PC);
- Vasil Nikolov (MDAM5 interface for LISP, MDAM6);
- Vassil Vassilev (ArM7 and ArM8);
- Ilia Mitov and Krassimira Minkova Ivanova (ArM 32);
- Vitalii Velychko (ArM32 interface to Java);
- Krassimira Borislavova Ivanova (NL-ArM).

For a long period, MDIM has been used as a basis for organization of various information bases.

One of the first goals of the development of MDIM was representing the digitalized military defense situation, which is characterized by a variety of complex objects and events, which occur in the space and time and have a long period of variable existence [Markov, 1984]. The great number of layers, aspects, and interconnections of the real situation may be represented only by information spaces' hierarchy. In addition, the different types of users with individual access rights and needs insist on the realization of a special tool for organizing such information base.

Over the years, the efficiency of MDIM is proved in wide areas of information service of enterprise managements and accounting. For instance, the using MDIM permits omitting the heavy work of creating of OLAP structures [Markov, 2005].

In this research we will use the Archive Manager – "ArM32" developed for MS Windows (32 bit) [Markov, 2004; Markov et al, 2008] and its upgrade to NL-ArM.

The ArM32 elements are organized in numbered information spaces with variable levels. There is no limit for the levels of the spaces. Every element may be accessed by a corresponding multidimensional space address (coordinates) given via coordinate array of type cardinal. At the first place of this array, the space level needs to be given. Therefore, we have two main constructs of the physical organizations of ArM32 information bases – numbered information spaces and elements.

The ArM32 Information space (IS) is realized as a (*perfect*) *hash table stored in the external memory.* Every IS has 2^{32} entries (elements) numbered from 0 up to 2^{32}-1. The number of the entry (element) is called its *co-ordinate*, i.e. the co-ordinate is a 32 bit integer value and it is the number of the entry (element) in the IS.

Every entry is connected to a container with variable length from zero up to 1G bytes. If the container holds zero bytes it is called "empty". In other words, in ArM32, the length of the element (string) in the container may vary from 0 up to 1G bytes. There is no limit for the number of containers in an archive but their total length plus internal indexes could not exceed 2^{32} bytes in a single file.

If all containers of an IS hold other IS, it is called "*IS of corresponded level*" depending of the depth of including subordinated IS. If containers of given IS hold arbitrary information but not other IS, it is called "*Terminal IS*".

To locate a container, one has to define ***the path in hierarchy*** using a ***co-ordinate array*** with all numbers of containers starting from the one of the *root* information space up to the *terminal* information space which is owner of the container.

The hierarchy of information spaces may be not balanced. In other words, it is possible to have branches of the hierarchy which have different depth.

*In ArM32, we assume that **all possible** information spaces **exist**.*

If all containers of the information space are empty, it is called "***empty***".

Usually, most of the ArM32 information spaces and containers are empty. "Empty" means that corresponded structure (space or container) does not occupy disk space. This is very important for practical realizations.

Remembering that **Trie** is a tree for storing strings in which there is one node for every common prefix and the strings are stored in extra leaf nodes, we may say the ArM32 has analogous organization and *can be used to store (burst) tries*.

➢ *Functions of ArM32*

ArM32 is realized as set of functions which may be executed from any user program. Because of the rule that all structures of MDIM exist, we need only two main functions with containers (elements):

- Get the value of a container (as whole or partially);
- Update a container (with several variations).

Because of this, the main ArM32 functions with information elements are:

- *Arm Read* (reading a part or a whole element);
- *Arm Write* (writing a part or a whole element);
- *Arm Append* (appending a string to an element);
- *Arm Insert* (inserting a string into an element);
- *Arm Cut* (removing a part of an element);
- *Arm Replace* (replacing a part of an element);
- *Arm Delete* (deleting an element);
- *Arm Length* (returns the length of the element in bytes).

MDIM operations with information spaces are over:

- **Single space** – *clearing the space*, i.e. updating all its containers to be empty;
- **Two spaces** – there exist several such type of operations. The most used is copying of one space in another, i.e. copying the contents of containers of the first space in the containers of the second. Moving and comparing operations are available, too.

The corresponded ArM32 functions over the spaces are:

- *ArmDelSpace* (deleting the space);

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

The ArM32 functions, aimed to serve the navigation in the information spaces return the space address of the **next** or **previous, empty** or **non-empty** elements of the space starting from any given co-ordinates. They are *ArmNextPresent, ArmPrevPresent, ArmNextEmpty*, and *ArmPrevEmpty*.

The ArM32 function, which create indexes, is *ArmSpaceIndex* – returns the space index of the non-empty structures in the given information space.

The service function for counting non-empty ArM32 elements or subspaces is *ArmSpaceCount* – returns the number of the non-empty structures in given information space.

Using ArM32 engine practically we have great limit for the number of dimensions as well as for the number of elements on given dimension. The boundary of this limit in the current realization of ArM32 engine is 2^{32} for every dimension as well as for number of dimensions. Of course, another limitation is the maximum length of the files, which depends on the possibilities of the operating systems and realization of ArM. For instance, in the next version, ArM64 called “BigArM”, these limits will be extended to cover the power of 64 bit addressing.

ArM32 engine supports multithreaded concurrent access to the information base in real time. Very important characteristic of ArM32 is possibility not to occupy disk space for empty structures (elements or spaces). Really, only non-empty structures need to be saved on external memory.

Summarizing, the advantages of the ArM32 are:

- Possibility to build growing space hierarchies of information elements;
- Great power for building interconnections between information elements stored in the information base;
- Practically unlimited number of dimensions (this is the main advantage of the numbered information spaces for well-structured tasks, where it is possible "***to address, not to search***").

➢ *Conclusion of the Chapter 2*

This chapter introduced the main data structures and storing technologies which further we will use to compare our results. Mainly they are graph data models as well as RDF storage and retrieval technologies.

Firstly we defined concepts of storage model and data model.

Mapping of the data models to storage models is based on program tools called “access methods”. Their main characteristics were outlined.

Graph models and databases were discussed more deeply and examples of different graph database models were presented. The need to manage information with graph-like nature especially in RDF-databases had reestablished the relevance of this area.

There is a real need of efficient tools for storing and querying knowledge using the ontologies and the related resources. In this context, the annotation of unstructured data has become a *necessity in order to increase the efficiency of query processing. Efficient data storage and query processing that can scale to large amounts of possibly schema-less data has become an important research topic. The proposed approaches usually rely on (object-) relational database technology or on main-memory virtual machine implementations, while employing a variety of storage schemes [Faye et al, 2012].*

In accordance with this, the analyses of RDF databases as well as of the storage and retrieval technologies for RDF structures were in the center of our attention. The analysis of the viewed tools showed that all of them use data storing models which are limited to text files, indexed data or relational databases. These approaches do not conform to the specific structures of the ontologies. This necessitates the development of new models and tools for storing ontologies which correspond to their structure.

Storing models for several popular ontologies and summary of main types of storing models for ontologies and, in particular, RDF data were discussed.

At the end of this chapter, our attention was paid to addressing and naming (labeling) in graphs with regards to introducing the Natural Language Addressing (NL-addressing) in graphs. A sample graph was analyzed to find its proper representation.

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.

Concluding, let us point on advantages and disadvantages of the multi-layer representation of graphs.

The main disadvantages are:

- *The layers are sparsed;*
- *The number of locations may be very great which causes the need of corresponded number of columns in the table (in any cases hundred or thousand).*

The main advantages are:

- *Reducing the used resources;*
- *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 and avoid using of supporting indexes for information retrieval services (B-trees, hash tables, etc.);*
- *Finally, using NL-addressing, the multi-layer representation is easily understandable by humans and interpretable by the computers.*

If we will use indexed files or relational data bases, the disadvantages are so serious that make the implementation impossible.

We propose to use the multi-dimensional model for organization of information. For this purpose the "Multi-Domain Infrmation Model"and its realizations were presented. The Multi-Dimensional Numbered Information Spaces are basis for context independed indexing. Because of this they may be used for storing Big Data.

3 Access method based on NL-addressing

Abstract

This chapter is aimed to introduce a new access method based on the idea of NL-addressing.

For practical implementation of NLA we need a proper model for database organization and corresponded specialized tools. Hash tables and tries give very good starting point. The main problem is that they are designed as structures in the main memory which has limited size, especially in small desktop and laptop computers. Because of this we need analogous disk oriented database organization.

To achieve such possibilities, we decided to use "Multi-Domain Information Model" (MDIM) and corresponded to it software tools. MDIM and its realizations are not ready to support NL-addressing. We will upgrade them for ensuring the features of NL-addressing via new access method called NL-ArM.

The program realization of NL-ArM, based on specialized hash functions and two main functions for supporting the NL-addressing, access will be outlined. In addition, several operations aimed to serve the work with thesauruses and ontologies as well as work with graphs, will be presented.

3.1 Example of NL-addressing via burst tries

Analyzing Figure 8 and Figure 9, one may see a common structure in both figures. It is *a trie which leafs are containers*. In Figure 8 leafs are Social Security Numbers (SS#) and in Figure 9 leafs are Binary Search Trees (BST). In addition, Figure 5 looks as it is created from many connected Perfect Hash Tables (PHT).

An inference from this is the idea about a multi-way burst trie which:

- Nodes are PHT with entries for all letters of alphabet plus some additional symbols, for instance "0" and "1" ("true" and "false");
- Containers may hold subordinated burst tries.

To illustrate this as a way for possible realization of NL-addressing using burst tries, let see an example (Figure 19).

In this example we use natural numbers instead of letters i.e. their machine codes. This way our alphabet will consists of:

- 256 natural numbers if we will use ASCII encoding;
- 2^{16} natural numbers for UNICODE 16 encoding;
- 2^{32} natural numbers for UNICODE 32 encoding.

We assume that the computer word is 32 bits long and our numbering will permit 2^{32} numbers. What encoding will be used depends of concrete requirements. For easy reading, here we will consider ASCII encoding. This way we will use a small part of all possibilities of such construction.

As we pointed above, we extend the idea of burst tries by creating a hierarchy. In this case every container of a burst trie may hold:

- A string of letters, i.e. a word or phrase of words (such container is colored in magenta on Figure 19 and has number 114);
- Subordinated burst trie.

The path to the container colored in magenta on Figure 19 is A = (66, 101, 101, 114).

It means that:

- Container 66 of root burst trie holds non empty burst trie in which:
    - Container 101 holds non empty burst trie in which:
        - Container 101 holds non empty burst trie in which:
            - Container 114 is holds any information (string).

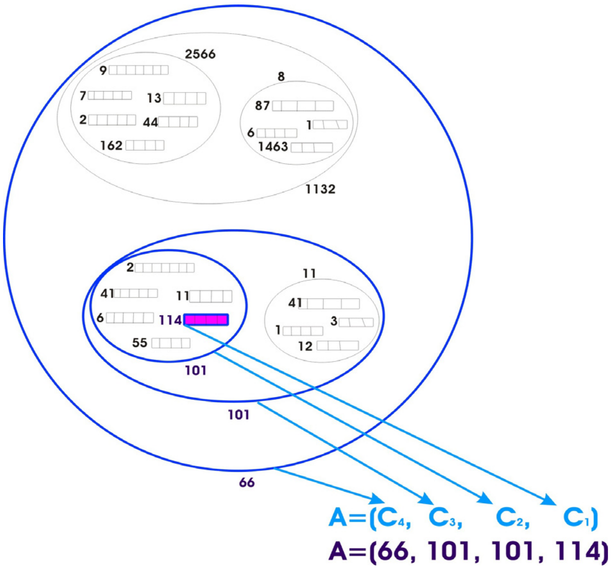


Figure 19. Example of location A=(66,101,101,114)

The numbering is unique for every burst trie. Because of this, there is no problem to have the same numbers of the containers in the subordinated burst tries what is illustrated at the Figure 19 – container 101 holds burst trie with container 101.

Consider the path we just have seen. If we assume these numbers as ASCII codes:

66 = "B", 101 = "e", 114 = "r",

we may "*understand*" the path as the word "***Beer***" (Figure 20).

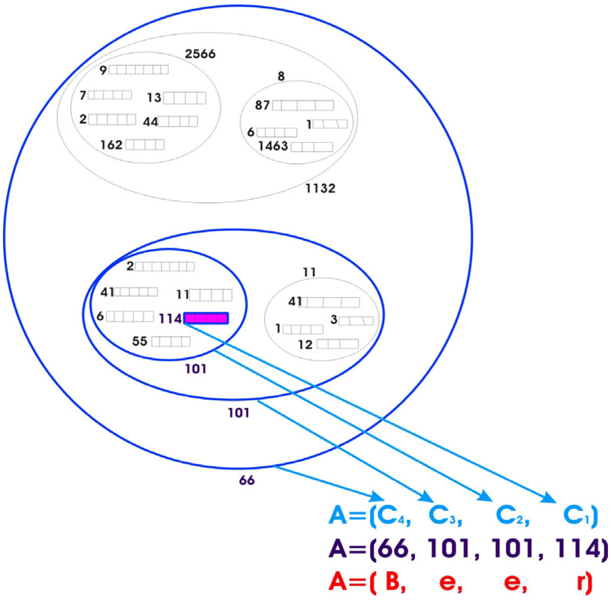


Figure 20. Example of natural language path A=(Beer)

At the end, the container, colored in magenta at Figure 21 and located by this path, may hold arbitrary long string of letters (words, phrases). In our example we choose it to be the remarkable aphorism of Benjamin Franklin:

"Beer is proof that God loves us and wants us to be happy".

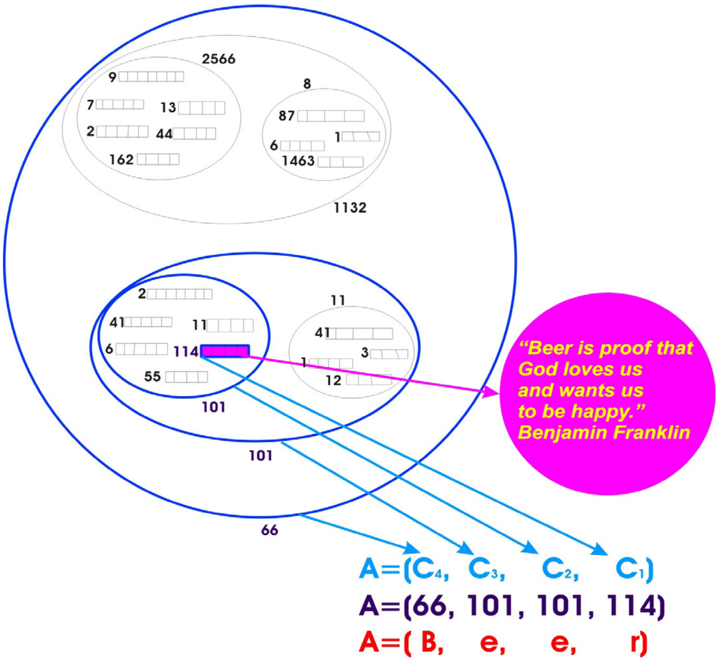


Figure 21. Example of content located by path "Beer"

Perfect hash tables and burst tries give very good starting point. The main problem is that they are designed as *structures in the main memory* which has limited size, especially in small desktop and laptop computers.

3.2 NL-ArM access method

MDAM and respectively ArM32 are not ready to support NL-addressing. We have to upgrade them for ensuring the features of NL-addressing. The new access method is called **NL-ArM** (Natural Language Addressing Archive Manager).

The program realization of NL-ArM is based on a specialized hash function and two main functions for supporting the NL-addressing access.

In addition, several operations were realized to serve the work with thesauruses and ontologies as well as work with graphs.

➢ *NL-ArM hash function*

The NL-ArM hash function is called "*NLArmStr2Addr*". It converts a string to space path. Its algorithm is simple: four ASCII symbols or two UNICODE 16 symbols form one 32 bit co-ordinate word. This reduces the space' level four, respectively – two, times. The string is extended with leading zeroes if it is needed. UNICODE 32 does not need converting – one such symbol is one co-ordinate word.

There exists a reverse function, "*NLArmAddr2Str*". It converts space address in ASCII or UNICODE string. The leading zeroes are not included in the string.

The functions for converting are not needed for the end-user because they are used by the NL-ArM upper level operations given below.

All NL-ArM operations access the information by NL-addresses (given by a NL-words or phrases). Because of this we will not point specially this feature.

Let's illustrate the algorithm of NL-ArM hash function.

In the case of Figure 21, the couple

{(name), (content)}

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 NL-path (B, e, e, r) to path of numbers (66, 101, 101, 114), 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).

NL-addressing means that human or program will set the correspondence:

(Beer) => ("Beer is proof that God loves us and wants us to be happy" Benjamin Franklin)

and all rest work has be done by the ***NL-ArM hash function*** which has to *convert name in a space path*, i.e.

Beer => (B, e, e, r) => (66, 101, 101, 114).

This hash function is one-one, and because of this, the resulting hash table is a perfect one.

Note that the NL-ArM is an access method and it receives commands only from other program units. Some of them are aimed to serve the human-computer interface and may redirect users' requests directly to NL-ArM (such units are experimental modules presented in this monograph). Other units may concern some information processing like reasoning and may send to NL-ArM requests according theirs need.

In the last case, the programs set the correspondence (name) => (content).

➢ *NL-ArM operations with terminal containers*

Terminal containers are those which belong to terminal information spaces. They hold strings up to 1GB long.

There are two main operations with strings of terminal containers:

- *NLArmRead* – read from a container (all string or substring);
- *NLArmWrite* – update the container (all string or substring).

Additional operations are:

- *NLArmAppend* (appending a substring to string of the container);
- *NLArmInsert* (inserting a substring into string of the container);
- *NLArmCut* (removing a substring from the string of the container);
- *NLArmReplace* (replacing a substring from the string of the container);
- *NLArmDelete* (empting the container);
- *NLArmLength* (returns the length of the string in the container in bytes).

In general, the container may be assumed not only as up to 1GB long string of characters but as some other information again up to 1GB. As a rule, the access methods do not interpret the information which is transferred to and from the main memory. It is important to have possibility to access information in the container as a whole or as set of concatenated parts.

Assuming that all containers exist but some of them are empty, we need only two main operations:

1) To update (write) the string or some of its parts.
2) To receive (read) the string or some of its parts.

The additional operations are modifications of the classical operations with strings applied to this case.

To access information from given container, NL-ArM needs the path to this container and buffer from or to which the whole or a part of its content will be transferred. Additional parameters are length in bytes and possibly - the starting position of substring into the string. When string has to be transferred as a whole, the parameters are the length of the string and zero as number of the starting position.

➢ *NL-ArM operations with information spaces (hash tables)*

With information spaces we may provide service operations with hash tables such as counting empty or non-empty containers, copying or moving strings of substrings from containers one to those of another terminal information space. We will not use these operations in the frame of this work.

3.3 Example of NL-storing the Sample graph

As final example of this chapter, let see how the sample graph from previous chapter can be stored using NL-addressing.

To make sample graph more "realistic", we will put a question about representing the characteristics of the nodes and edges. At the Figure 18 characteristics have been written as comments to nodes and edges.

In the graph, 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.

The final multi-layer representation of our sample graph is given in Table 10 and the final version of the sample graph is shown at Figure 22.

Table 10. Final multi-layer representation of sample graph

	space path		
human location	*Alice*	*Bob*	*Chess*
NL-ArM location	(65, 108, 105, 99, 101)	(66, 111, 98)	(67, 104, 101, 115, 115)
layer (file name)			
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**	

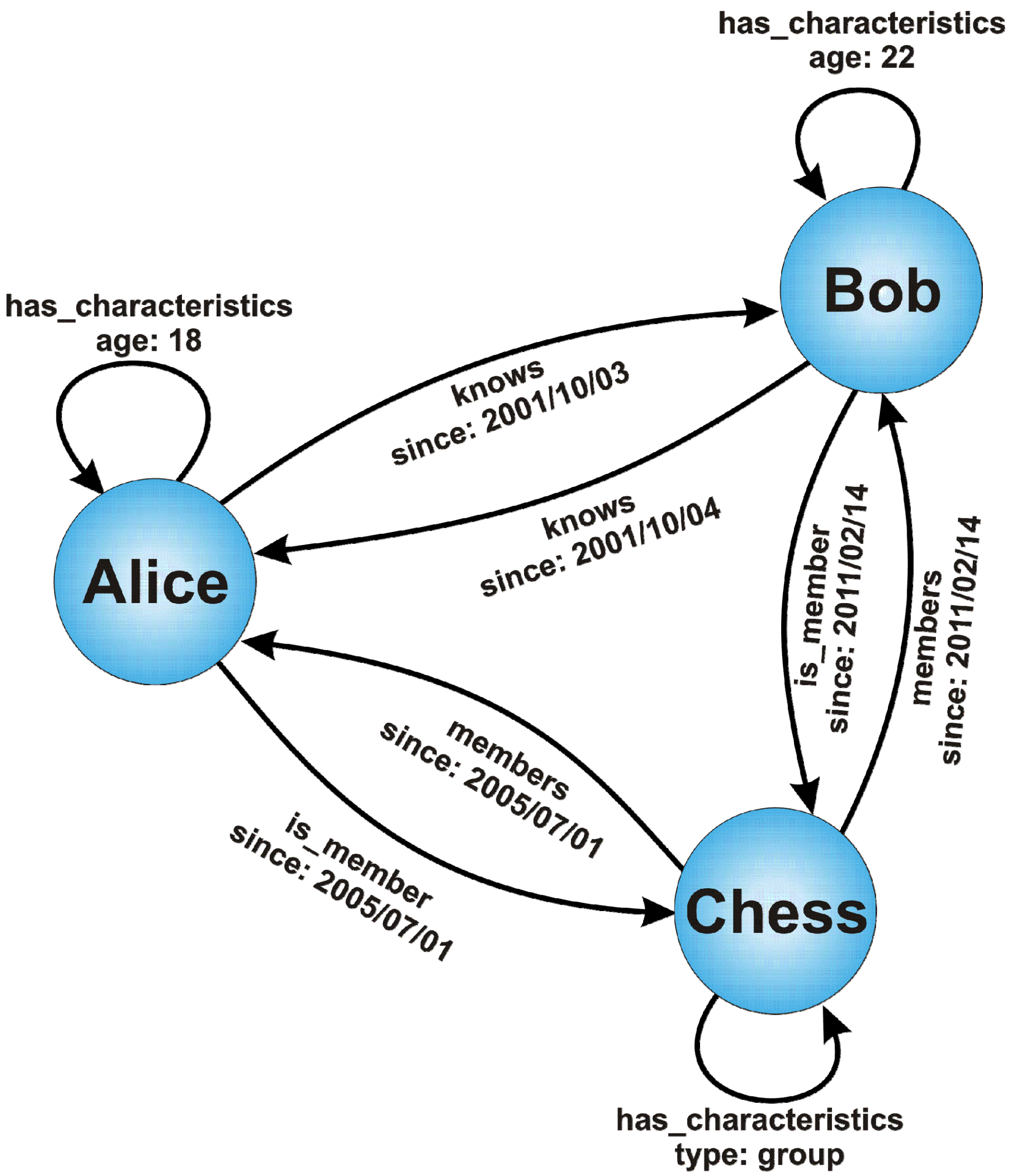


Figure 22. Final variant of the sample graph

Really, the Table 10 show what we will have in our sample database:

- Every layer (row of Table 10) is separate trie and will be stored in a separate file with name of the layer;
- Human locations are given by names: Alice, Bob, and Chess, and NL-ArM (internal computer) locations age given by paths: (65, 108, 105, 99, 101), (66, 111, 98), (67, 104, 101, 115, 115);
- All cells of Table 10 written in bold are containers which hold corresponded information (strings) from the cells;
- The locations (space paths) are common for all layers.

➢ *Storing RDF-graphs by NL-ArM*

We may easy represent the Table 10 by RDF-triples and vice versa (Table 11).

Table 11. Representation of the sample graph by RDF-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**

In other words, NL-ArM is ready for storing RDF information. Mapping of Table 11 in Table 10 is just the algorithm for creating RDF-triple stores based on MDIM and NL-addressing.

From Table 11 it follows that we may define two main information models for storing RDF-graphs using NL-ArM.

The first model we will denote as

RSO model, i.e. Relation-Subject-Object model,

and the second one as

SRO model, i.e. Subject-Relation-Object model.

The first information model for storing RDF-graphs is based on choosing the *relations as separate layers* (file names) and subjects as NL-paths in all layers, i.e.

RSO model: **Relation (Subject) => Object**.

The second is the dual one – the *subjects may be chosen as layers* and the relations as NL-path, i.e.

SRO model: **Subject (Relation) => Object**.

In both cases, the object is the only information to be stored in the archives.

The abstract structure of both models is:

NL-ArM_archive_file_name (NL-address) => Stored_information

What model has to be preferred depends of the sets of relations and subjects, i.e. one that has less size is preferable to be selected as set of layers. If both of models have great size than the next Universal model may be preferred.

In the Universal information model (**UNL model**), both Subject and Relation are equally presented. In this model concatenation of Subject and Relation is assumed as NL-path in a common archive (trie), i.e.

UNL model: **NL-ArM_archive (Subject, Relation) => Object**.

➢ *Conclusion of chapter 3*

This chapter was aimed to introduce a new access method based on the idea of Natural Language Addressing.

MDIM and its realizations are not ready to support NL-addressing. We upgraded them for ensuring the features of NL-addressing via new access method called ***NL-ArM****.*

The program realization of NL-ArM is based on specialized hash functions and two main functions for supporting the NL-addressing access.

In addition, several operations were realized to serve the work with thesauruses and ontologies as well as work with graphs.

NL-ArM is ready for storing RDF information. It is possible to define tree information models for storing RDF-graphs using NL-ArM: (1) ***RSO model*** *(Relation-Subject-Object model), (2)* ***SRO model*** *(Subject-Relation-Object model), and (3)* ***UNL model ((Subject, Relation) => Object*** *Universal model).*

4 Basic experiments

Abstract

In this chapter we will present two types "clear" experiments: with a text file and a relational database. The reason is that they are wide used for storing semi-structured data.

4.1 Comparison with a text file

The need to compare NL-ArM access method with text files is determined by practical considerations – in many applications the text files are main approach for storing semi-structured data. Text files are kind of files of records which are still in the basis of modern database management systems. Because of this, it is important to determine whether there exists a benefit of structuring data in tries. For instance, if the records are connected to keywords (strings), the information may be recorded by burst tries (i.e. by NL-addressing) rather than as usually with a clear indication of the keywords in the records.

Important consideration in this case is that the sequential reading text file (to find concrete keyword) is very slow operation and every indexed approach will be quicker. Indexed text files are typical for relational data bases and this case will be analyzed further in this chapter.

Here, the goal is to investigate the *size* of the files and *speed* of their generation. In other words, we have to compare writing in a sequential text file with writing in NL-ArM archive.

To compare so different file structures we need to use a common record structure. The structure of text file is a record identified by a keyword, i.e.

<keyword><text information><CR>.

In ArM32 and, respectively, in NL-ArM archives, the information is structured in multi-way multi-layer hash structures and the content of record may be accessed by keyword as path to location of container where the text is stored, i.e.

<path> ⇒ <text>.

In this case the keyword (path) is not stored on the disk.

In both cases the <text> is the same.

We have to make choice for length of the keywords. It may be arbitrary but with fixed length.

Our choice for keywords' length is based on understanding that *the average word length in English is approximately* ***5.10 characters*** [Sigurd et al, 2004]. In other words, the most frequently used English words are up to 8 characters long (Figure 23).

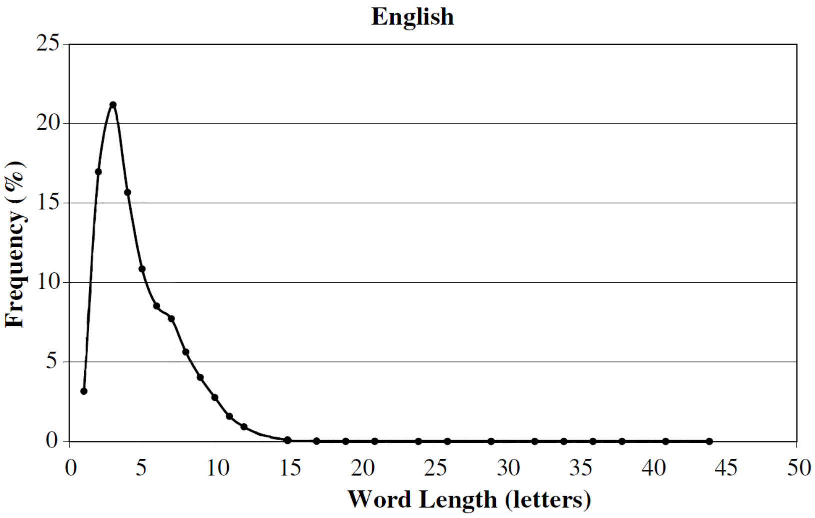


Figure 23. Word Length for English (extracted from [Sigurd et al, 2004])

The average word length in French is approximately 5.13 characters, in Spanish - 5.22 characters, in German - 6.26 characters [Sigurd et al, 2004], in Russian – 5.28 characters [Sharoff, 2001]. This means that length of 8 characters cover these languages, too.

Following this consideration, we chose 8 characters as length of the keywords in our experiments.

The basis of the experiments is a structure of 30 symbols:

- *Keyword* – artificial arbitrary string with maximal length of 8 ASCII symbols. Duplication of symbols is permitted;
- *String* of 22 arbitrary ASCII symbols,

which are stored (written) as follow:

- In the text file – as a structure consisting of keyword and string of length 30 bytes;
- In the NL-ArM archive - the same string (22 characters) stored in the elements with the specified by path of 8 characters (four ASCII symbols in one co-ordinate), i.e. in the archive will be written 22 bytes of the string and 8 symbols of the keyword will be assumed as path (NL-address).

For the experiments, the NL-ArM hash function was programmed to convert ASCII string to space path, i.e. four symbols to form one 32 bit co-ordinate word. The generated keyword string was extended with leading zeroes if it is needed. This way the 8 byte string keyword is converted in two 32 bit hash values and we have to use two layer hash structure. For instance, the string "ABCDEFGH" will be converted in two 32 bit words (ACBD) and (EFGH) where every letter's ASCII code occupies one byte.

The experiments follow the ones made in [Markov, 2006] and were provided on the same computer with a processor Celeron, 3.08 GHz, 512 MB RAM 160 GB HDD and operating system Microsoft Windows XP Professional Ver. 2002, Service Pack 2. The results of the experiments are given in Table 12, Table 13, Table 14, and Table 15. Corresponded graphic visualizations are given at Figure 24, Figure 25, Figure 26, and Figure 27. We received the same results as in [Markov, 2006] which means that the NL-addressing do not add significant complexity to one of the ArM32.

➢ *Comparison of time characteristics*

The Table 12 contains information about six experiments provided with different quantity of records from 2500 up to 100 millions. The table contains data in milliseconds (ms) about time for storing all data sets as well as the average time for storing of one record. The Figure 24 illustrates graphically the same data.

Table 12. Time (ms) for writing in text file and NL-ArM archive

writing in:	text file (ms)		in NL-ArM-archive (ms)	
number of records	all data	one record	all data	one record
2500	16	0.006400	47	0.018800
10000	62	0.006200	140	0.014000
250000	1625	0.006500	3328	0.013312
1000000	6703	0.006703	13359	0.013359
25000000	128719	0.005149	323407	0.012936
100000000	524281	0.005243	1314562	0.013146

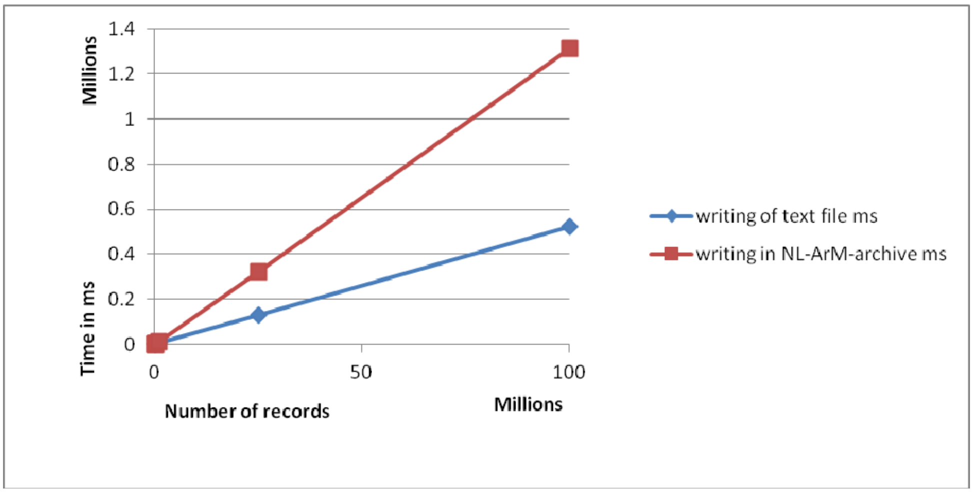


Figure 24. Time in milliseconds for writing in text file and NL-ArM archive

The conclusion is that the time of storing the information has expectable regularities: for great number of elements, writing in a NL-ArM archive became almost twice and half slower than writing in a text file. It is because of building the hash tables of the information in the NL-ArM archive.

Table 13 represents the time correlation between writing in text file and in NL-ArM archive. This correlation is illustrated on Figure 25.

Table 13. Time correlation for writing in text file and NL-ArM archive

number of records	text file	NL-archive
2500	1	2.94
10000	1	2.26
250000	1	2.05
1000000	1	1.99
25000000	1	2.51
100000000	1	2.51

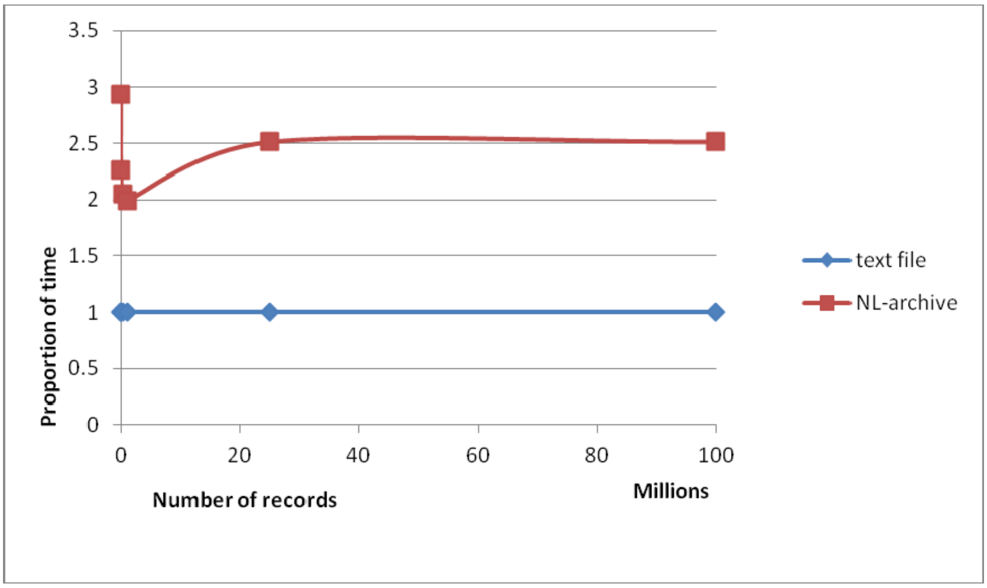


Figure 25. Time correlation between text file and NL-ArM for writing

It is important that the NL-ArM is *constantly* about two and half times slower. This means that the including new records in NL-ArM archive take the same time (about 0.013 ms) per record irrespective of the number of already stored records. For building very large data bases this is very crucial characteristic.

The speed of NL-ArM during storing the first 2500 – 5000 records is about three times slower. This is due to initial creating empty hash tables (information spaces) which takes additional time. This is illustrated on Figure 25 where in the beginning (most left part) of red curve there exists specific irregularity.

➢ *Comparison of size characteristics*

The comparison of file sizes shows that for great number of elements the text file became a little longer than NL-ArM archive. Adding indexes for speeding search in text file will increase the occupied memory because of:

- Duplicating the keywords in the index structures;
- Adding pointers to the records of the text file.

In the same time, after every writing operation, the NL-ArM archive is readies for immediate direct access (by arbitrary keywords) without need of additional indexing and duplication the keywords, which "annihilate" transmuting into paths (NL-addresses).

Table 14 contains data for sizes (in bytes) of the text file and the NL-ArM archive. The corresponded graphical visualization is shown on Figure 26.

The first column of Table 14 contains the number of records. The next two columns contain the size of the text file and the NL-ArM archive after storing the corresponded numbers of records, i.e. 2 500 records occupy 75 000 bytes in the text file and 130 048 bytes in NL-ArM archive.

It is important that in this case (8 bytes keywords) the NL-ArM ***takes about 5.5 bytes*** additional memory for every record to support hash tables' organization.

In other words, if we take the value of the size NL-ArM archive from the last row of the Table 14 (2 740 055 552 bytes) and subtract from it the real length of the stored 100 000 000 records of 22 bytes (2 200 000 000), we will receive the size of internal NL-ArM additional memory for hash indexes, which in this case is 540 055 552 bytes.

Now, dividing it on the number of records, i.e. 540055552/100000000 we receive the average of additional hash indexing memory for each record, i.e. 5.40055552 bytes.

Assuming this value as 5.5 bytes we may say that file with keywords longer than 6 bytes up to 8 bytes each will be stored by NL-ArM in a file with smallest size.

The same result is illustrated on the Figure 26 where the line of the size of the NL-ArM archive is under the line of the size of the corresponded text file (of records).

In the same time we receive one very important quality: *NL-ArM archive permits random direct access to all stored records immediately after writing it without any additional indexing.*

In experiments with text file we used artificially generated strings up to 8 symbols. If we use real English words, at the first glance, it will be more effective to use text file for storing couples (English word, definition), for instance, from a dictionary. If we want only to store the information, this conclusion is correct.

But if we want to use it via random read and/or update, we have to build indexes for quick access to the records which will duplicate the keywords and in addition will contain pointers to locations of keywords and definitions in the file. The external indexing structures used in modern databases need additional memory as well as time for realizing the same functionality. NL-ArM avoids such indexes.

Table 14. Size in bytes of the text file and the NL-ArM archive

number of records	text file	NL-ArM-archive
2 500	75 000	130 048
10 000	300 000	360 448
250 000	7 500 000	6 659 072
1 000 000	30 000 000	26 116 608
25 000 000	750 000 000	640 016 896
100 000 000	3 000 000 000	2 740 055 552

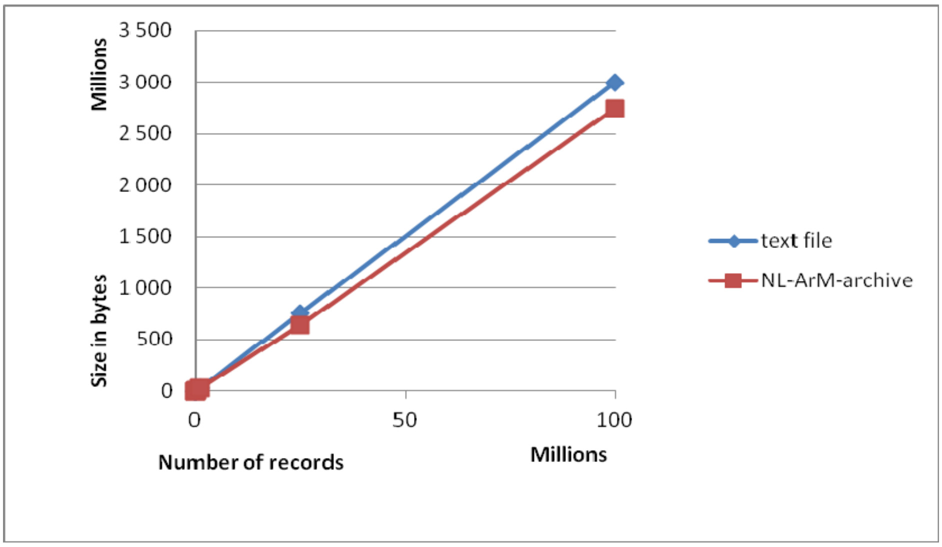


Figure 26. Size in bytes of the text file and the NL-ArM archive

Table 15 represents the relation between sizes of the text file and NL-ArM archive. This correlation is illustrated on Figure 27.

Table 15. Relation between sizes of the text file and NL-ArM archive

number of records	text file	NL-ArM archive
2 500	1	1.7
10 000	1	1.2
250 000	1.13	1
1 000 000	1.15	1
25 000 000	1.17	1
100 000 000	1.09	1

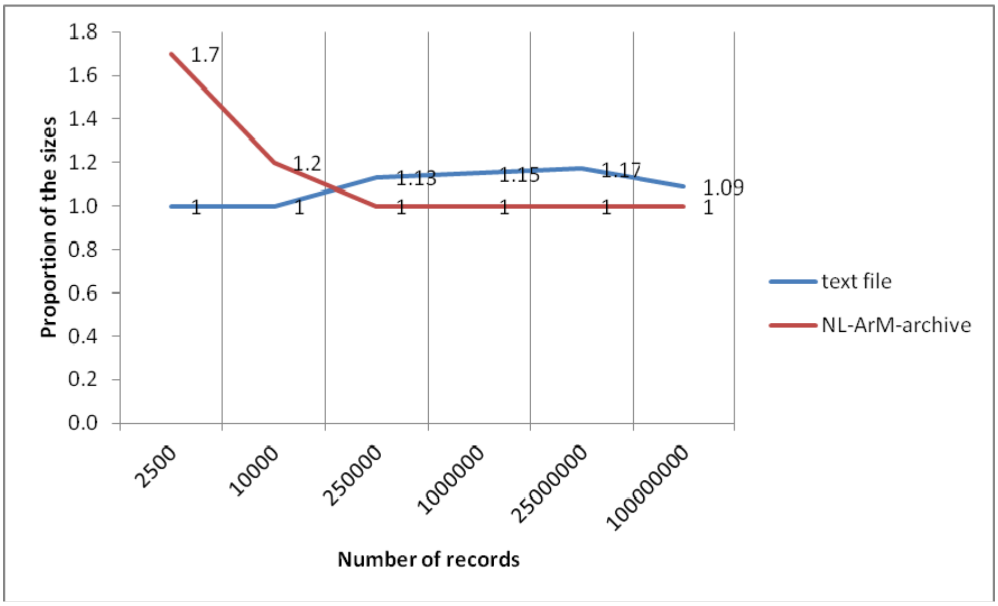


Figure 27. Relation between text file and NL-ArM for writing

The analysis of this relation indicates that, for 8 characters as length of the keywords and small quantity of records, the NL-ArM archive occupies more memory than text file but for the case of very large data the NL-ArM archive is smaller.

The explanation of this regularity is in the specific hash indexing in the NL-ArM archives. In the beginning, large empty hash structures are created which during the storing new records step by step are filled with internal pointers. This way, for great number of records, the hash indexing memory became about 5.5 bytes per record.

It is seen at the Figure 27. The graphics lines which represent sizes of text file and NL-ArM archive are crossed after 250 000 records. After 25 000 000 records the ratio comes to 1.09:1 which has to be examined in further experiments to find possible next cross point.

It is important to underline that these experiments were based on artificial data with fixed length (record of 30 bytes with 8 bytes artificially generated keyword of arbitrary ASCII symbols). If the length of the keywords is variable, the size of NL-ArM archive will be different according of length of the strings of keywords of stored information, i.e. according of number of layers of hash tables (depth of trie).

In sequential storing of records, NL-ArM access method is slower than same operation in text file. For applications where it is important in real time to register incoming information, the text files are preferable than archives with NL-Addressing.

We did not provide experiments to compare NL-access with searching and random reading/updating of records from text file because they are the slowest operations and every indexed approach will be quicker [Connolly & Begg, 2002]. Indexed files are typical for relational data bases and this case will be analyzed below.

4.2 Comparison with a relational database

To provide experiments with a relational database, we have chosen the system "Firebird" because:

- It is a relational database offering many ANSI SQL standard features that runs on Windows, Linux, and a variety of UNIX platforms;
- It offers excellent concurrency, high performance, and powerful query language;
- It has been used in production systems, under a variety of names, since 1981.

The Firebird Project is a commercially independent project of C and C++ programmers, technical advisors and supporters developing and enhancing a multi-platform relational database management system based on the source code released by Inprise Corp (known as Borland Software Corp, too) on 25 July, 2000 [Firebird, 2013].

Database management system "Firebird" is built on the code of Borland InterBase [InterBase, 2012]. It is a complete DBMS capable of managing databases in size from a few kilobytes to scores of gigabytes with excellent performance [Cantu, 2012].

Firebird supports all major operating systems including Windows, Linux, Solaris, MacOS, and there are multiple ways to access its database: native/API, dbExpress drivers, ODBC, OLEDB, Net provider, JDBC, Python module, PHP, Perl, and others. InterBase [InterBase, 2012] and Firebird [Borrie, 2004; ibphoenix, 2012] are widely distributed.

The experiments below were carried out with the version 2.0 of Firebird.

The experiments were provided in two steps: (1) *Writing* and (2) *Reading*.

The information was structured in the same manner as for experiments with sequential text files. The basis of the experiments is a table with two columns:

- *Keyword* - several (8 or 14) digital symbols;
- *String* - 22 arbitrary symbols,

which are stored (written) as follow:

- In the relational database – as a table structure consisting of two columns: keyword and string both encoded in ASCII;
- In the NL-ArM archive - the same string (22 characters) will be stored in the locations specified by keyword as path with two parts (digital ASCII symbols - 4+4 or 7+7, respectively).

For experiments with relational database NL-ArM hash function was programmed to convert ASCII digital strings of the two parts of keyword in two ***32 bit integer*** co-ordinate values, i.e. every part of keyword (digital string) is assumed as integer number and it is converted in a 32 bit integer value. In other words, string of 7 digit symbols is integer value which is less than 2^{32} and may be stored in 32 bits. This way we illustrate the possibility to have different hash functions for different specific cases of information.

To analyze performance of NL-ArM the keys were generated by special algorithm to test different variants of storing. For instance 10 000 rows may be stored via different combinations of the keys' two parts values:

- From 1 x 1 up to 10 x 1000;
- From 1 x 1 up to 100 x 100;
- From 1 x 1 up to 1000 x 10.

This way we have different "rectangular" varying the values of theirs vertex points. For Firebird keys were concatenated strings of the two parts:

- From 00010001 up to 00101000;
- From 00010001 up to 01000100;
- From 00010001 up to 10000010.

Special experiments were provided with "rectangular" which first point is shifted to point 1000000 x 1000000, i.e. for instance:

- From 1000000 x 1000000 up to 1000010 x 1010000;
- From 1000000 x 1000000 up to 1010000 x 1000010;
- From 1000000 x 1000000 up to 1001000 x 1001000.

For Firebird these keys looked as:

- From 10000001000000 up to 10000101010000;
- From 10000001000000 up to 10100001000010;
- From 10000001000000 up to 10010001001000.

➢ *Comparison of writing time characteristics*

In Table 16, several results from three variants of writing experiments are presented. The variants are based on a tables with two columns (key, string) and 10 000, 100 000, and 1 000 000 rows respectively.

The two columns of Table 16, marked as (X) and (Y), contain values of the two parts of the keywords which in the same time are coordinates of initial point of NL-ArM experimental rectangular. Its diagonal point is given by the corresponded offsets (ΔX) and (ΔY).

In the next column, the quantity of rows (ΔX∗ΔY) is given, respectively – 10 000, 100 000, and 1 000 000.

The writing time has been measured in milliseconds (ms) and the results are presented in the next two columns for Firebird and NL-ArM respectively.

In the last column, the ratio between Fireburd and NL-ArM for writing time is given. Graphical visualization of the ratio is on Figure 28.

Table 16. Writing time comparison of Firebird and NL-ArM

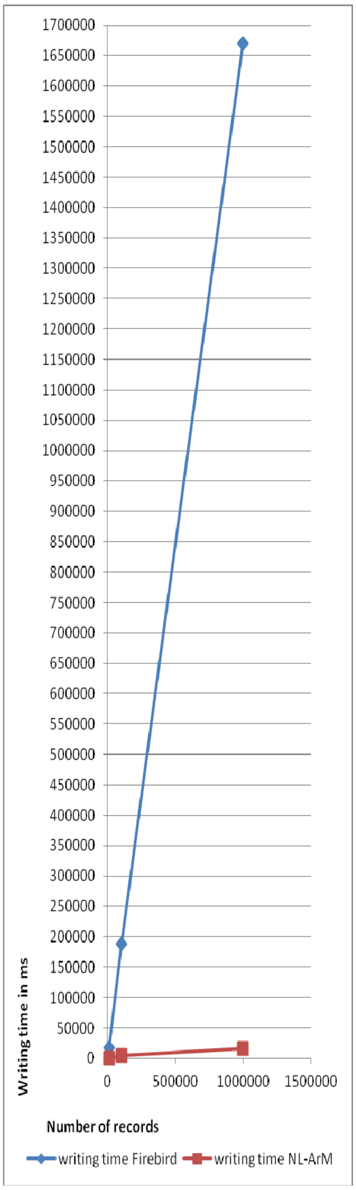
row No.:	initial values of co-ordinates		size of intervals of co-ordinates		Number of cells	writing time (ms)		ratio
						Firebird	NL-ArM	Firebird : NL-ArM
	(X)	(Y)	(ΔX)	(ΔY)	(ΔX*ΔY)	(ms)	(ms)	
1	1	1	10	1000	10000	21297	141	*151 : 1*
2	1	1	100	100	10000	14094	140	*100 : 1*
3	1	1	1000	10	10000	15438	156	*98 : 1*
4	1	1	10	10000	100000	160094	1563	*102 : 1*
5	1	1	100	1000	100000	145719	14062	*103 : 1*
6	1	1	1000	100	100000	141547	1265	*112 : 1*
7	1	1	10000	10	100000	155578	1719	*90 : 1*
8	1	1	1	100000	100000	292625	2907	*100 : 1*
9	1	1	100000	1	100000	289406	9390	*30 : 1*
10	1000000	1000000	10000	10	100000	156656	2109	*74 : 1*
11	1000000	1000000	10	10000	100000	162000	1672	*96 : 1*
12	1	1	10	100000	1000000	1740234	16640	*104 : 1*
13	1	1	100	10000	1000000	1591688	15187	*104 : 1*
14	1	1	1000	1000	1000000	1589734	14656	*108 : 1*
15	1	1	10000	100	1000000	1583906	13250	*119 : 1*
16	1	1	100000	10	1000000	1738047	17875	*97 : 1*
17	1000000	1000000	1000	1000	1000000	1778953	15750	*112 : 1*
	Total:				6830000	11577016	128482	*90.1 : 1*

The average of writing time data in milliseconds for the three groups (10 000, 100 000, and 1 000 000) are presented in Table 17.

In the last column, the average ratio of Firebird and NL-ArM is given. This relation is illustrated graphically on Figure 29.

Table 17. Average in milliseconds of writing time data

Number of cells	average writing time (ms) Firebird	 ArM	ratio Firebird : NL-ArM
10000	16943.000	145.670	116.330 : 1
100000	187953.125	4335.875	88.375 : 1
1000000	1670427.000	15559.670	107.330 : 1



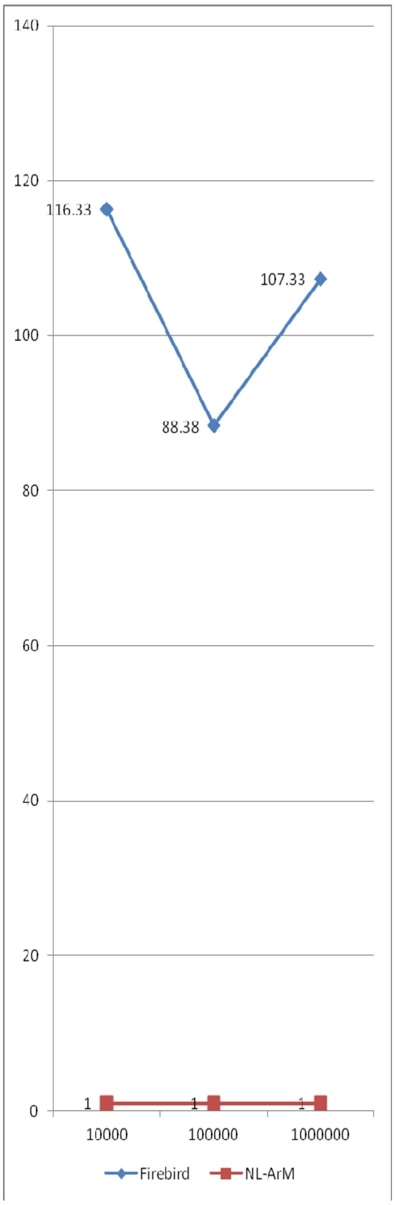


Figure 28. Time in miliseconds for writing by Firebird and NL-ArM

Figure 29. Time relation for writing by Firebird and NL-ArM

The results are expectable.

During initialization, Firebird take some additional time, than for rest records the consuming time has logarithmic regularity.

NL-ArM has no initialization procedures and has linear regularity for writing of all records (Figure 28). This relation may be seen at Figure 29, and more easily at Figure 30 where axes are logarithmic.

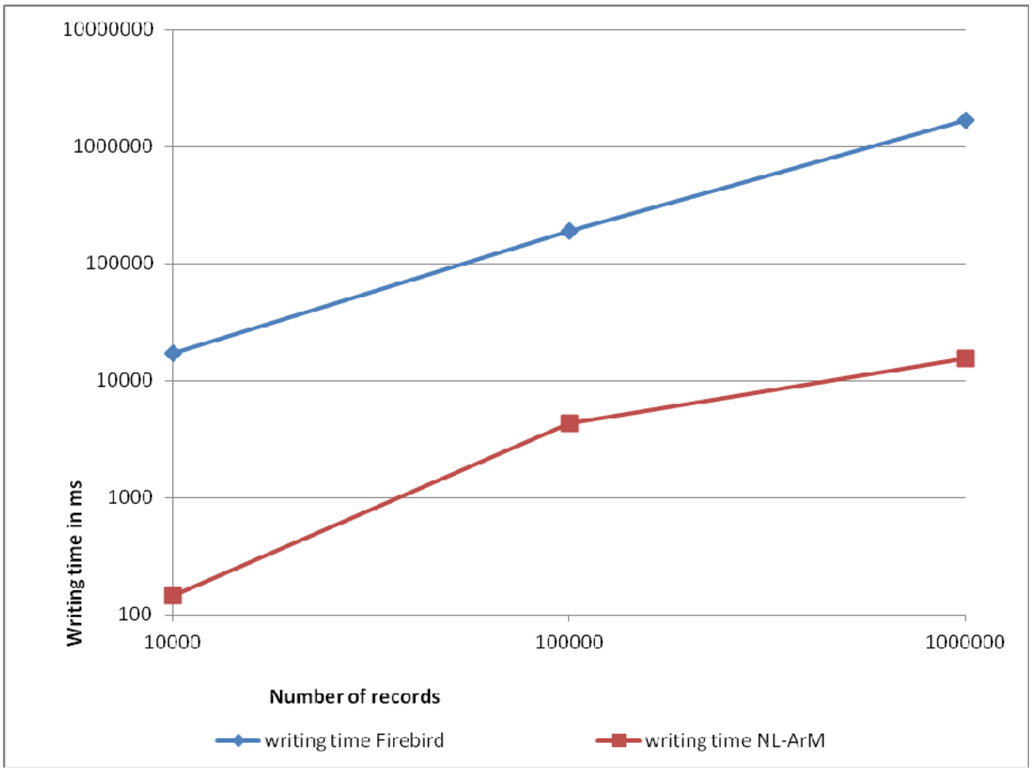


Figure 30. Logarithmic time relation for writing

In the same time we have to comment some disadvantages of NL-ArM in relation to Firebird. The keys used in the experiments were strings for the Firebird (relational) variants and two separate 32 bit (4-byte) co-ordinates for NL-ArM.

In relational model all keys have same influence on the writing time – they are written in the plain file by the same manner (as parts of records) and extend the balanced index in one or other its section which takes practically same time.

In NL-ArM the different values of co-ordinates cause various archive structures which take corresponded time for combinations of values. Practically, NL-ArM creates hyper-matrix and large empty zones need additional resources – time and disk space, which are not so great due to smart internal index organization but really exists.

Comparison of Firebird and NL-ArM writing times for the case of large empty zones in the matrix is given in Table 18. It is a sub-table from Table 16 and numbers of rows are the same.

Table 18. Comparison of Firebird and NL-ArM for the case of large empty zones in the matrix

row No.:	initial values of co-ordinates		size of intervals of co-ordinates		Number of cells	writing time (ms)		ratio
						Firebird	NL-ArM	Firebird : NL-ArM
	(X)	(Y)	(ΔX)	(ΔY)	(ΔX*ΔY)	(ms)	(ms)	
7	1	1	10000	10	100000	155578	1719	*90 : 1*
4	1	1	10	10000	100000	160094	1563	*102 : 1*
10	1000000	1000000	10000	10	100000	156656	2109	*74 : 1*
11	1000000	1000000	10	10000	100000	162000	1672	*96 : 1*

The influence of storing types is presented in Table 19. Visualization of ratios is shown on Figure 31.

For NL-ArM we have two cases:

1. Row oriented NL-ArM storing.
2. Column oriented NL-ArM storing.

Firebird is not so sensitive to the NL-ArM row and column oriented cases because these cases are only switched parts of keyword and the key length is the same. For NL-ArM the second case is more suitable because of column oriented hierarchical storing. Let remember, NL-ArM has multi-layer structure of perfect hash tables with 2^{32} entries. In addition, the NL-ArM hash tables have balanced internal indexes specially adapted for storing large data sets. Because of this, they are not so effective for small values of co-ordinates. For instance, the better ratio for the rectangle with starting point 1000000x1000000 is just due to special internal indexing of NL-ArM which is adapted to great co-ordinate values.

This conclusion is seen on Figure 31 where the Firebird ration line is practically horizontal but NL-ArM ratio line descends.

Table 19. Influence of storing types

case	type of ratio	Firebird	NL-ArM
1.	column to row oriented case (ratio of row 4 to row 7)	**1.029**	**0.909**
2.	column to row oriented case for the rectangle with starting point 1000000x1000000 (ratio of row 11 to row 10)	**1.034**	**0.793**

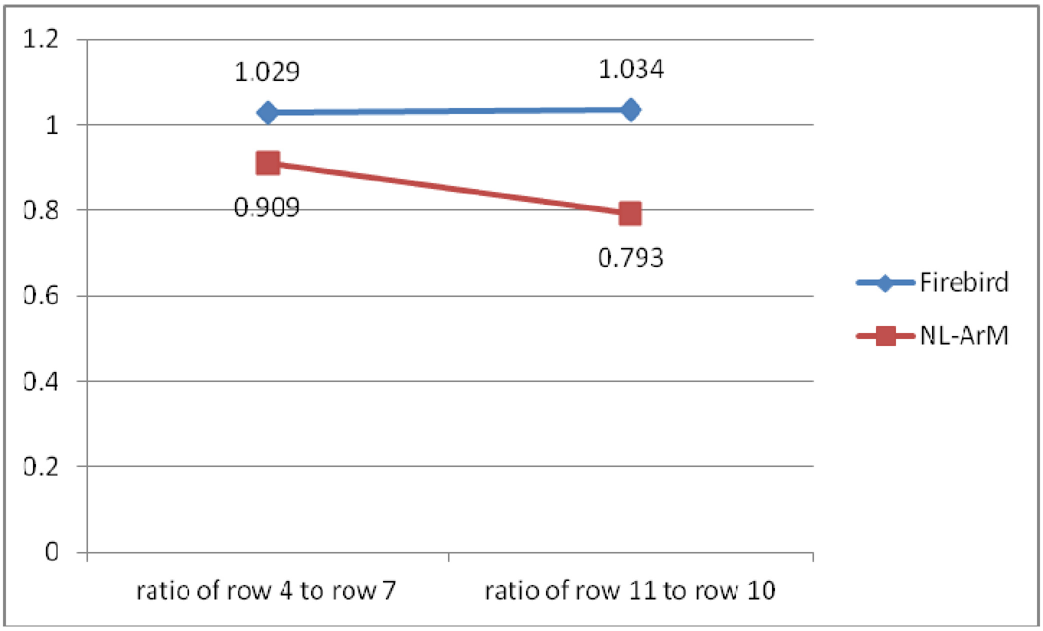


Figure 31. Ratios for NL-ArM row and column oriented writing

The influence of offset (1000000) is presented in Table 20. Visualization of ratios is shown on Figure 32.

Again, for NL-ArM we have two cases:

1. Row oriented NL-ArM storing.
2. Column oriented NL-ArM storing.

As in previous, Firebird is not so sensitive to the NL-ArM row and column oriented cases because they are only switched parts of keyword and the key length is the same. For NL-ArM the second case is more suitable because of column oriented hierarchical storing.

This conclusion is seen on Figure 32 where the Firebird ratio line is practically horizontal but NL-ArM ratio line descends.

Table 20. Influence of the offset from 1 to 1000000

case	**type of ratio**	**Firebird ratio**	**NL-ArM ratio**
1.	row oriented storing (ratio of row 10 to row 7)	**1.007**	**1.227**
2.	column oriented storing (ratio of row 11 to row 4)	**1.012**	**1.070**

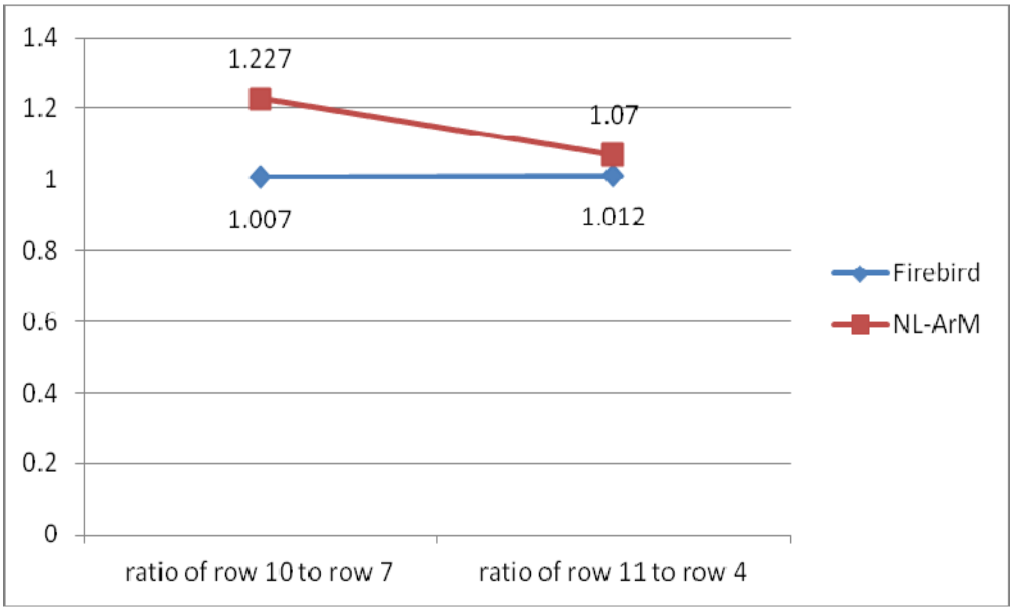


Figure 32. Ratios for the offset from 1 to 1000000

Concluding this part of experiments we have to note that the relations in Table 16 show that in writing experiments, regarding NL-ArM, Firebird is on average ***90.1 times slower.*** This result is due to two reasons. The first is that balanced indexes of Firebird need reconstruction for including of every new keyword. This is time consuming process. The second reason is the speed of updating NL-ArM hash tables which do not need recompilation after including new information.

Due to specific of realization, for small values of co-ordinates NL-ArM is not as effective as for the great ones.

Nevertheless, NL-ArM is always many times faster than Firebird.

If we need direct access to large dynamic data sets (via NL-path), than more convenient are hash based tools like NL-ArM. For instance, such cases are large ontologies and RDF-graphs.

➢ *Comparison of reading time characteristics*

The experimental data for reading time characteristics are given in Table 21, which has similar format as one for the writing time characteristics.

The experiments were done on the base of 6830000 queries with 8 or 14 byte numbers as keywords (to model two-dimensional 4-bytes binary co-ordinates), stored in a text file in order to maintain equivalence of Firebird with NL-ArM.

The columns of Table 21, marked as (X) and (Y), contain the co-ordinates of the initial point of the experimental rectangular, i.e. initial values of the first and second parts of the keywords.

The diagonal point is given by the corresponded offsets (ΔX) and (ΔY). In the next column the quantity of read elements is given, respectively – 10 000, 100 000, and 1 000 000.

The reading time has been measured in milliseconds (ms) and the results are presented in the next two columns. In the last column, the ratio between Fireburd and NL-ArM for reading time is given.

Table 21. Reading time comparison of Firebird and NL-ArM

	initial values of co-ordinates		size of intervals of co-ordinates		Number of elements generated in the specified interval	reading time for 10,000 elements		ratio of Firebird to ArM
						Firebird	ArM	
No.:	(X)	(Y)	(ΔX)	(ΔY)	(ΔX*ΔY)	(ms)	(ms)	
1.	1	1	10	1000	10000	532	156	*3:1*
2.	1	1	100	100	10000	406	172	*2:1*
3.	1	1	1000	10	10000	563	187	*3:1*
4.	1	1	10	10000	100000	1265	172	*7:1*
5.	1	1	100	1000	100000	234	188	*1:1*
6.	1	1	1000	100	100000	672	110	*6:1*
7.	1	1	10000	10	100000	2297	203	*11:1*
8.	1	1	1	100000	100000	13406	125	*107:1*
9.	1	1	100000	1	100000	15953	422	*37:1*
10.	1000000	1000000	10000	10	100000	1906	32	*59:1*
11.	1000000	1000000	10	10000	100000	1265	31	*40:1*
12.	1	1	10	100000	1000000	13547	188	*72:1*
13.	1	1	100	10000	1000000	2562	156	*16:1*
14.	1	1	1000	1000	1000000	1359	125	*10:1*
15.	1	1	10000	100	1000000	3719	250	*14:1*
16.	1	1	100000	10	1000000	21625	204	*106:1*
17.	1000000	1000000	1000	1000	1000000	750	32	*23:1*
				Total:	6830000	82061	2753	*29.8:1*

The average of *reading time* data in milliseconds for the three groups (10 000, 100 000, and 1 000 000) are presented in Table 22. This is illustrated on Figure 33.

Table 22. Average in milliseconds (ms) of reading time data

number of records	reading time		ratio	
	Firebird	NL-ArM	Firebird	NL-ArM
10000	500.33	171.67	*2.67 : 1*	
100000	4624.75	160.38	*33.5 : 1*	
1000000	7260.33	159.17	*40.17 : 1*	

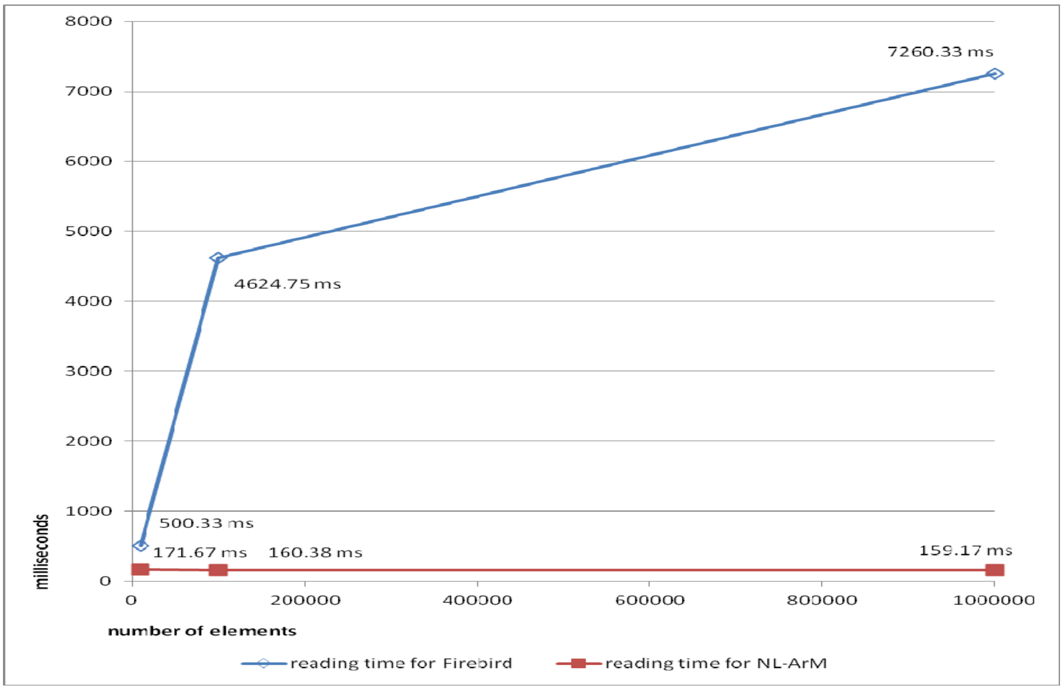


Figure 33. Time in milliseconds (ms) for reading by Firebird and NL-ArM

In the last column of Table 22, the *average ratio* of Firebird and NL-ArM for reading time is given. This relation is illustrated graphically on Figure 34.

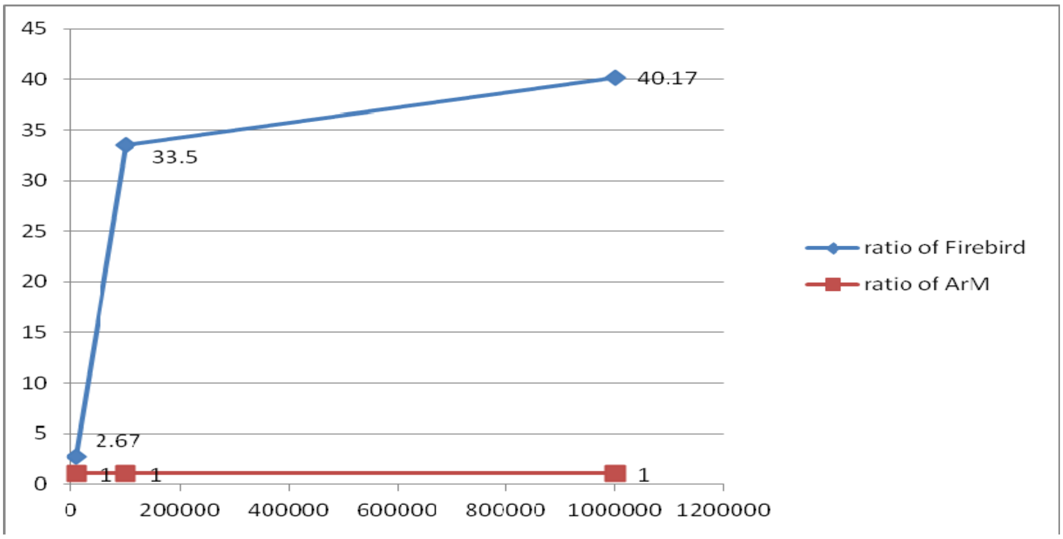


Figure 34. Time relation for reading by Firebird and NL-ArM

Concluding this part of experiments we have to note that the relations in Table 21 show that in reading experiments, regarding NL-ArM, Firebird is on average ***29.8 times slower.***

This result is due to the speed of access in NL-ArM hash tables which do not need search operations.

Again, note that if we need direct access to large dynamic data sets (via NL-path), than more convenient are hash based tools like NL-ArM. For instance, such cases are large ontologies and RDF-graphs.

➢ *Conclusion of chapter 4*

In this chapter two main types of basic experiments were presented. NL-ArM has been compared with (1) sequential text file of records and (2) relational database management system Firebird.

The need to compare NL-ArM access method with text files was determined by practical considerations – in many applications the text files are main approach for storing semi-structured data. To investigate the size of files and speed of their generation we compared writing in a sequential text file and in a NL-ArM archive.

For 8 characters as length of the keywords and small quantity of records, the NL-ArM archive occupies more memory than text file but for the case of very large data the NL-ArM archive is smaller. It is important to underline that these experiments were based on artificial data with fixed length (record of 30 bytes with 8 bytes artificially generated keyword of arbitrary ASCII symbols). If the length of the keywords is variable, the size of NL-ArM archive will be different according of length of the strings of keywords of stored information, i.e. according of number of layers of hash tables (depth of trie).

In sequential storing of records, NL-ArM access method is slower than same operation in text file. For applications where it is important in real time to register incoming information, the text files are preferable than archives with NL-Addressing.

To provide experiments with a relational database, we have chosen the system "Firebird". It should be noted that Firebird and NL-ArM have fundamentally different physical organization of data and the tests cover small field of features of both systems.

We did not compare the sizes of files of NL-ArM and Firebird because of difference of keywords – symbols for Firebird and integer values for NL-ArM.

In writing experiments, regarding NL-ArM, Firebird is on average ***90.1 times slower.*** *This result is due to two reasons. The first is that balanced indexes of Firebird need reconstruction for including of every new keyword. This is time consuming process. The second reason is the speed of updating NL-ArM hash tables which do not need recompilation after including new information. Due to specific of realization, for small values of co-ordinates NL-ArM is not as effective as for the great ones.*

In reading experiments, regarding NL-ArM, Firebird is on average ***29.8 times slower.*** *This result is due to the speed of access in NL-ArM hash tables which do not need search operations.*

If we need direct access to large dynamic data sets (via NL-path), than more convenient are hash based tools like NL-ArM. For instance, such cases are large ontologies and RDF-graphs.

5 Experiments for NL-storing of small datasets

Abstract

In this chapter we will present several experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of small size datasets which contain up to one hundred thousands of instances.

The goal of the experiments with different small datasets, like dictionary, thesaurus or ontology, is to discover regularities in the NL-addressing realization. More concretely, two regularities of time for storing by using NL-addressing will be examined:

— *It depends on number of elements in the instances;*

— *It not depends on number of instances in datasets.*

This chapter starts with introduction of the idea of knowledge representation. Further in the chapter three experiments with small size datasets are outlined: for NL-storing of dictionaries, thesauruses, and ontologies. Presentation of every experiment starts with introductory part aimed to give working definition and to outline state of the art in storing concrete structures.

We start with analyzing the easiest one: NL-storing dictionaries. After that, NL-storing of thesauruses will be analyzed. An experiment with WordNet thesaurus and program WordArM based on NL-addressing will be discussed.

At the end, a special attention will be given to NL-storing ontologies. This part of the chapter begins with introducing the basic ontological structures as well as the corresponded operations and tools for operating with ontologies. Further, NL-storing models for ontologies will be discussed and experiments with OntoArM program for storing ontologies based on NL-addressing will be outlined.

5.1 Knowledge representation

In a letter written to Philip Jourdain in 1914, Gottlob Frege had written:

> “Let us suppose an explorer travelling in an unexplored country sees a high snow-capped mountain on the northern horizon.
>
> By making inquiries among the natives he learns that its name is 'Aphla'. By sighting it from different points he determines its position as exactly as possible, enters it in a map, and writes in his diary: 'Aphla is at least 5000 meters high'.

Another explorer sees a snow-capped mountain on the southern horizon and learns that it is called Ateb. He enters it in his map under this name.

Later comparison shows that both explorers saw the same mountain. Now the content of the proposition 'Ateb is Aphla' is far from being a mere consequence of the principle of identity, but contains a valuable piece of geographical knowledge. What is stated in the proposition 'Ateb is Aphla' is certainly not the same thing as the content of the proposition 'Ateb is Ateb'.

Now if what corresponded to the name 'Aphla' as part of the thought was the reference of the name and hence the mountain itself, then this would be the same in both thoughts. The thought expressed in the proposition 'Ateb is Aphla' would have to coincide with the one in 'Ateb is Ateb', which is far from being the case. What corresponds to the name 'Ateb' as part of the thought must therefore be different from what corresponds to the name 'Aphla' as part of the thought. This cannot therefore be the reference which is the same for both names, but must be something which is different in the two cases, and I say accordingly that the sense of the name 'Ateb' is different from the sense of the name 'Aphla'.

Accordingly, the sense of the proposition 'Ateb is at least 5000 meters high' is also different from the sense of the proposition 'Aphla is at least 5000 meters high'. Someone who takes the latter to be true need not therefore take the former to be true. An object can be determined in different ways, and every one of these ways of determining it can give rise to a special name, and these different names then have different senses; for it is not self-evident that it is the same object which is being determined in different ways.

We find this in astronomy in the case of planetoids and comets. Now if the sense of a name was something subjective, then the sense of the proposition in which the name occurs, and hence the thought, would also be something subjective, and the thought one man connects with this proposition would be different from the thought another man connects with it; a common store of thoughts, a common science would be impossible.

It would be impossible for something one man said to contradict what another man said, because the two would not express the same thought at all, but each his owns.

For these reasons I believe that the sense of a name is not something subjective (crossed out: in one's mental life), that it does not therefore belong to psychology, and that it is indispensable” [Frege, 1980].

What is important in this example is [Ivanova et al, 2013c]:

- The names Ateb and Aphla refer different parts of the same natural object (mountain, let call it *Pirrin*);
- The position of the referred object (mountain) is fixed by any artificial system (geographical co-ordinates, address) which is another name of the same object;
- The names and the address correspond one to another and both to the real object but without the explorer’s map, respectively – the explorer’s diary, it is impossible to restore the correspondence;

- At the end, the names Ateb and Aphla are connected hierarchically to the name Pirrin and the relations are:
  - Aphla *is_a_South_Side_of* Pirrin;
  - Ateb *is_a_North_Side_of* Pirrin.

The last case forms a simple vocabulary (Table 23):

Table 23. A simple vocabulary

name	*definition*
Aphla	The South Side of Pirrin mountain
Ateb	The North Side of Pirrin mountain
Pirrin	A mountain in the unexplored country with co-ordinates (x,y)

In addition, all cases given above form a simple ontology with four concepts which may be represented by a graph (Figure 35):

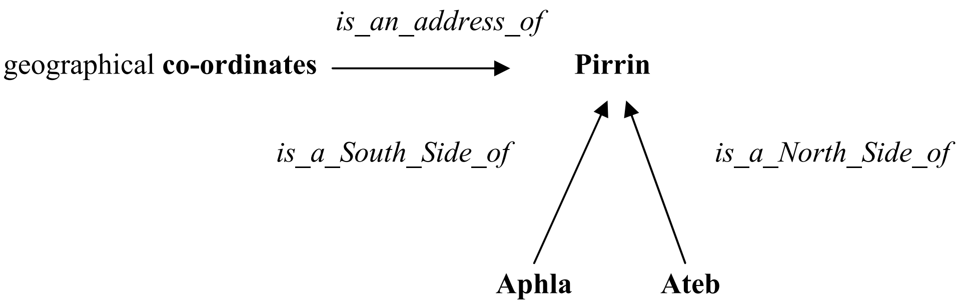


Figure 35. A simple ontology

The same information may be represented by a table (Table 24):

Table 24. A simple ontology

object	*is_a_South_Side_of*	*is_a_North_Side_of*	*is_an_address_of*
Pirrin	**Aphla**	**Ateb**	**co-ordinates**

What vocabularies, taxonomies, thesauruses, and ontologies, **all have in common** are [Pidcock & Uschold, 2012]:

- They are approaches to help structure, classify, model, and or represent the concepts and relationships pertaining to some subject matter of interest to some community;

- They are intended to enable a community to come to agreement and to commit to use the same terms in the same way;
- There is a set of terms that some community agrees to use to refer to these concepts and relationships;
- The meaning of the terms is specified in some way and to some degree;
- They are fuzzy, ill-defined notions used in many different ways by different individuals and communities.

The **major differences** that distinguish these approaches [Pidcock & Uschold, 2012]:

- How much meaning is specified for each term?
- What notation or language is used to specify the meaning?
- What is the thing for? Taxonomies, thesauruses, and ontologies have different but overlapping uses.

At the end, some additional information may be connected to the names. For instance, it may be the type of mountain, minerals found, some photos, textual descriptions, etc. All such information is connected to names and has to be accessed by names as keywords or paths to it, i.e. its computer representation has to be organized using corresponded pointers, indexes of keyword, etc.

In this case the concept “knowledge representation” is used. As we have seen above, the ontologies are useful approach for knowledge representation, which is understandable for humans as well as for the specialized software.

Knowledge representation is closely connected to data models, i.e. the information structures used for organizing the information in the internal or external computer memory. In other words, knowledge representation is depended on the storing patterns and program tools for accessing data.

Below in this chapter and in the next chapter, storing of knowledge, represented by structured and semi-structured data sets, will be discussed from point of view of using the NL-addressing and NL-ArM for this purpose. Results from provided experiments will be analyzed.

In this chapter we will present several experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of small size datasets which contain up to one hundred thousands of instances.

The experiments were provided at PC SONY Vaio, with Intel® Core™2 Duo CPU T9550 @ 2.66GHz 2.67GHZ, RAM 4.00 GB, 64-bit operating system Windows 7 Ultimate SP1.

The goal of the experiments with different small datasets, like dictionary, thesaurus or ontology, is to discover regularities in the NL-addressing realization. More concretely, two regularities of time for storing by using NL-addressing will be examined:

- It depends on number of elements in the instances;
- It not depends on number of instances in datasets.

5.2 Experiment for NL-storing dictionaries

Our first experiment is to realize a small multi-language dictionary based on NL-addressing. For this purpose, we have taken data from the popular in Bulgaria “SA Dictionary” [Angelov, 2012].

SA Dictionary is a computer dictionary, which translates words from Bulgarian language to English and vice versa.

For experiments we take a list of ***23 412*** words in English and Bulgarian with their definitions in Bulgarian, stored in a sequential file with size of 2 410 KB.

The experimental program "WordArM" used for the experiments is specially designed for storing dictionaries and thesauruses based on NL-addressing. It is outlined in the Appendix A.

➢ *Definition of dictionary*

For the purposes of this research, next definition of dictionary is appropriate:

Dictionary*: a reference resource, in printed or electronic form, that consists of an alphabetical list of words with their meanings and parts of speech, and often a guide to accepted pronunciation and syllabification, irregular inflections of words, derived words of different parts of speech, and etymologies* [Collins, 2003]*:*

This definition is modeled by the construction:

$$\text{<name>} \longrightarrow \text{<definition>}.$$

➢ *Multi-language dictionary based on NL-addressing*

For storing dictionaries we use simple model: the words (concepts) are used as paths to theirs definitions stored in corresponded terminal containers.

The speed for storing, accessing, and size of the work memory and permanent archives are given in Table 25.

Work memory is the memory taken for storing hash tables and service information during the work of NL-ArM. Usually it has to be part of main computer memory. To analyze its real size in our experiments, work memory is allocated as file.

Permanent archives are static copies of work memory (zipped files), aimed for long storing the information. They have to be of small size and converting to and from expanded work memory structures has to be quick (usually several seconds or minutes). For compressing of work memory we use a separate archiving program.

Table 25. Experimental data for NL-storing of a dictionary

operation	number of instances	total time in milliseconds	average time for one instance	work memory	permanent archive
NL-writing	23 412	22 105	0.94 ms	80 898 KB	5 938 KB
NL-reading	23 412	20 826	0.89 ms		

The work memory taken during the work was ***80 898*** KB.

After finishing the work, occupied permanent compressed archive is ***5 938*** KB. This means that the NL-indexing takes 5 102 KB additional compressed disk memory (the sequential file with initial data is 2 410 KB and compressed by WinZip it is 836 KB).

To analyze work of the system, work memory was chosen to be in a file but not in the main memory. In further realizations of WordArM, it may be realized as a part of main memory of computer as:

- Dynamically allocated memory;
- File mapped in memory.

In this case, the speed of storing and accessing will be accelerated and used hard disk space will be reduced.

The analysis of the results in Table 25 shows that the NL-addressing in this realization permits access practically equal for writing and reading for all data.

The speed is more than a thousand instances per second.

Reading is possible immediately after writing and no search indexes are created.

5.3 Experiment for NL-storing thesauruses

The NL-storing model for vocabularies was simple because the one-one correspondence “word-definition”.

The storing models for thesauruses are more complicated due to existing more than one corresponded definitions for a given word (synonyms). Because of this, below we will outline and analyze one such model – the storing model of WordNet thesaurus [WordNet, 2012].

The idea is to use NL-addressing to realize the WordNet lexical database and this way to avoid recompilation of its database after every update.

The program used for the experiments is “WordArM” (see Appendix A).

➢ *Definition of thesaurus*

For the purposes of this research, the next definition of ***thesaurus*** is appropriate:

***Thesaurus**: a book or catalog of words and their synonyms and antonyms* [YourDictionary, 2013].

A **thesaurus** is a networked collection of controlled vocabulary terms. This means that a thesaurus uses associative relationships in addition to parent-child relationships. The expressiveness of the associative relationships in a thesaurus varies and can be as simple as “correlated to terms” as in term **A** is related to term **B**. The thesaurus has two kinds of links: broader/narrower term, which is much like the generalization/specialization link, but may include a variety of others [Pidcock & Uschold, 2012].

➢ *WordNet thesaurus*

WordNet® is a large lexical database of English (http://WordNet.princeton.edu).

Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser.

WordNet is freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing [Fellbaum et al, 1998; Miller, 1995].

WordNet was created and is being maintained at the Cognitive Science Laboratory of Princeton University under the direction of psychology professor George A. Miller. Development began in 1985.

As of November 2012 WordNet's latest Online-version is 3.1 (announced on June 2011), but latest released version is 3.0 (released on December 2006).

The 3.0 database contains *155 287 words* organized in *117 659 synsets* for a total of *206 941 word-sense pairs*; in compressed form, it is about 12 megabytes in size [WordNet, 2012].

➢ *WordNet system*

The mathematical model of the WordNet is a graph V = (X, R), where X is the set of graph nodes, and R is the set of edges between them.

The set X is divided into two disjoint subsets: $X = X_1 \cup X_2$, $X_1 \cap X_2 = \emptyset$. The nodes from X_1 correspond to the words and phrases, and nodes from X_2 - to their meanings (interpretations). Each of meanings correlates to one of the parts of speech: noun, verb, adjective or adverb.

The set of edges is also divided into two subsets which are not intersecting: $R = R_1 \cup R_2$, $R_1 \cap R_2 = \emptyset$. Edges of R_1 connect the words with theirs meanings i.e. the elements X_1 with elements X_2. These edges represent relationships which belong to X_1 x X_2. Edges of R_2 connect "words with words" and "meanings with meanings”, i.e. represent relationships that belong to X_1 x X_1 and X_2 x X_2 respectively [Bashmakov, 2005]. Other types of relationships are defined by typification of edges from R_2.

Technically, the WordNet is an electronic thesaurus which defines a wide range of meanings of words bounded together by semantic pointers. WordNet logical structure is shown in Figure 36.

In developing WordNet lexical database, it has been convenient to divide the work into two interdependent tasks which bear a vague similarity to the traditional tasks of writing and printing a dictionary [Fellbaum, 1998]:

- One task is to write the source files that contain the basic lexical data — the contents of those files are the lexical substance of WordNet;
- The second task is to create a set of computer programs that would accept the source files and do all the work leading ultimately to the generation of a display for the user.

The WordNet system falls naturally into four parts:

- The WordNet lexicographers’ source files;
- The software utility called “Grinder” aimed to ***convert*** lexicographers’ source files into the WordNet lexical database;
- The WordNet lexical database;
- And the suite of software tools used to access the database.

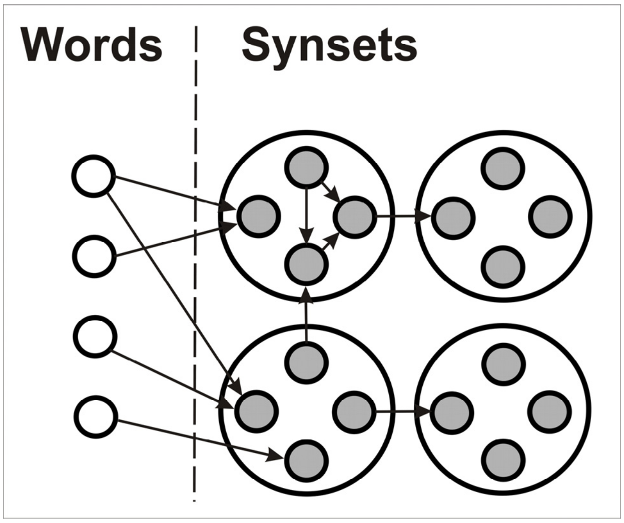


Figure 36. Logical structure of the WordNet

WordNet's source files are written by lexicographers. They are the product of a detailed relational analysis of lexical semantics: a variety of lexical and semantic relations are used to represent the organization of lexical knowledge.

The "Grinder" utility compiles the lexicographers' files. It verifies the syntax of the files, resolves the relational pointers, then generates the WordNet database that is used with the retrieval software and other research tools. ***To build a complete WordNet database, all of the lexicographers' files must be processed at the same time***.

The main relation among words in WordNet is synonymy, as between the words "shut" and "close" or "car" and "automobile". Synonyms (words that denote the same concept and are interchangeable in many contexts) are grouped into unordered sets (synsets). Each of WordNet's 117 000 synsets is linked to other synsets by means of a small number of "conceptual relations". Additionally, a synset contains a brief definition ("gloss") and, in most cases, one or more short sentences illustrating the use of the synset members. Word forms with several distinct meanings are represented in as many distinct synsets. Thus, each form-meaning pair in WordNet is unique [WordNet, 2012].

Consider a representation of a synset of the word "accession" in the WordNet lexical database:

00047131 04 n 02 accession 0 addition 0 001 @ 09536731 n 0000 |
something added to what you have already;
"the librarian shelved the new accessions";
"he was a new addition to the staff"

The number 00047131 is a unique identifier of the synset of the noun {accession, addition}. The part of the record between the symbols "@" and "|" indicates that this synset is subordinated to the synset with ID 09536731 which correspond to meaning "acquisition". The last part of the record (after the symbol "|") is interpretation of synset and some examples of using the words included in the synset.

From a software standpoint, this record requires a number of additional indexes for service the access, which of course needs additional resources.

As an example, consider the information about the word "accession". As an answer to the request for "accession", the WordNet system returns the following information (Figure 37):

The noun accession has 6 senses (no senses from tagged texts)

1. **{13251723}** <noun.process> accession#1 -- (a process of increasing by addition (as to a collection or group); "the art collection grew through accession")
2. **{13170404}** <noun.possession> accession1#2 -- ((civil law) the right to all of that which your property produces whether by growth or improvement)
3. **{13082910}** <noun.possession> accession#3, addition#4 -- (something added to what you already have; "the librarian shelved the new accessions"; "he was a new addition to the staff")
4. **{07078650}** <noun.communication> accession2#4, assenting#1 -- (agreeing with or consenting to (often unwillingly); "accession to such demands would set a dangerous precedent"; "assenting to the Congressional determination")
5. **{05115154}** <noun.attribute> entree#2, access#1, accession#5, admittance#1 -- (the right to enter)
6. **{00232781}** <noun.act> accession3#6, rise to power#1 -- (the act of attaining or gaining access to a new office or right or position (especially the throne); "Elizabeth's accession in 1558")

The verb accession has 1 sense (no senses from tagged texts)

1. **{00989696}** <verb.communication> accession#1 -- (make a record of additions to a collection, such as a library)

Figure 37. Answer by WordNet system to a query for the word "accession"

➢ *WordNet storing model*

The WordNet database is in an ASCII format that is human- and machine-readable, and is easily accessible to those who wish to use it with their own applications.

There are two main types of database files:

- Data file - contains all of the lexicographic data gathered from the lexicographers' files for the corresponding syntactic category, with relational pointers resolved to addresses in data files;
- Index file - an alphabetized list of all of the word forms in WordNet for the corresponding syntactic category.

WordNet stores information about words in four main data files and four main index files (for nouns, verbs, adjectives and adverbs).

The index and data files are interrelated.

The data structure is the same in each of data files – one or more synsets are stored for every word and the access is performed by the address of the first bytes of the synsets, which is apparently given by an eight digit number beginning namely in this byte (Figure 38 and Figure 39). This value is the unique identifier of the synset. It is its "***relative address***" from the beginning (first byte) of the file. The synset data elements are separated by spaces. We should note that links to other synsets are given again by the relative addresses.

13251723 22 n 01 **accession** 0 001 @ **13323403** n 0000 | a process of increasing by addition (as to a collection or group); "the art collection grew through accession"

13170404 21 n 01 **accession** 1 002 @ **13070995** n 0000 ;c **08338303** n 0000 | (civil law) the right to all of that which your property produces whether by growth or improvement

13082910 21 n 02 **accession** 0 addition 0 001 @ **13082742** n 0000 | something added to what you already have; "the librarian shelved the new accessions"; "he was a new addition to the staff"

07078650 10 n 02 **accession** 2 assenting 0 002 @ **07076600** n 0000 + **00795631** v 0102 | agreeing with or consenting to (often unwillingly); "accession to such demands would set a dangerous precedent"; "assenting to the Congressional determination"

05115154 07 n 04 entree 0 access 0 **accession** 0 admittance 0 003 @ **05113619** n 0000 + **02426186** v 0401 ~ **05119817** n 0000 | the right to enter

00232781 04 n 02 **accession** 3 rise_to_power 0 003 @ **00060914** n 0000 + **01989112** v 0101 + **02358456** v 0101 | the act of attaining or gaining access to a new office or right or position (especially the throne); "Elizabeth's accession in 1558"

Figure 38. Synsets of the word "accession" in WordNet data file for nouns

00989696 32 v 01 **accession** 0 002 @ **00990286** v 0000; c **00897092** n 0000 01 + 08 00 | make a record of additions to a collection, such as a library

Figure 39. Synsets of the word "accession" in WordNet data file for verbs

What is important for us now is the ***algorithm of reaching the synsets***.

There are four index files of WordNet (for nouns, verbs, adjectives and adverbs). They are sorted in alphabetical order of words and for each word a special record is stored at separated line. Its structure is clear: at the first place the word is given and, after some coded information, the relative addresses of the corresponded synsets in corresponded data files age given (Figure 40 and Figure 41).

```
accession n 6 4 @ ~ + ; 6 0 13251723 13170404 13082910 07078650
05115154 00232781
```

Figure 40. Record for the word "accession" in the index of nouns

```
accession v 1 2 @ ; 1 0 00989696
```

Figure 41. Record for the word "accession" in the index of verbs

To reach all synsets of a word, firstly a binary search is made in all index files, the corresponded relative addresses are collected and then system reads the synsets directly from the data files.

Algorithmic complexity in this case is O(log(n_n)+log(n_v)+log(n_a)+log(n_r)), where n_n, n_v, n_a and n_r are the quantities of nouns, verbs, adjectives and adverbs, respectively.

There is a second way to reach synsets. It is served by so called "*sense index*". This index is also sorted, but for every word there exist as much records as number of synsets exists for given word in all data files. For example, the word accession has seven records: six for its meanings as a noun and one for its meaning as a verb. Each record contains only one relative address of a synset (Figure 42).

In this case, to reach all synsets of a word, firstly a binary search is made in the sense index and the corresponded relative addresses are collected from all records for the word. Then, the system reads the synsets directly from the data files.

Algorithmic complexity in this case is greater than O(log(n)), n= n_n+n_v+n_a+n_r is the total number of words in the database (nouns + verbs + adjectives + adverbs), because the words may be repeated many times, and further work is needed to retrieve all occurrences of the word.

```
accession%1:04:03:: 00232781 6 0
accession%1:07:00:: 05115154 5 0
accession%1:10:02:: 07078650 4 0
accession%1:21:00:: 13082910 3 0
accession%1:21:01:: 13170404 2 0
accession%1:22:00:: 13251723 1 0
accession%2:32:00:: 00989696 1 0
```

Figure 42. Records for the word "accession" in the sense index

➢ *Disadvantages of WordNet storing model*

The WordNet storing model permits quick response of the system during its everyday using. The (binary) search in four types sorted index files and one general sense index, using corresponded hash tables, allows high speed of the search and, based on it, extracting the needed information via direct access based on the relative addresses in the data files.

Many disadvantages of the WordNet organization are discussed in [Poprat et al, 2008]. Due to importance of them, below we will include larger citations.

When the WordNet project started more than two decades ago, markup languages such as SGML or XML were unknown. Because of this reason, a rather idiosyncratic, fully text-based data structure for these lexicographic files was defined in a way to be readable and editable by humans — and survived until today. This can really be considered as *an outdated legacy* given the fact that the WordNet community has been so active in the last years in terms of data collection, but has refrained from adapting its data formats in a comparable way to today's specification standards.

There are two types of problems founded for the data format underlying the WordNet lexicon and the software that helps building a WordNet file and creating an index for this file:

- First, WordNet's data structure puts several restrictions on what can be expressed in a WordNet lexicon. For example, it constrains lexical information to a fixed number of homonyms and a fixed set of relations;
- Second, the data structure imposes a number of restrictions on the string format level.

If these restrictions are violated the WordNet processing software throws error messages which differ considerably in terms of informativeness for error tracing and detection or even do not surface at all at the lexicon builder's administration level.

In addition, it seems that the length of a word is restricted to 425 characters and synsets are only allowed to group up to 988 direct hyponymous synsets.

According to our experiences the existing WordNet software *is hardly (re)usable* due to insufficient error messages that the software throws and limited documentation [Poprat et al, 2008].

In terms of the actual representation format of WordNet we found that using the current format is not only cumbersome and error-prone, but also limits what can be expressed in a WordNet resource [Poprat et al, 2008].

From our perspective this indicates the need for a *major redesign* of WordNet's data structure foundations to keep up with the standards of today's meta data specification languages (e.g., based on RDF [Graves & Gutierrez, 2006], XML or OWL [Lungen et al, 2007]). We encourage the reimplementation of WordNet resources based on such a state-of-the-art markup language (for OWL in particular a representation of WordNet is already available [van Assem et al, 2006].

Of course, if a new representation format is used for a WordNet resource also the software accessing the resource has to be adapted to the new format. This may require substantial implementation efforts that we think are worth to be spent, if the new format overcomes the major problems that are due to the original WordNet format [Poprat et al, 2008].

Finally, one more shortcoming of the WordNet database's structure is that although all the files are in ASCII, and are therefore editable, and in theory extensible, in practice *this is almost impossible*.

The end user has access only to the static ("compiled") version of the database, which couldn't be extended and further developed. In addition, due to relative addresses that are used as pointers, any change which cause alteration of the number of bytes in any data file makes it unusable and it must be recompiled as well as the corresponded index files.

One of the Grinder's primary functions is the calculation of addresses for the synsets in the data files. Editing any of the database files would (most likely) create incorrect byte offsets, and would thus derail many searching strategies. At the present time, building a WordNet database requires the use of the Grinder and the *processing of all lexicographers' source files at the same time* [Fellbaum, 1998].

Let see a small example shown on Figure 43 - two variants of the synset of the word “accession” from different compilations of the data file for nouns:

(a) Version of WordNet from August, 2012;

(b) An older version of WordNet from 2011 year, published in [Palagin et al, 2011].

The difference between relative addresses is visible on Figure 43.

```
13082910 21 n 02 accession 0 addition 0 001 @ 13082742 n 0000 |
        something added to what you already have; "the librarian shelved the new
        accessions"; "he was a new addition to the staff"
                              a) version from the 2012 year

00047131 04 n 02 accession 0 addition 0 001 @ 09536731 n 0000 |
        something added to what you have already; "the librarian shelved the new
        accessions"; "he was a new addition to the staff"
                              b) version from the 2011year
```

Figure 43. Synset the word "accession" from the data file for nouns

In the main, the WordNet database organization has the following important disadvantages:

1. Relative addressing is convenient for the computer processing, but it is difficult to be used by the customer;
2. Manual creating of numerical addresses is impossible, and their use can be done only by the special program;
3. The end user has access only to the static ("compiled") version of the database, which couldn't be extended and further developed;
4. Building the WordNet database requires the use of the Grinder and the processing of all lexicographers' source files at the same time;
5. Using the current format is not only cumbersome and error-prone, but also limits what can be expressed in a WordNet resource.

We are going to experiment to realize WordNet lexical database without using relative addresses as pointers and this way to avoid the pointed above limitations and recompilation of the database after every update.

➢ *Experiment to store WordNet by NL-addressing*

The main source information of WordNet is published as lexicographer files.

The names of the WordNet lexicographer files and their corresponding file numbers are listed in Table 26, along with a brief description of each file's content and corresponded number of included instances (synsets).

The total number of instances (file records) is 117 871.

206 instances contain service information but not concepts' definitions, so we have 117 665 instances for experiments, distributed in 45 thematically organized lexicographer files.

It is important to note that there is equal synsets in several lexicographer files. This has matter when we integrate the 45 files in one source file for representing a thesaurus.

Table 26. WordNet lexicographer files

No.:	Name	Content	number of instances
01	adj.all	all adjective clusters	14435
02	adj.pert	relational adjectives (pertainyms)	3661
03	adj.ppl	participial adjectives	60
04	adv.all	all adverbs	3621
05	noun.Tops	unique beginner for nouns	51
06	noun.act	nouns denoting acts or actions	6650
07	noun.animal	nouns denoting animals	7514
08	noun.artifact	nouns denoting man-made objects	11587
09	noun.attribute	nouns denoting attributes of people and objects	3039
10	noun.body	nouns denoting body parts	2016
11	noun.cognition	nouns denoting cognitive processes and contents	2964
12	noun.communication	nouns denoting communicative processes and contents	5607
13	noun.event	nouns denoting natural events	1074
14	noun.feeling	nouns denoting feelings and emotions	428
15	noun.food	nouns denoting foods and drinks	2574
16	noun.group	nouns denoting groupings of people or objects	2624
17	noun.location	nouns denoting spatial position	3209
18	noun.motive	nouns denoting goals	42
19	noun.object	nouns denoting natural objects (not man-made)	1545
20	noun.person	nouns denoting people	11088
21	noun.phenomenon	nouns denoting natural phenomena	641

No.:	Name	Content	number of instances
22	noun.plant	nouns denoting plants	8159
23	noun.possession	nouns denoting possession and transfer of possession	1061
24	noun.process	nouns denoting natural processes	770
25	noun.quantity	nouns denoting quantities and units of measure	1350
26	noun.relation	nouns denoting relations between people or things or ideas	437
27	noun.shape	nouns denoting two and three dimensional shapes	342
28	noun.state	nouns denoting stable states of affairs	3544
29	noun.substance	nouns denoting substances	2983
30	noun.time	nouns denoting time and temporal relations	1028
31	verb.body	verbs of grooming, dressing and bodily care	547
32	verb.change	verbs of size, temperature change, intensifying, etc.	2383
33	verb.cognition	verbs of thinking, judging, analyzing, doubting	695
34	verb.communication	verbs of telling, asking, ordering, singing	1548
35	verb.competition	verbs of fighting, athletic activities	459
36	verb.consumption	verbs of eating and drinking	243
37	verb.contact	verbs of touching, hitting, tying, digging	2196
38	verb.creation	verbs of sewing, baking, painting, performing	694
39	verb.emotion	verbs of feeling	343
40	verb.motion	verbs of walking, flying, swimming	1408
41	verb.perception	verbs of seeing, hearing, feeling	461
42	verb.possession	verbs of buying, selling, owning	847
43	verb.social	verbs of political and social activities and events	1106
44	verb.stative	verbs of being, having, spatial relations	756
45	verb.weather	verbs of raining, snowing, thawing, thundering	81
		TOTAL:	117871

Let see an example of two variants of the synset of the word "accession" (Figure 44): (a) WordNet version and (b) NL-version.

13082910 21 n 02 **accession** 0 addition 0 001 @ **13082742** n 0000 | something added to what you already have; "the librarian shelved the new accessions"; "he was a new addition to the staff"

a) WordNet version

accession 21 n 02 **;** 0 addition 0 001 @ **acquisition** n 0000 | something added to what you already have; "the librarian shelved the new accessions"; "he was a new addition to the staff"

b) NL-version

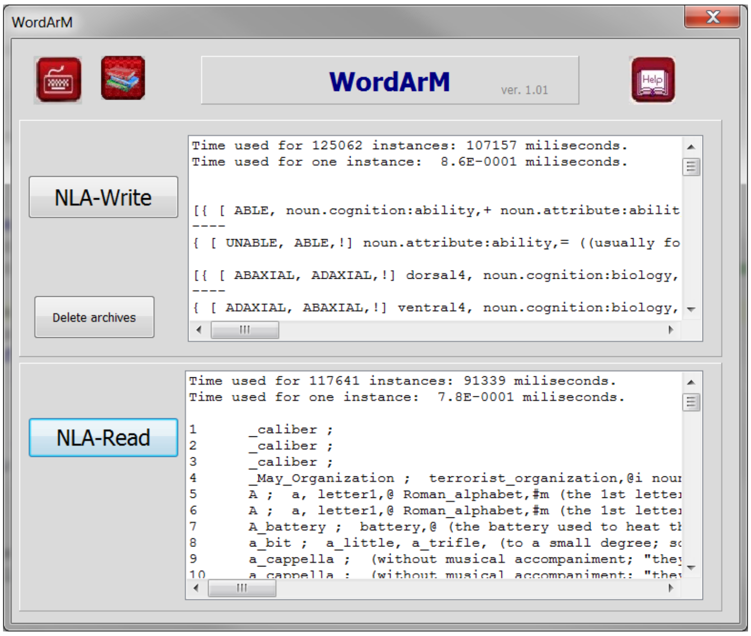
Figure 44. WordNet and NL-versions of the synset of the word "accession"

This example gives idea to experiment using NL-addressing to realize the WordNet lexical database without using relative addresses and this way to avoid the limitations and recompilation of the database after every update.

For experiments we have used files from Table 26 as source in two variants:

1. All 45 files concatenated in one big file as thesaurus with more than one hundred thousands of concepts.
2. Every file was assumed as different layer of WordNet ontology.

The fist case will be discussed below, and the second case will be outlined in the next section. The program used for experiments in the first case is “WordArM” (see Appendix A). A screenshot from the WordArM for the case of WordNet as thesaurus is shown at Figure 45. The results are given in Table 27.



```
Time used for 125062 instances: 107157 miliseconds.
Time used for one instance:  8.6E-0001 miliseconds.


[{ [ ABLE, noun.cognition:ability,+ noun.attribute:abilit
----
{ [ UNABLE, ABLE,!] noun.attribute:ability,= ((usually fo

[{ [ ABAXIAL, ADAXIAL,!] dorsal4, noun.cognition:biology,
----
{ [ ADAXIAL, ABAXIAL,!] ventral4, noun.cognition:biology,
```

```
Time used for 117641 instances: 91339 miliseconds.
Time used for one instance:  7.8E-0001 miliseconds.

1       _caliber ;
2       _caliber ;
3       _caliber ;
4       _May_Organization ;  terrorist_organization,@i nour
5       A ;  a, letter1,@ Roman_alphabet,#m (the 1st letter
6       A ;  a, letter1,@ Roman_alphabet,#m (the 1st letter
7       A_battery ;  battery,@ (the battery used to heat th
8       a_bit ;  a_little, a_trifle, (to a small degree; sc
9       a_cappella ;  (without musical accompaniment; "they
10      a_cappella ;  (without musical accompaniment; "they
```

Figure 45. WordArM results for the case of WordNet as thesaurus

Table 27. Experimental data for storing WordNet as thesaurus

operation	number of instances	total time in milliseconds	average time for one instance
writing	125 062	107 157	0.86 ms
reading	117 641	91 339	0.78 ms
work memory: 385 538 KB; permanent archive: 15 603 KB; source text: 1 333 KB			

We receive practically the same results as for storing dictionaries.

The analysis of the results in Table 27 shows that the NL-addressing permits access practically equal for writing and reading for all data. The speed is more than a thousand instances per second. Reading is possible immediately after writing and no search indexes are created.

The work memory for hash tables and their containers taken during the work of WordArM was *385 538 KB*. To analyze work of the system, work memory was chosen to be in a file in the external memory. In further realizations, to accelerate the speed and reduce of used disk space, the work memory may be realized as part of main memory (as dynamically allocated memory or as file mapped in memory).

After finishing the work, occupied permanent archive for compressed archive is *15 603 KB*, i.e. in this case the NL-indexing takes 14 270 KB additional compressed memory (the sequential file with initial data is 1 333 KB).

- ***What we gain and loss using NL-Addressing for storing thesauruses?***

The loss is additional memory for storing structures which serve NL-addressing. But the same if no great losses we will have if we will build balanced search trees or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published.

The benefit is in two main achievements:

1. High speed for storing and accessing the information.
2. The possibility to access the information immediately after storing without recompilation the database and rebuilding the indexes.

5.4 Experiment for NL-storing ontologies

Storing graphs and ontologies has one important aspect – the layers which correspond to types of relations between nodes of graph or ontology [Ivanova et al, 2013e]. The example with sample graph in previous chapter indicates that it is important to ensure possibility for multi-layer representation. To make experiment with real data, we will use the WordNet as ontology and its 45 types of relations (given by its files of different types) we store as 45 layers. To provide experiments in this case, we have realized program “OntoArM”. It is outlined in the Appendix A.

➢ *Definition of ontology*

For the purposes of this research, the next definition of ***ontology*** is appropriate:

Ontology*: a rigorous and exhaustive organization of some knowledge domain that is usually hierarchical and contains all the relevant entities and their relations* [WordNet, 2012].

People use the word **ontology** to mean different things, e.g. glossaries & data dictionaries, thesauruses & taxonomies, schemas & data models, and formal ontologies & inference. A **formal ontology** is a controlled vocabulary expressed in an ontology representation language. This language has a grammar for using vocabulary terms to express something meaningful within a specified domain of interest. The grammar contains formal constraints (e.g. specifies what it means to be a well-formed statement, assertion, query, etc.) on how a term in the ontology's controlled vocabulary can be used together.

People make commitments to use a specific controlled vocabulary or ontology for a domain of interest. Enforcement of ontology's grammar may be rigorous or lax. Frequently, the grammar for "light-weight" ontology is not completely specified, i.e. it has implicit rules that are not explicitly documented [Pidcock & Uschold, 2012].

The word **"ontology"** has been used to refer to all of the above things. When used in the AI/Knowledge_Representation community, it tends to refer to things that have a rich and formal logic-based language for specifying meaning of the terms. Both a thesaurus and taxonomy can be seen as having a simple language that could be given a grammar, although this is not normally done. Usually they are not formal, in the sense that there is no formal semantics given for the language. However, one can create a model in UML and a model in some formal ontology language and they can have identical meaning. It is thus not useful to say one is ontology and the other is not because one lacks a formal semantics. The truth is: there is a fuzzy line connecting these things [Pidcock & Uschold, 2012].

In 1992, Tom Gruber offers a formal description of concepts and relationships between them, called "*ontology*", which is a basis for communication between agents.

Ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where *Ontology* is a systematic account of *Existence* [Gruber, 1993a].

People, organizations and software systems must communicate between and among themselves. However, due to different needs and background contexts, there can be widely varying viewpoints and assumptions regarding what is essentially the same subject matter. Each uses different jargon; each may have differing, overlapping and/or mismatched concepts, structures and methods

[Uschold & Gruninger, 1996] stressed that "ontology is a *unifying framework* for the different viewpoints and serves as the basis for:

- *Communication* between people with different needs and viewpoints arising from their differing contexts;
- *Inter-Operability* among systems achieved by translating between different modeling methods, paradigms, languages and software tools;
- *System Engineering Benefits*: In particular:

- *Re-Usability*: the shared understanding is the basis for a formal encoding of the important entities, attributes, processes and their inter-relationships in the domain of interest. This formal representation may be (or become so by automatic translation) a re-usable and/or shared component in a software system;
- *Reliability*: A formal representation also makes possible the automation of consistency checking resulting in more reliable software;
- *Specification*: the shared understanding can assist the process of identifying requirements and defining a specification for an IT system. This is especially true when the requirements involve different groups using different terminology in the same domain, or multiple domains".

John Sowa notes that "*the art of ranking things in general and species is of no small importance and very much assists our judgment as well as our memory. (...) This helps one not merely to retain things, but also to find them. And those who have laid out all sorts of notions under certain headings or categories have done something very useful*" [Sowa, 2000].

The word "*ontology*" comes from the Greek "ontos" for being and "logos" for word. It is a relatively new term in the long history of philosophy, introduced by the 19th century German philosophers to distinguish the study of being as such from the study of various kinds of beings in the natural sciences. The more traditional term is Aristotle's word "category" (kathgoria), which he used for classifying anything that can be said or predicated about anything [Sowa, 2000a].

In the literature on artificial intelligence, the "ontology" is a term used to describe formally represented knowledge based on a conceptualization. It requires a description of a set of objects by corresponded concepts and relationships between these concepts (knowledge).

The word ontology can be used and has been used with very different meanings attached to it. Ironically, the ontology field suffered a lot from ambiguity. The Knowledge Engineering Community borrowed the term "Ontology" from the name of a branch of philosophy some 15 years ago and converted into an object: "ontology". In the mid-90s philosophers "took it back" and began to clean the definitions that had been adopted [Gandon, 2002].

Formally, the ontology consists of:

- Terms organized in taxonomy;
- Definitions of terms and attributes;
- Axioms and rules for inference.

The ontology formally can be described by the ordered triple [Palagin & Yakovlev, 2005; Gavrilova, 2001; Palagin, 2006; Guarino, 1998]:

$$O = <X,R,F>,$$

where X, R, F are a finite sets accordingly:

- X is a set of concepts (terms) from the subject area;
- R is a set of relationships between the elements of X;
- F is a set of functions to interpret the X and/or R.

Classifications of ontologies from different points of view and based on different principles are given in many publications [Bashmakov, 2005; Dobrov et al, 2009].

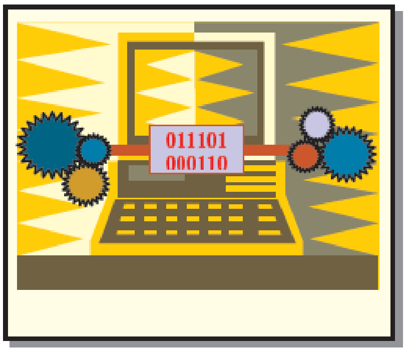
A formal classification scheme is given in [Guarino, 1998]. From its viewpoint: "*ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. Ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models*" [Guarino, 1998].

To illustrate the ***usefulness of ontologies*** let consider an example given by Prof. Assunción Gómez-Pérez from the Universidad Politechnika de Madrid, during a lecture on ECAI 98, presented in [Gandon, 2002].

The general problem is to formulate a query over a mass of information and get an answer as precise and relevant as possible. In her tutorial at ECAI 98, Prof. Pérez asked: "What is a pipe?"

Extending her example we can imagine three answers to this very same question (Table 28).

Table 28. Three notions behind the word "pipe"

		
A short narrow tube with a small container at one end, used for smoking e.g. tobacco.	A long tube made of metal or plastic that is used to carry water or oil or gas.	A temporary section of computer memory that can link two different computer processes.

We have one term and three concepts; *it is a case of ambiguity*. The contrary is one concept behind several terms, and *it is a case of synonyms* e.g. car, auto, automobile, motorcar, etc. These trivial cases poses a serious problem to computerize systems that are not able the see these difference or equivalence unless they have been made explicit to it.

Communication between people is based on an implicit consensus about concepts that are used. For example, when discussing a document, people involved in the discussion implicitly implied that they have a common conceptual consensus about the nature of the document. On a question about "text", the answer "knows" that "text" means the “document”.

This basic knowledge is lacking in information systems based only on usual terms and text search (by coincidence).

One possible approach is knowledge to be clearly formulated and presented in a logical structure that can be used by automated systems. This is exactly the purpose of ontology: to capture semantics and relationships of concepts we use, making it clear (explicit) and possibly encoded in symbolic systems so as to retrieve and exchange between different agents, which in particular can be computer systems.

In conclusion, we recall some of the generally accepted terms and their definitions that are used with ontologies [Gruber, 1993; Guarino & Giaretta, 1995; Bachimont, 2000; Gandon, 2002] (Table 29).

Table 29. Some commonly accepted concepts and definitions

Concepts	Definitions
Notion	- Something formed in the mind, a constituent of thought; - It is used to structure knowledge and perceptions of the world; - Principle, idea semantically evaluable and redeploy able.
A Concept	- Notion usually expressed by a term (or more generally by a sign); - The concept represents a group of objects or beings having shared characteristics "t" that enable us to recognize them a forming and belonging to this group.
A Relation	- Notion of an association or a link between concepts usually expressed by a term or a graphical convention (or more generally by a sign).
Ontology	- "That branch of philosophy which deals with the nature and the organization of reality"; - A branch of metaphysics which investigates the nature and essential properties and relations of all beings as such.
Formal Ontology	- The systematic, formal, axiomatic development of the logic of all forms and modes of being.
Conceptualization	- An intentional semantic structure which encodes the implicit rules constraining the structure of a piece of reality; - It also denotes the action of building such a structure.
An Ontology	- A logical theory which gives an explicit, partial account of a conceptualization; - The aim of ontologies is to define which primitives, provided with their associated semantics, are necessary for knowledge representation in a given context.
Ontological theory	- A set of formulas intended to be always true according to a certain conceptualization.
A Taxonomy	- A classification based on similarities.
A Partonomy	- A classification based on "part-of" relation.

➢ *Representing and operations with ontologies*

Traditionally, ontologies are built by highly trained knowledge engineers with the assistance of domain specialists. It's time-consuming and laborious task. Ontology tools also require users to be trained knowledge representation and predicate logic.

There are several approaches for representing ontologies. An example of such approach is using of XML. It is a popular markup language of metadata. With the development of the XML, different definitions of metadata have been proposed such as Dublin Core [Weibel et al, 1998] and ebXML [ebxml, 2012].

However, from the viewpoint of ontology, XML is not suited to describe the interrelationships of resources [Gunther, 1998]. Therefore, W3C has suggested the "Resource Description Framework" (RDF). There are several ontology languages like XML, RDF schema RDF(S), DAML+OIL and OWL. Many ontology tools have been developed for implementing metadata of ontology using these languages [Hertel et al, 2009].

Operations with ontologies are functions of the so called "middleware". What is called middleware is the layer implementing the access to the physical ontology data store.

Besides an inference mechanism, the access layer should provide functions for creating, querying and deleting data in the store.

While adding data requires parsing and ideally a validation of the incoming ontology sentences, querying the ontology store needs the implementation of some kind of query language as well as an interpretation and a translation of this query language into calls to the physical storage.

Another important feature of this layer is the possibility to export ontology data to a file for exchange with other systems [Hertel et al, 2009].

The *operations with several ontologies* are needed when one application uses multiple ontologies, especially when using modular design of ontologies or when we need to integrate with systems that use other ontologies.

We will summarize some of these operations. The terminology in this area is still not stable and different authors may use these terms in a bit shifted meaning, and so the terms may overlap, however, all of these operations are important for maintenance and integration of ontologies [Obitko, 2007].

- *Merging* of ontologies means creation of a new ontology by linking up the existing ones. Conventional requirement is that the new ontology contains all the knowledge from the original ontologies, however, this requirement does not have to be fully satisfied, since the original ontologies may not be together totally consistent. In that case the new ontology imports selected knowledge from the original ontologies so that the result is consistent. The merged ontology may introduce new concepts and relations that serve as a bridge between terms from the original ontologies;
- *Mapping* from ontology to another one is expressing of the way how to translate statements from ontology to the other one. Often it means translation between concepts and relations. In the simplest case it is mapping from one concept of the first ontology to one concept of the second ontology. It is not always possible to do such "one to one" mapping. Some information can be lost in the mapping. This is permissible; however mapping may not introduce any inconsistencies;
- *Alignment* is a process of mapping between ontologies in both directions whereas it is possible to modify original ontologies so that suitable translation exists (i.e. without losing information during mapping). Thus it is possible to add new concepts and

relations to ontologies that would form suitable equivalents for mapping. The specification of alignment is called articulation. Alignment, as well as mapping, may be partial only;

- *Refinement* is mapping from ontology A to another ontology B so that every concept of ontology A has equivalent in ontology B, however primitive concepts from ontology A may correspond to non-primitive (defined) concepts of ontology B. Refinement defines partial ordering of ontologies;
- *Unification* is aligning all of the concepts and relations in ontologies so that inference in ontology can be mapped to inference in other ontology and vice versa. Unification is usually made as refinement of ontologies in both directions;
- *Integration* is a process of looking for the same parts of two different ontologies A and B while developing new ontology C that allows to translate between ontologies A and B and so allows interoperability between two systems where one uses ontology A and the other uses ontology B. The new ontology C can replace ontologies A and B or can be used as an inter-lingua for translation between these two ontologies. Depending on the differences between A and B, new ontology C may not be needed and only translation between A and B is the result of integration. In other words, depending on the number of changes between ontologies A and B during development of ontology C the level of integration can range from alignment to unification;
- *Inheritance* means that ontology A inherits everything from ontology B. It inherits all concepts, relations and restrictions or axioms and there is no inconsistency introduced by additional knowledge contained in ontology A. This term is important for modular design of ontologies where an upper ontology describes general knowledge and lower application ontology adds knowledge needed only for the particular application. Inheritance defines partial ordering between ontologies.

Not all of these operations can be made for all ontologies [Obitko, 2007]. In general, these are very difficult tasks that are in general not solvable automatically, for example, because of:

- Undecidability when using very expressive logical languages;
- Insufficient specification of an ontology that is not enough to find similarities with another one.

Because of these reasons these tasks are usually made manually or semi-automatically, where a machine helps to find possible relations between elements from different ontologies, but the final confirmation of the relation is left on human. Human then decides based on:

- The natural language description of the ontology elements;
- The natural language names of the ontology elements and common sense.

➢ *Tools for developing ontologies*

The tools for developing ontologies allow users to define new concepts, relationships and instances, i.e. to create and/or expand existing ontologies. The ontology tools may contain some additional features such as graphical representation, information search and additional tuning [Noy &

Musen, 2002]. Such tools are, for instance, SWOOP [Kalyanpur et al, 2005], Top Braid composer [TBC, 2012], Internet Business Logic [IBL, 2012], OntoTrack [Liebig & Noppens, 2003] and IHMC Cmap Ontology Editor [Hayes et al, 2005].

"Chimaera" helps with merging ontologies. It provides suggestions for subsumption, disjointness or instance relationship. These suggestions are generated heuristically and are provided for an operator, so that he may choose which one will be actually used [Chimaera, 2012]. "PROMT" (or "SMART") system is a similar system that provides suggestions based on linguistic similarity, ontology structure and user actions. It points the user to possible effects of these changes [Promt, 2012].

In [OntoTools, 2012] more than 150 tools (ontology editors) are outlined. At the first glance, these tools may be classified on two groups – non commercial and commercial.

For instance, the first group include tools like Protégé [protégé, 2012], OilEd [Bechhofer et al, 2001], Apollo [Apollo, 2012], RDFedt [rdfedit, 2012] OntoLingua [Farquhar et al, 1996], OntoEdit [ontoprise, 2012; Sure et al, 2002; Sure et al, 2003], WebODE [Arpírez et al, 2001], KAON [Kaon, 2012], ICOM [ICOM, 2012], DOE [Bachimont, 2000; Bachimon et al, 2002; Troncy & Isaac, 2002; DOE, 2012], WebOnto [Webonto, 2012], and OntoIntegrator [Nevzorova et al, 2007; Nevzorova & Nevzorov, 2009; Nevzorova & Nevzorov, 2011].

Example of the commercial tools are Medius Visual Ontology Modeller [Polikoff, 2003; sandsoft, 2012], LinKFactory Workbench [Deray & Verheyden, 2003] and K-Infinity [Macris, 2004; OntoLex, 2012].

Many of the tools are closed systems. Therefore, it is not possible to evaluate the full functional capabilities. Thus, the choice of editor of ontologies for practical purposes depends of:

- Free distribution;
- Local use of the web interface;
- Extensibility of functional possibilities of the applications;
- Ability to include modules designed by the user.

The basic features, capabilities, advantages, disadvantages, and comparative analysis of available onto-editors are given in a number of meaningful overviews [Ovdei & Proskudina, 2004; Calvanese et al, 2007; Filatov et al, 2007]. Analysis of literary sources about ontoeditors shows that ontoeditor Protégé is closest to the listed requirements.

The instrumental systems for ontological engineering can be divided into three main groups [Ovdei & Proskudina, 2004]:

The first group includes tools for creating ontologies that provide:

- Maintenance of collaborative development and review;
- Creation of ontologies according to any methodology;
- Maintenance of reasoning.

The second group includes tools for [Noy & Musen, 1999]:

- Unification of ontologies;
- Discovering semantic relations between different ontologies;
- Alignment the ontologies by establishing links between them and allowing the aligned ontologies to reuse information from one another.

The third group includes tools for annotation of Web-based ontology resources.

Adding some new systems to survey of [Youn & McLeod, 2006], in Table 30 and Table 31 the basic and advanced features of several well-known ontological systems are outlined.

Table 30. Basic functions of the well-known ontological systems [Youn & McLeod, 2006]

	Import format	**Export format**	**Graph view**	**Consistency check**	**Multi-user**	**Web support**	**Merging**
Protégé [protégé, 2012]	XML, RDF(S), XML Schema	XML, RDF(S), XML Schema, FLogic, CLIPS, Java html	Via plug-ins like GraphViz and Jambalaya	Via pluggins like PAL and FaCT	Limited (Multi-user capability added to it in 2.0 version)	Via Protégé-OWL plug-in	Via Anchor-PROMPT plug-in
OilEd [Bechhofer et al, 2001]	RDF(S), OIL, DAML+OIL	RDF(S), OIL, DAML+OIL, SHIQ, dotty, html	No	Via FaCT	No	Very limited namespaces	No
Apollo [Apollo, 2012]	OCML, CLOS	OCML, CLOS	No	Yes	No	No	No
RDFedt [rdfedit, 2012]	RDF(S), OIL, DAML, SHOE	RDF(S), OIL, DAML, SHOE	No	Only checks writing mistakes	No	Via RSS (RDF Site Summary)	?
OntoLingua [Farquhar et al, 1996]	IDL, KIF	KIF, CLIPS, IDL, OKBC syntax, Prolog syntax	No	Via Chimaera	Via write-only locking, user access levels	Yes	?
OntoEdit (Free version) [ontoprise, 2012; Sure et al, 2002; Sure et al, 2003]	XML, RDF(S), FLogic and DAML+OIL	XML, RDF(S), FLogic and DAML+OIL	Yes	Yes	No	Yes	?

	Import format	Export format	Graph view	Consistency check	Multi-user	Web support	Merging
WebODE [Arpírez et al, 2001]	RDF(S), UML, DAML+OIL and OWL	RDF(S), UML, DAML+OIL, OWL, PROLOG, X-CARIN, Java/Jess	Form based graphical user interface	Yes	By synchronization, authentication and access restriction	Yes	Via ODEmerge
KAON [Kaon, 2012]	RDF(S)	RDF(S)	No	Yes	By concurrent access control	Via KAON portal	No
ICOM [ICOM, 2012]	XML , UML	XML, UML	Yes	Via FaCT	No	No	With inter-ontology mapping
DOE [Bachimont, 2000; Bachimon et al, 2002; Troncy & Isaac, 2002; DOE, 2012]	XSLT, RDF(S), OIL, DAML+OIL, OWL and CGXML	XSLT, RDF(S), OIL, DAML+OIL, OWL and CGXML	No	Via type inheritance and detection of cycles in hierarchies	No	Load ontology via URL	No
WebOnto [Webonto, 2012]	OCML	OCML, GXL, RDF(S) and OIL	Yes	Yes	With global write-only locking	Web based	?
OntoIntegrator [Nevzorova & Nevzorov, 2011]	own format	own format	Yes	?	No	No	?
Medius VOM [Polikoff, 2003; sandsoft, 2012]	XML Schema, RDF and DAML+OIL	XML Schema, RDF and DAML+OIL	UML diagrams via Rose	With a set of ontology authoring wizards	Network based	Via read-only browser from Rose	Limited (only native Rose model)
LinKFactory [Deray & Verheyden, 2003]	XML, RDF(S), DAML+OIL and OWL	XML, RDF(S), DAML+OIL, OWL and html	No	Yes	Yes	Yes	Yes
K-Infinity [Macris, 2004; OntoLex, 2012]	RDF	RDF	With Graph editor	Yes	Network based	No	?

Table 31. Additional functions of the well-known ontological systems [Youn & McLeod, 2006]

	Collaborative working	**Ontology library**	**Inference engine**	**Exception handling**	**Ontology storage**	**Extensibility**	**Availability**
Protégé [protégé, 2012]	No	Yes	With PAL	No	File & DBMS (JDBC)	Via plug-ins	Free
OilEd [Bechhofer et al, 2001]	No	Yes	With FaCT	No	File	No	Free
Apollo [Apollo, 2012]	No	Yes	No	No	Files	Via plug-ins	Free
RDFedt [rdfedit, 2012]	No	No	No	Yes	Files	No	Free
OntoLingua [Farquhar et al, 1996]	Yes	Yes	No	No	Files	No	Free
OntoEdit (Free version) [ontoprise, 2012; Sure et al, 2002; Sure et al, 2003]	No	No	No	No	File	Via plug-ins	Free
WebODE [Arpírez et al, 2001]	Yes	No	Prolog	No	DBMS (JDBC)	Via plug-ins	Free
KAON [Kaon, 2012]	?	Yes	Yes	No	?	No	Free
ICOM [ICOM, 2012]	No	?	Yes	No	DBMS	Yes	Free
DOE [Bachimont, 2000; Bachimon et al, 2002; Troncy & Isaac, 2002; DOE, 2012]	No	No	Yes	No	File	No	Free
WebOnto [Webonto, 2012]	Yes	Yes	Yes	No	File	No	Free web access
OntoIntegrator [Nevzorova & Nevzorov, 2011]	No	Yes	No	?	relational database	No	Free

	Collaborative working	Ontology library	Inference engine	Exception handling	Ontology storage	Extensibility	Availability
Medius VOM [Polikoff, 2003; sandsoft, 2012]	Yes	Yes (IEEE SUO)	Yes	?	?	Yes	Commercial
LinKFactory [Deray & Verheyden, 2003]	Yes	Yes	Yes	No	DBMS	Yes	Commercial
K-Infinity [Macris, 2004; OntoLex, 2012]	Yes	Yes	Yes	?	DBMS	No	Commercial

General disadvantages of the outlined instrumental systems are:

- Lack of automatic (or automated) procedures for forming components of ontologies;
- User interface based only on English, which does not permit using of other languages, such as Bulgarian, Russian, Greek, etc.;
- The structure of concepts may be built by only one type of relationships;
- For most commonly available ontological systems it is impossible to work with ontologies of large volume (e.g. OntoEdit free – up to 50 concepts);
- Many tools store the ontologies in text files, which limits the speed of access to ontologies;
- Some functions are not available in the free versions of the tools;
- User documentation is not good enough.

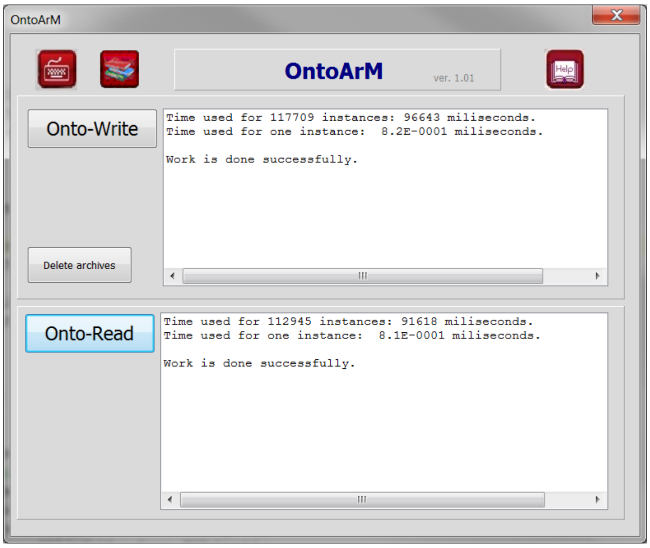
The above shortcomings of popular English language ontological tools exist in similar instruments from Russian segment, in particular, "Multi-layer ontology editor" [Artemieva & Reshtanenko, 2008], "OntoEditor+" [Nevzorova et al, 2004] and others.

➢ *Experiment to store ontology by NL-addressing*

Comparative analysis of the tools shows that all systems use finished products for data storing, which are limited to text files or relational databases. Both approaches for storing do not meet specific structures of the ontologies. This necessitates the development of new tools for storing ontologies.

Storing graphs and ontologies has one important aspect – the layers which correspond to types of relations between nodes of graph or ontology. The example with sample graph in previous chapter indicates that it is important to ensure possibility for multi-layer representation.

To make experiment with real data, we use the WordNet as ontology and its 45 types of relations (given by its files of different types, see Table 26) we store as 45 layers. For experiments in this case, we have realized a program called “OntoArM”. It is outlined in the Appendix A. A screenshot from the OntoArM with results for the case with 45 layers is given in Figure 46.



```
Time used for 117709 instances: 96643 miliseconds.
Time used for one instance:  8.2E-0001 miliseconds.

Work is done successfully.
```

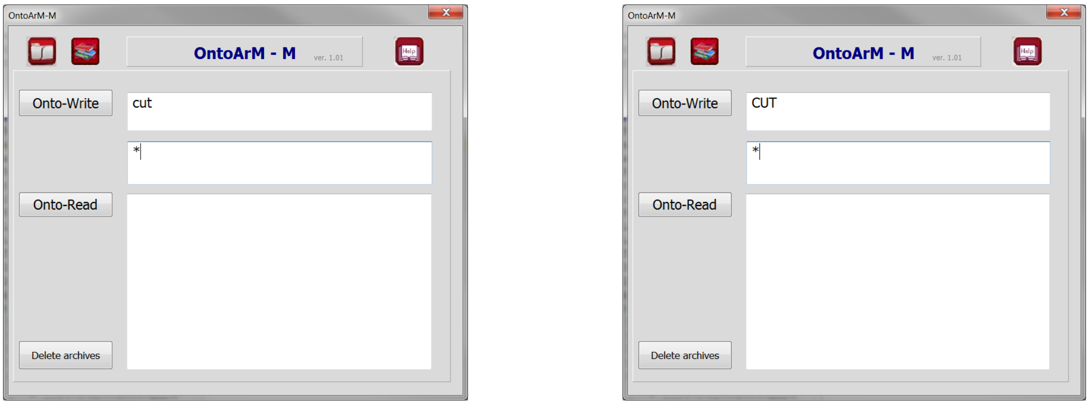
```
Time used for 112945 instances: 91618 miliseconds.
Time used for one instance:  8.1E-0001 miliseconds.

Work is done successfully.
```

Figure 46. OntoArM results for the case of WordNet with 45 layers

The NL-addressing is case sensitive. The words “cut” and “CUT” are absolutely different as NL-addresses. Because of this, for words “cut” and “CUT” we have to use separate queries (Figure 47). Of course, it is easy to program the system automatically to use both capital and small letters. This is a problem to be solved at the middleware level.

In Figure 48 the OntoArM report for the query “cut; *” is shown.

The information on Figure 48 is shown with “no word wrap” option of WordPad program. In Table 32, the same report is given in whole for both queries (cut; *) and (CUT; *). The definitions are shown “as_is” in the lexicographer files, i.e. the access method does not convert the information to any other style and stores and extracts the information “as_is”.

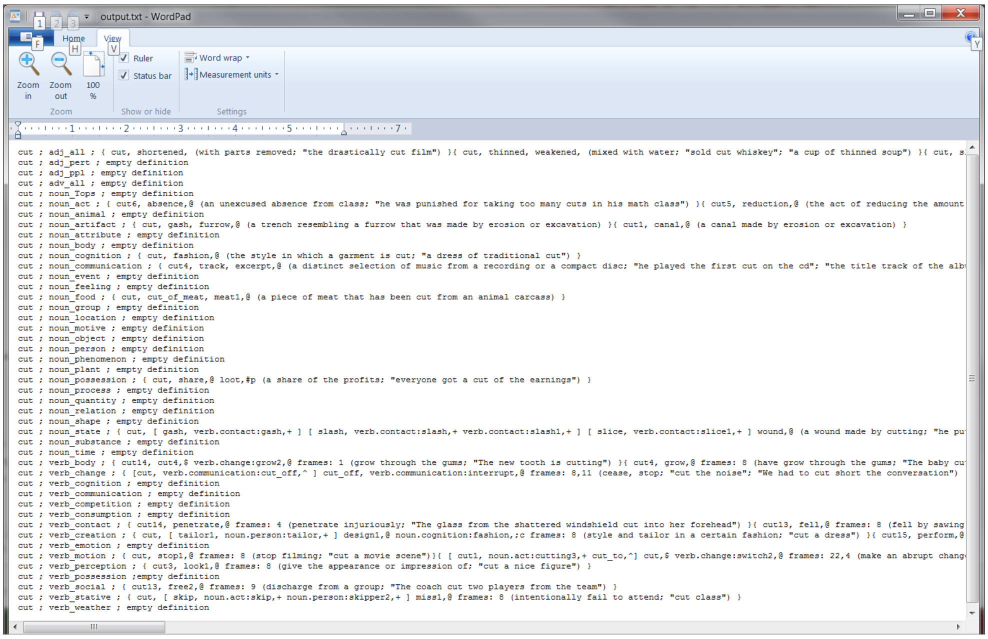


cut
*

CUT
*

a) small letters b) capital letters

Figure 47. OntoArM panel for manual querying words cut and CUT



```
cut ; adj_all ; { cut, shortened, (with parts removed; "the drastically cut film") }{ cut, thinned, weakened, (mixed with water; "sold cut whiskey"; "a cup of thinned soup") }{ cut, s
cut ; adj_pert ; empty definition
cut ; adj_ppl ; empty definition
cut ; adv_all ; empty definition
cut ; noun_Tops ; empty definition
cut ; noun_act ; { cut6, absence,@ (an unexcused absence from class; "he was punished for taking too many cuts in his math class") }{ cut5, reduction,@ (the act of reducing the amount
cut ; noun_animal ; empty definition
cut ; noun_artifact ; { cut, gash, furrow,@ (a trench resembling a furrow that was made by erosion or excavation) }{ cut1, canal,@ (a canal made by erosion or excavation) }
cut ; noun_attribute ; empty definition
cut ; noun_body ; empty definition
cut ; noun_cognition ; { cut, fashion,@ (the style in which a garment is cut; "a dress of traditional cut") }
cut ; noun_communication ; { cut4, track, excerpt,@ (a distinct selection of music from a recording or a compact disc; "he played the first cut on the cd"; "the title track of the alb
cut ; noun_event ; empty definition
cut ; noun_feeling ; empty definition
cut ; noun_food ; { cut, cut_of_meat, meat1,@ (a piece of meat that has been cut from an animal carcass) }
cut ; noun_group ; empty definition
cut ; noun_location ; empty definition
cut ; noun_motive ; empty definition
cut ; noun_object ; empty definition
cut ; noun_person ; empty definition
cut ; noun_phenomenon ; empty definition
cut ; noun_plant ; empty definition
cut ; noun_possession ; { cut, share,@ loot,#p (a share of the profits; "everyone got a cut of the earnings") }
cut ; noun_process ; empty definition
cut ; noun_quantity ; empty definition
cut ; noun_relation ; empty definition
cut ; noun_shape ; empty definition
cut ; noun_state ; { cut, [ gash, verb.contact:gash,+ ] [ slash, verb.contact:slash,+ verb.contact:slash1,+ ] [ slice, verb.contact:slice1,+ ] wound,@ (a wound made by cutting; "he pu
cut ; noun_substance ; empty definition
cut ; noun_time ; empty definition
cut ; verb_body ; { cut14, cut4,$ verb.change:grow2,@ frames: 1 (grow through the gums; "The new tooth is cutting") }{ cut4, grow,@ frames: 8 (have grow through the gums; "The baby cu
cut ; verb_change ; { [cut, verb.communication:cut_off,^ ] cut_off, verb.communication:interrupt,@ frames: 8,11 (cease, stop; "cut the noise"; "We had to cut short the conversation")
cut ; verb_cognition ; empty definition
cut ; verb_communication ; empty definition
cut ; verb_competition ; empty definition
cut ; verb_consumption ; empty definition
cut ; verb_contact ; { cut14, penetrate,@ frames: 4 (penetrate injuriously; "The glass from the shattered windshield cut into her forehead") }{ cut13, fell,@ frames: 8 (fell by sawing
cut ; verb_creation ; { cut, [ tailor1, noun.person:tailor,+ ] design1,@ noun.cognition:fashion,;c frames: 8 (style and tailor in a certain fashion; "cut a dress") }{ cut15, perform,@
cut ; verb_emotion ; empty definition
cut ; verb_motion ; { cut, stop1,@ frames: 8 (stop filming; "cut a movie scene")}{ [ cut1, noun.act:cutting3,+ cut_to,^] cut,$ verb.change:switch2,@ frames: 22,4 (make an abrupt chang
cut ; verb_perception ; { cut3, look1,@ frames: 8 (give the appearance or impression of; "cut a nice figure") }
cut ; verb_possession ;empty definition
cut ; verb_social ; { cut13, free2,@ frames: 9 (discharge from a group; "The coach cut two players from the team") }
cut ; verb_stative ; { cut, [ skip, noun.act:skip,+ noun.person:skipper2,+ ] miss1,@ frames: 8 (intentionally fail to attend; "cut class") }
cut ; verb_weather ; empty definition
```

Figure 48. OntoArM report to query from Figure 47 a)

Table 32. Report of the queries from Figure 47 a) and b) for all 45 layers of WordNet and for both queries (cut;*) and (CUT;*)

No.	layer	definition
1	cut ; **adj_all** ;	{ cut, shortened, (with parts removed; "the drastically cut film") } { cut, thinned, weakened, (mixed with water; "sold cut whiskey"; "a cup of thinned soup") } { cut, slashed, ((used of rates or prices) reduced usually sharply; "the slashed prices attracted buyers") } { cut, emasculated, gelded, ((of a male animal) having the testicles removed; "a cut horse") }
	CUT ; **adj_all ;**	{ [CUT1, UNCUT1,!] (separated into parts or laid open or penetrated with a sharp edge or instrument; "the cut surface was mottled"; "cut tobacco"; "blood from his cut forehead"; "bandages on her cut wrists") } { [CUT2, UNCUT2,!] ((of pages of a book) having the folds of the leaves trimmed or slit; "the cut pages of the book") } { [CUT3, UNCUT3,!] (fashioned or shaped by cutting; "a well-cut suit"; "cut diamonds"; "cut velvet") }
2	cut ; **adj_pert** ;	empty definition
3	cut ; **adj_ppl** ;	empty definition
4	cut ; **adv_all** ;	empty definition
5	cut ; **noun_Tops** ;	empty definition

No.	layer	definition
6	cut ; **noun_act** ;	{ cut6, absence,@ (an unexcused absence from class; "he was punished for taking too many cuts in his math class") } { cut5, reduction,@ (the act of reducing the amount or number; "the mayor proposed extensive cuts in the city budget") } { cut, [cutting, verb.creation:cut11,+] cutting_off1, shortening,@ (the act of shortening something by chopping off the ends; "the barber gave him a good cut") } { cut1, [cutting1, verb.contact:cut10,+ verb.contact:cut,+] division,@ (the act of cutting something into parts; "his cuts were skillful"; "his cutting of the cake made a terrible mess") } { cut2, [cutting2, verb.contact:cut10,+] opening2,@ (the act of penetrating or opening open with a sharp edge; "his cut in the lining revealed the hidden jewels") } { cut9, [cutting9, verb.contact:cut5,+] division,@ card_game,#p (the division of a deck of cards before dealing; "he insisted that we give him the last cut before every deal"; "the cutting of the cards soon became a ritual") } { cut8, [undercut, verb.contact:undercut,+] stroke,@ tennis,;c badminton,;c squash,;c ((sports) a stroke that puts reverse spin on the ball; "cuts do not bother a good tennis player") }
7	cut ; **noun_animal** ;	empty definition
8	cut ; **noun_artifact** ;	{ cut, gash, furrow,@ (a trench resembling a furrow that was made by erosion or excavation) } { cut1, canal,@ (a canal made by erosion or excavation) }
9	cut ; **noun_attribute** ;	empty definition
10	cut ; **noun_body** ;	empty definition
11	cut ; **noun_cognition** ;	{ cut, fashion,@ (the style in which a garment is cut; "a dress of traditional cut") }
12	cut ; **noun_communication** ;	{ cut4, track, excerpt,@ (a distinct selection of music from a recording or a compact disc; "he played the first cut on the cd"; "the title track of the album") } { cut, transition,@ ((film) an immediate transition from one shot to the next; "the cut from the accident scene to the hospital seemed too abrupt") }
13	cut ; **noun_event** ;	empty definition
14	cut ; **noun_feeling** ;	empty definition
15	cut ; **noun_food** ;	{ cut, cut_of_meat, meat1,@ (a piece of meat that has been cut from an animal carcass) }
16	cut ; **noun_group** ;	empty definition

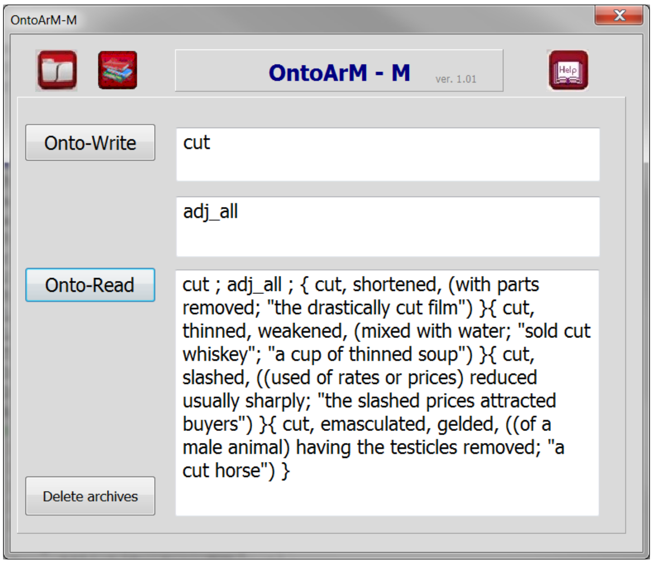
No.	layer	definition
17	cut ; **noun_location** ;	empty definition
18	cut ; **noun_motive** ;	empty definition
19	cut ; **noun_object** ;	empty definition
20	cut ; **noun_person** ;	empty definition
21	cut ; **noun_phenomenon** ;	empty definition
22	cut ; **noun_plant** ;	empty definition
23	cut ; **noun_possession** ;	{ cut, share,@ loot,#p (a share of the profits; "everyone got a cut of the earnings") }
24	cut ; **noun_process** ;	empty definition
25	cut ; **noun_quantity** ;	empty definition
26	cut ; **noun_relation** ;	empty definition
27	cut ; **noun_shape** ;	empty definition
28	cut ; **noun_state** ;	{ cut, [gash, verb.contact:gash,+] [slash, verb.contact:slash,+ verb.contact:slash1,+] [slice, verb.contact:slice1,+] wound,@ (a wound made by cutting; "he put a bandage over the cut") } { cut1, gradation,@ (a step on some scale; "he is a cut above the rest") }
29	cut ; **noun_substance** ;	empty definition
30	cut ; **noun_time** ;	empty definition
31	cut ; **verb_body** ;	{ cut14, cut4,$ verb.change:grow2,@ frames: 1 (grow through the gums; "The new tooth is cutting") }{ cut4, grow,@ frames: 8 (have grow through the gums; "The baby cut a tooth") }
32	cut ; **verb_change** ;	{ [cut, verb.communication:cut_off,^] cut_off, verb.communication:interrupt,@ frames: 8,11 (cease, stop; "cut the noise"; "We had to cut short the conversation") } { cut12, cut6,$ decrease1,@ frames: 4 (have a reducing effect; "This cuts into my earnings")} { cut15, dissolve1,@ frames: 11 (dissolve by breaking down the fat of; "soap cuts grease") } { [cut2, cut_back,^ cut_back1,^ cut_out,^] prune, rationalize, rationalise, eliminate1,@ frames: 8 (weed out unwanted or unnecessary things; "We had to lose weight, so we cut the sugar from our diet") } { cut14, shorten9,@ frames: 8 (shorten as if by severing the edges or ends of; "cut my hair") }
33	cut ; **verb_cognition** ;	empty definition
34	cut ; **verb_communication** ;	empty definition
35	cut ; **verb_competition** ;	empty definition
36	cut ; **verb_consumption** ;	empty definition

No.	**layer**	**definition**
37	cut ; **verb_contact** ;	{ cut14, penetrate,@ frames: 4 (penetrate injuriously; "The glass from the shattered windshield cut into her forehead") } { cut13, fell,@ frames: 8 (fell by sawing; hew; "The Vietnamese cut a lot of timber while they occupied Cambodia") } { cut15, reap,@ frames: 8 (reap or harvest; "cut grain") } { cut7, hit,@ noun.act:sport,;c frames: 8 (hit (a ball) with a spin so that it turns in the opposite direction; "cut a Ping-Pong ball") } { [cut, noun.person:cutter,+ noun.artifact:cutter,+ noun.act:cutting1,+ cut_away,^ cut_out2,^ cut_up,^ cut_into1,^ cut_off2,^ cut_out1,^] separate1,@ frames: 8,11 (separate with or as if with an instrument; "Cut the rope")} { [cut5, noun.act:cutting9,+] shuffle,@ frames: 2,8 (divide a deck of cards at random into two parts to make selection difficult; "Wayne cut"; "She cut the deck for a long time") } { [cut10, noun.act:cutting2,+ noun.act:cutting1,+] frames: 22 (make an incision or separation; "cut along the dotted line") } { cut11, cut10,$ verb.stative:be3,@ frames: 1 (allow incision or separation; "This bread cuts easily") } { cut12, function,@ frames: 1 (function as a cutting instrument; "This knife cuts well") }
38	cut ; **verb_creation** ;	{ cut, [tailor1, noun.person:tailor,+] design1,@ noun.cognition:fashion,;c frames: 8 (style and tailor in a certain fashion; "cut a dress") } { cut15, perform,@ frames: 8 (perform or carry out; "cut a caper") } { [cut11, noun.act:cutting5,+ noun.act:cutting,+] create,@ frames: 8,11 (form or shape by cutting or incising; "cut paper dolls") } { cut1, cut11,$ create,@ frames: 8,11 (form by probing, penetrating, or digging; "cut a hole"; "cut trenches"; "The sweat cut little rivulets into her face") } { cut6, burn5, create3,@ frames: 8 (create by duplicating data; "cut a disk"; "burn a CD")} { cut4, cut6,$ verb.communication:record1,@ frames: 8 (record a performance on (a medium); "cut a record") } { cut5, cut4,$ verb.communication:record1,@ frames: 8 (make a recording of; "cut the songs"; "She cut all of her major titles again") }
39	cut ; **verb_emotion** ;	empty definition
40	cut ; **verb_motion** ;	{ cut, stop1,@ frames: 8 (stop filming; "cut a movie scene")} { [cut1, noun.act:cutting3,+ cut_to,^] cut,$ verb.change:switch2,@

No.	layer	definition
		frames: 22,4 (make an abrupt change of image or sound; "cut from one scene to another") } { cut12, pass_through,@ frames: 8,9 (pass through or across; "The boat cut the water") } { cut13, cut12,$ pass,@ frames: 22 (pass directly and often in haste; "We cut through the neighbor's yard to get home sooner") } { cut15, move,@ noun.act:boxing,;c frames: 22 (move (one's fist); "his opponent cut upward toward his chin") }
41	cut ; **verb_perception** ;	{ cut3, look1,@ frames: 8 (give the appearance or impression of; "cut a nice figure") }
42	cut ; **verb_possession** ;	empty definition
43	cut ; **verb_social** ;	{ cut13, free2,@ frames: 9 (discharge from a group; "The coach cut two players from the team") }
44	cut ; **verb_stative** ;	{ cut, [skip, noun.act:skip,+ noun.person:skipper2,+] miss1,@ frames: 8 (intentionally fail to attend; "cut class") }
45	cut ; **verb_weather** ;	empty definition

To update the content of a definition one may use form for manual work (Figure 49) and to follow the next steps:

- To enter the concept and layer;
- To activate reading current definition pressing the RDF-Read button;
- To update content of definition on screen;
- To press RDF-Write button to store new variant of definition in the correspond archive.



cut

adj_all

cut ; adj_all ; { cut, shortened, (with parts removed; "the drastically cut film") }{ cut, thinned, weakened, (mixed with water; "sold cut whiskey"; "a cup of thinned soup") }{ cut, slashed, ((used of rates or prices) reduced usually sharply; "the slashed prices attracted buyers") }{ cut, emasculated, gelded, ((of a male animal) having the testicles removed; "a cut horse") }

Figure 49. OntoArM panel for manual updating definitions

The results from experiments for storing WordNet as ontology with 45 layers are given in Table 33.

Table 33. Experimental data for storing WordNet as ontology

number of layers	operation	number of instances	total time in milliseconds	average time (ms) for one instance
45	writing	117 709	96 643	**0.82**
45	reading	112 945	91 618	**0.81**
45	work memory: 538 408 KB; permanent archive: 17 013 KB source text in 45 files - not compressed: 16 338 KB; compressed by WinZip: 4937 KB			

The work memory for storing hash tables and theirs containers was 538 408 KB. To analyze work of system, the work memory was chosen to be in a file in the external memory. In further realizations of OntoArM, to accelerate the speed and to reduce used disk space, work memory may be realized as part of main memory (as dynamically allocated memory or as file mapped in memory).

After finishing the work, occupied disk memory for compressed permanent archives is 17 013 KB, i.e. in this case the NL-indexing takes 12 076 KB additional compressed memory (the 45 sequential files with initial data occupy 16 338 KB, and compressed by WinZip they take 4 937 KB).

The difference in the numbers of instances in Table 27 and Table 33 is due to removing the equal instances and service information from input files when we use WordNet as thesaurus, stored as one layer archive. For instance, the word "cut" has many instances and when we work with one layer it must be written at least two NL-addresses – using small and capital letters: "cut" and "CUT".

After updating, no recompiling of the data base is needed. For less than one millisecond after entering new data, the information is ready for using.

➢ *Comparing OntoArM and WordArM programs*

To compare WordArM and OntoArM programs, we made experiment with both programs to store WordNet data as one layer ontology. The results are given in the Table 34.

Table 34. Results for speed of WordArM and OntoArM programs

	WordNet as one layer ontology		WordNet as 45 layers ontology
operation	WordArM average time (ms) for one instance	OntoArM average time (ms) for one instance	OntoArM average time (ms) for one instance
writing	**0.86**	**1.00**	**0.82**
reading	**0.78**	**0.95**	**0.81**

From results in Table 34 we may conclude that OntoArM, working in parallel with 45 layers ensures more high speed than working with one layer. This is due to available separate buffering for each layer and small size of each of 45 archives (in the case of one layer all information is written in one big file).

In addition, OntoArM is slower than WordArM in the case of one layer because of existing operations for control of layers in OntoArM, which is not needed and not realized in WordArM.

➢ *What gain and loss using NL-Addressing for storing ontologies?*

The conclusions are the same as for the dictionaries and thesauruses.

The loss is additional memory for storing internal hash structures. But the same if no great losses we will have if we will build balanced search threes or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published.

The benefit is in two main achievements:

- High speed for storing and accessing the information;
- The possibility to access information immediately after storing without recompilation the database and rebuilding indexes.

➢ *Conclusion of chapter 5*

In this chapter we have presented several experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of structured datasets.

Firstly we introduced the idea of knowledge representation. Further in the chapter we discussed three main experiments - for NL-storing of dictionaries, thesauruses, and ontologies.

Presentations of every experiment started with introductory part aimed to give working definition and to outline state of the art in storing concrete structures.

The explanation of the experiments begins with the easiest case – storing dictionaries. Analyzing results from the experiment with a real dictionary data we may conclude that it is possible to use NL-addressing for storing such information.

Next experiment was aimed to answer to question: "What we gain and loss using NL-Addressing for storing thesauruses?"

Analyzing results from the experiment we point that the loss is additional memory for storing hash structures which serve NL-addressing. But the same if no great losses we will have if we will build balanced search trees or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published. The benefit is in two main achievements:

(1) High speed for storing and accessing the information.

(2) The possibility to access the information immediately after storing without recompilation the database and rebuilding the indexes.

The third experiment considered the complex graph structures such as ontologies. The presented survey of the state of the art in this area has shown that main models for storing ontologies are files and relational databases.

Our experiment confirmed the conclusion about losses and benefits from using NL-addressing given above for thesauruses. The same is valid for more complex structures.

Here we have to note that for static structured datasets it is more convenient to use standard utilities and complicated indexes. NL-addressing is suitable for dynamic processes of creating and further development of structured datasets due to avoiding recompilation of the database index structures and high speed access to every data element.

The goal of the experiments with different small datasets, like dictionary, thesaurus or ontology, was to discover regularities in the NL-addressing realization. Analyzing Table 25, Table 27, and Table 33 we may see the main two regularities of storing time using NL-addressing:

- *It depends on number of elements in the instances;*
- *It not depends on number of instances in datasets.*

6 Experiments for NL-storing of middle-size and large RDF-datasets

Abstract

In this chapter we will present results from series of experiments which are needed to estimate the storing time of NL-addressing for middle-size and large RDF-datasets.

The experiments for NL-storing of middle-size and large RDF-datasets are aimed to estimate possible further development of NL-ArM. We assume that its "software growth" will be done in the same grade as one of the known systems like Virtuoso, Jena, and Sesame. We will analyze what will be the place of NL-ArM in this environment. Our hypothesis is that NL-addressing will have good performance.

Chapter will start with describing the experimental storing models and algorithm used in this research. Further an estimation of experimental systems will be provided to make different configurations comparable. Special proportionality constants for hardware and software will be proposed. Using proportionality constants, experiments with middle-size and large datasets became comparable.

Experiments will be provided with both real and artificial datasets. Experimental results will be systematized in corresponded tables. For easy reading visualization by histograms will be given.

6.1 Experimental storing model

Our first experiments in this research were to realize a small multi-language dictionary. In this case, NL-storing model is simple because the one-one correspondence "word - definition". The storing models for thesauruses are more complicated due to existing more than one corresponding definitions for a given word. Because of this, we outlined and analyzed the storing model of WordNet thesaurus [WordNet, 2012]. The idea was to use NL-addressing to realize the WordNet lexical database and this way to avoid recompilation of its database after every update. The program used for the experiments was "WordArM" (see Appendix A).

The next step of experiments was storing graphs and ontologies which have one important aspect – the layers which correspond to types of relations between nodes of graph or ontology [Ivanova et al, 2013e]. To make experiments with real data, we have used the WordNet as ontology and its 45 types of relations (given by its files of different types) we have stored as 45 layers. To provide experiments in this case, we used program "OntoArM" (see Appendix A).

The goal of the experiments with different small datasets, like dictionary, thesaurus or ontology, was to discover regularities in the NL-addressing realization. More concretely, two regularities of time for storing by using NL-addressing were discovered:

- It depends on number of elements in the instances;
- It not depends on number of instances in datasets.

The experiments for NL-storing of middle-size and large RDF-datasets have another goal. We are interested to estimate possible further development of NL-ArM. We assume that its "software growth" will be done in the same grade as one of the known systems like Virtuoso, Jena, and Sesame. We will analyze what will be the place of NL-ArM in this environment. Our hypothesis is that NL-addressing will have good performance.

Now, our next step is to provide experiments to use NL-addressing for storing middle-size and large RDF-datasets. We have realized experimental program RDFArM (see Appendix A) for storing RDF-datasets.

Let remember from Chapter 1 that the primary goal of Resource Description Framework (RDF) is to handle *non regular or semi-structured data* [Muys, 2007]. RDF provides a general method to decompose any information into pieces called triples [Briggs, 2012]:

- Each triple is of the form "Subject", "Predicate", "Object";
- Subject and Object are the names for two things in the world. Predicate is the relationship between them;
- Subject, Predicate, Object may be given as URI's (stand-ins for things in the real world);
- Object can additionally be raw text.

The power of RDF relies on the flexibility in representing arbitrary structure without a priori schemas. Each edge in the graph is a single fact, a single statement, similar to the relationship between a single cell in a relational table and its row's primary key. RDF offers the ability to specify concepts and link them together into a graph of data [Faye et al, 2012].

Middle-size RDF-datasets are those which contain from several hundred thousand up to 10 millions of RDF-instances.

Large RDF-datasets may contain 50, 100, or more millions, as well as billions or trillions of RDF-instances.

Unfortunately, due to financial limitations, we have no proper hardware for making comprehensive program experiments with many gigabytes of source data. Because of this, storing of very large RDF structures by NL- addressing is planned as future work. Here we will outline partial experiments with limited quantity from several hundred thousand up to 100 millions of triples or quadruples due to small main and hard disk memory as well as computational possibilities for processing very large datasets both by our program and by other installations.

In the same time, due to constant complexity of NL-addressing, we may extrapolate the results and provide preliminary comparison with published benchmarks of known RDF-stores using corresponded normalizing the results.

➢ *Experimental datasets*

We will provide experiments with middle-size RDF-datasets, based on selected real middle-size datasets from DBpedia's homepages [DBpedia, 2007a; DBpedia, 2007b] and artificial middle-size datasets from Berlin SPARQL Bench Mark (BSBM) [Bizer & Schultz, 2008; Bizer & Schultz, 2009]. Real large datasets are taken from DBpedia's homepages [DBpedia, 2007c], BSBM [Bizer & Schultz, 2009] and Billion Triple Challenge (BTC) 2012 [BTC, 2012]. Artificial large datasets are taken from Berlin SPARQL Bench Mark (BSBM) [Bizer & Schultz, 2009].

The reason to make this choice is that we want to provide experiments with both real and artificial data. The artificial datasets like BSBM [BSBM, 2012] contain standard artificially generated data and it is possible to adapt the software to have best results just for this kind of data. The DBpedia and BTC datasets were crawled using several seed sets collected from multiple real sources. Data in BTC datasets are encoded in N-Quads format.

The N-Quads is a format that extends N-triples with context. Each triple in N-Quad's 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 notion of provenance is essential when integrating data from different sources. Therefore, modern RDF repositories store "subject-predicate-object-context" quadruples, where the context typically denotes the provenance of a given statement. The SPARQL query language can query such RDF datasets or entire collections of RDF graphs [SPARQL, 2013]. The context element is also sometimes used to track a dimension such as time or geographic location.

Applications of N-Quads include:

- Exchange of RDF datasets between RDF repositories, where the fourth element is the URI of the graph that contains each statement;
- Exchange of collections of RDF documents, where the fourth element is the HTTP URI from which the document was originally retrieved;
- Publishing of complex RDF knowledge bases, where the original provenance of each statement has to be kept intact.

N-Quads inherit the practical advantages of N-Triples:

- Simple parsing;
- Succinctness compared to alternatives such as reification or multi-document archives;
- Effective streaming and processing with line-based tools.

Let see a quadruple from BTC extracted from the data set [datahub_data0, 2012]:

<http://nektar.oszk.hu/resource/auth/magyar_irodalom><http://www.w3.org/2004/02/skos/core#narrower><http://nektar.oszk.hu/resource/auth/hungarikum><http://nektar.oszk.hu/data/auth/magyar_irodalom>.

In this quadruple:

<subject>	=	<http://nektar.oszk.hu/resource/auth/magyar_irodalom>
<predicate>	=	<http://www.w3.org/2004/02/skos/core#narrower>
<object>	=	<http://nektar.oszk.hu/resource/auth/hungarikum>
<context>	=	<http://nektar.oszk.hu/data/auth/magyar_irodalom>

- ***Storing models***

The storing models we will use are multi-layer. First two models are for N-Triples and the third is for N-Quads format.

In the *first storing model*, values of *Predicates* may be names of the layers (archives), the *Subjects* will be the NL-addresses, and only *Objects* will be saved.

In the *second storing model*, values of *Subjects* and *Predicates* will be the NL-addresses in the same archive, and only *Objects* will be saved using couple (Subject, Relation) as co-ordinates of container where the Object will be saved.

In the *third storing model*, values of *Subjects* and *Predicates* will be the NL-addresses but *Objects* and *Context*s will be saved in different archives using couple (Subject, Relation) as co-ordinates.

6.2 Experimental storing algorithm

All storing models pointed above may be generalized in one common model where *Subjects, Predicates, Objects,* and *Context*s are numbered separately and these numbers are used to construct storing co-ordinates. For triple datasets the elements which contain context have to be omitted.

The experimental storing algorithm is illustrated on Figure 50. Main idea is to use NL-addressing for quick unique numbering of elements of triples/quadruples and after that to use these numbers as co-ordinates for storing information in the archives. In this case we have two kinds of archives (1) archive of counters and (2) archive of values.

In Figure 50 we illustrated storing of RDF – triple

(beer, is, proof that...)

First we assign a number to subject – in this case: “beer”.

The same we do for the relation – in this case: “is”.

And after that we used these numbers as coordinates of the object - in this case: “proof that ...”.

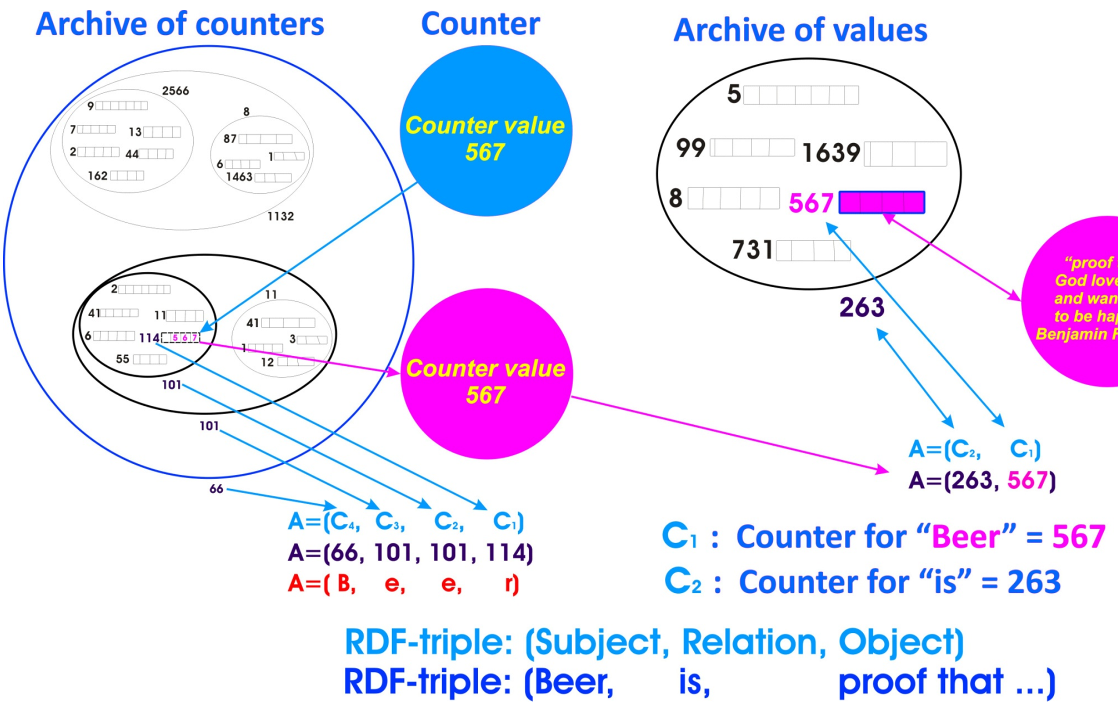

Archive of counters
Counter
Archive of values
Counter value 567
Counter value 567
"proof that God loves us and wants us to be happy." Benjamin Franklin
263
A=(C₂, C₁)
A=(263, 567)
A=(C₄, C₃, C₂, C₁)
A=(66, 101, 101, 114)
A=(B, e, e, r)
C₁ : Counter for "Beer" = 567
C₂ : Counter for "is" = 263
RDF-triple: (Subject, Relation, Object)
RDF-triple: (Beer, is, proof that ...)

Figure 50. Illustration of the experimental storing algorithm

➢ *Common storing algorithm 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.

Comment: If any of parameters are not given, i.e. <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 in the end of Appendix B.

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

6.3 Estimation of experimental systems

The goal of experiments presented in this chapter is to compare loading times of the experimental program RDFArM with ones of several known systems measured for various datasets. For this purpose, due to different characteristics of the experimental computer configurations and software systems, we need to apply some proportionality constants to make results comparable.

Evaluation, comparison, and selection of modern computer and communication systems are complex decision problem. System evaluation techniques can be either qualitative or quantitative [Dujmovi'c, 1996]:

- Qualitative techniques are usually based on a list of features to be analyzed for each competitive system. The list includes technical characteristics, costs, and other components for evaluation. After a study of proposed systems the evaluator creates for each proposal a list of advantages and a list of disadvantages. The lists summarizing advantages and disadvantages are then intuitively compared and the final ranking of proposed systems is suggested. Such an approach is obviously attractive only when the decision problem is sufficiently simple. In cases with many decision criteria it is difficult to properly intuitively aggregate a number of components affecting the final decision, and it is not possible to precisely identify minor differences between similar proposals. In addition, it is extremely difficult to justify whether a given difference in total cost is commensurate to a corresponding difference in total performance. These difficulties can be reduced by introducing quantitative components in the decision process [Dujmovi'c, 1996].
- The aim of quantitative methods is to make the system evaluation process well structured, relatively simple, and accurate, providing global quantitative indicators which are used to find and to justify the optimum decision [Dujmovi'c, 1996].

For purposes of this research we will use simple evaluation system based on traditional scoring techniques. The basic idea is very simple [Dujmovi'c, 1996]: for a set of evaluated systems we first identify *n* relevant components (performance variables) that are individually evaluated. The results of evaluation are individual normalized scores $E_1, ..., E_n$, where $0 \leq E_i \leq 1$ (or $0 \leq E_i \leq 100\%$). The average score is then

$$E = (E_1 + ... + \quad E_n)/n.$$

If all components are not equally important then we introduce positive normalized weights, which reflect the relative importance of individual components. $W_1,...,W_n$. Usually, $0 \leq W_i \leq 1,$ i = 1, 2,..., n, and $W_1 + ... + W_n = 1.$

The global score is defined as a weighted arithmetic mean:

$$E = W_1E_1 + W_2E_2 + ... W_nE_n, \ 0 \leq E \leq 1.$$

Below we will compare our benchmark hardware configuration with three others. The characteristics we will take in account are Processor, Physical Memory and Hard Disk capacity. We assume that the operating systems and service software are equivalent in all cases. For concrete computer systems used in the experiments we have respectively:

- ✓ **Configuration K** is our benchmark configuration:
  - Processor: Intel Core2 Duo T9550 2.66GHz; CPU Launched: 2009, *Average CPU Mark:* ***1810*** (P_K=1810) [T9550, 2009] http://www.cpubenchmark.net/cpu.php?cpu=Intel+Core2+Duo+T9550+%40+2.66GHz &id=1011;
  - Physical Memory: 4.00 GB (M_K=4);
  - Hard Disk: 100 GB data partition; 2 GB swap (D_K=100);
  - Operating System: 64-bit operating system Windows 7 Ultimate SP1.

  Characteristic values of **Configuration K** are: **P_K=1810, M_K=4, D_K=100**.

- ✓ **Configuration A** is benchmark configuration of [Becker, 2008]:
  - Processor: Intel Pentium Dual Core 2.8 GHz; CPU Launched: 2008; Average CPU Mark: 598 (PA =598) [Pentium Dual, 2008];
  - Physical Memory: 1 GB (MA =1);
  - Hard Disk: 40 GB data partition; 2 GB swap (DA =40);
  - Operating System: Ubuntu Linux 7.10 64-bit.

  Characteristic values of **Configuration A** are: **P_A =598, M_A =1, D_A =40**.

- ✓ **Configuration B**: is benchmark configuration of [BSBMv2, 2008] and [BSBMv3, 2009] DELL workstation:
  - Processor: Intel Core2Quad Q9450 @ 2.66GHz, CPU Launched:2008, Average CPU Mark: 3791 (PB =3791) [Q9450, 2008];
  - Physical Memory: 8GB DDR2 667 (4 x 2GB) (MB =8);
  - Hard Disks: 160GB (10,000 rpm) SATA2, 750GB (7,200 rpm) SATA2 (DB = 160 + 750 = 910);
  - Operating System: Ubuntu 8.04 64-bit, Kernel Linux 2.6.24-16-generic; Java Runtime: VM 1.6.0, HotSpot(TM) 64-Bit Server VM (build 10.0-b23); Separate partitions for application data (on 7,200 rpm HDD) and data bases (on 10,000 rpm HDD).

  Characteristic values of **Configuration B** are: **P_B =3791, M_B =8, D_B =910**.

- ✓ **Configuration C** is benchmark configuration used for LDIF [LDIF Benchmarks, 2013; LDIF, 2013]:
  - Processor: Intel i7 950, 3.07GHz (quad core); CPU Launched: 2009, *Average CPU Mark:* ***5664*** (P_C =5664) [i7 950, 2009];
  - Physical Memory: 24GB (M_C =24);
  - Hard Disks: 2 × 1.8TB (7,200 rpm) SATA2 (D_C =3600);
  - Operating System: Ubuntu 11.04 64-bit, Kernel: 2.6.38-10; Java version: 1.6.0_22.

  Characteristic values of **Configuration C** are: **P_C =5664, M_C =24, D_C =3600**.

- ***Global scores of computer configurations***

Normalized estimation E_P of processors' power will be computed by formula:

$$E_p = \frac{P_i}{P_K}, i = A,B,C$$

where P_j, j=K,A,B,C is the processor's average CPU mark.

We assume that the processors' power is very important and because of this we will use processors weight as 0.5, i.e.

$$W_P = 0.5.$$

Normalized estimation E_M of physical memory will be computed by formula:

$$E_M = \frac{M_i}{M_K}, i = A,B,C$$

where M_j, j=K,A,B,C is the size of main memory in Giga bytes.

We assume that main memory is more important than hard disk memory and because of this we will use main memory weight as 0.3, i.e.

$$W_M = 0.3.$$

Normalized estimation E_{HD} of hard disk capacity will be computed by formula:

$$E_D = \frac{D_i}{D_K}, i = A,B,C$$

where D_j, j=K,A,B,C is the size of hard disk memory in Giga bytes.

We assume that the hard disk memory weight as 0.2, i.e.

$$W_D = 0.2.$$

Formula for computing the global score of computer configuration is defined as a weighted arithmetic mean:

$$E_i = W_P\,E_P + W_M\,E_M + W_D\,E_D$$

or

$$E_i = 0.5\,E_P + 0.3\,E_M + 0.2\,E_D$$

- ***Global scores of experimental computer configurations***

The global scores of experimental computer configurations are as follow.

✧ Global score $\boldsymbol{E_K}$ of configuration **K** is **1:**

$$P_K=1810;\ \mathbf{EK_P} = 1810/1810 = \mathbf{1}$$

$$M_K=4;\ \mathbf{EK_M} = 4/4 = \mathbf{1}$$

$$D_K=100;\ \mathbf{EK_D} = 100/100 = \mathbf{1}$$

$$\boldsymbol{E_K} = 0.5EK_P + 0.3EK_M + 0.2EK_D = 0.5*1+0.3*1+0.2*1 =$$

$$= 0.5+0.3+0.2 = \mathbf{1}$$

✧ Global score $\boldsymbol{E_A}$ of configuration **A** is **0.32**:

$$P_A=598;\ \mathbf{EA_P} = 598/1810 = \mathbf{0.33}$$

$$M_A=1;\ \mathbf{EA_M} = 1/4 = \mathbf{0.25}$$

$$D_A=40;\ \mathbf{EA_D} = 40/100 = \mathbf{0.40}$$

$$\boldsymbol{E_A} = 0.5EA_P+0.3EA_M+0.2EA_D= 0.5*0.33+0.3*0.25+0.2*0.40 =$$

$$= 0.165+0.075+0.08 = \mathbf{0.32}$$

✧ Global score $\boldsymbol{E_B}$ of configuration **B** is **3.465**:

P_B=3791	$\mathbf{EB_P}$ =	3791/1810	= **2.09**
M_B=8	$\mathbf{EB_M}$ =	8/4	= **2**
D_B=910	$\mathbf{EB_D}$ =	910/100	= **9.1**

$$\boldsymbol{E_B} = 0.5EB_P+0.3EB_M+0.2EB_D = 0.5*2.09+0.3*2+0.2*9.1 =$$
$$= 1.045+0.6+1.82 = \mathbf{3.465}$$

✧ Global score $\boldsymbol{E_C}$ of configuration **C** is **10.565**:

P_C=5664;	$\mathbf{EC_P}$ =	5664/1810	= **3.13**
M_C=24;	$\mathbf{EC_M}$ =	24/4	= **6**
D_C=3600;	$\mathbf{EC_H}$ =	3600/100	= **36**

$$\boldsymbol{E_C} = 0.5EC_P+0.3EC_M+0.2EC_D = 0.5*3.13+0.3*6+0.2*36=$$
$$= 1.565+1.8+7.2 = \mathbf{10.565}$$

✓ *Hardware proportionality constants*

The hardware proportionality constants H_i, i = A, B, C, for normalizing our results to be comparable with results received on other computer configurations are as follow:

K∝A :	H_A =	E_K/E_A =	1 / 0.32	= **3.125**
K∝B :	H_B =	E_K/E_B =	1 / 3.465	= **0.289**
K∝C :	H_C =	E_K/E_C =	1 / 10.565	= **0.095**

➢ *Comparing software systems' performance*

Enhancing the hardware power does not cause linear enhancing of the software performance. To discover the value of growth one has to test both source and enhanced systems running equal or similar software.

In our case we have the same problem. Configurations A, K, B, and C, may be ordered by their Average CPU Marks as well as their General scores. In both cases we need to discover the growth of software performance for different configurations. This is needed because we want to have common basis for comparing our load time with those of other systems which are tested on different computer configurations.

For this purpose we will follow simple algorithm.

Let program system **X** is tested on two computer configurations: **U** and **W**, where **W** is enhanced configuration; and program system **Y** is tested on different computer configuration **V** of the same class and similar characteristics as **U**. We have couples (**X**,**U**), (**X**,**W**), and (**Y**,**V**).

Computer configurations **U** and **W** are not available for testing and all work has to be done on computer configuration **V**.

Computer configurations' global scores are respectively:

$\mathbf{E_U}$ = 0.3, $\mathbf{E_V}$ = 1, and $\mathbf{E_W}$ = 3.

X is tested on **U** by dataset **S1** with 200 instances and on **W** with similar dataset **S2** with 250 instances.

Y is tested on configuration **V** by datasets **S1** and **S2**.

Loading times are respectively:

$\mathbf{L}_{(X,U,S1)}$=1000 sec., $\mathbf{L}_{(X,W,S2)}$=5 sec.;

$\mathbf{L}_{(Y,V,S1)}$=400 sec., $\mathbf{L}_{(Y,V,S2)}$=500 sec.

The problem we have to solve is:

"What will be the loading time of system **Y** if it will be run on computer configuration **W** with dataset **S2**?" i.e. $\mathbf{L}_{(Y,W,S2)}$ **= ?**.

Firstly we will illustrate the algorithm and after that we will give it in details.

We have the diagram (Figure 51):

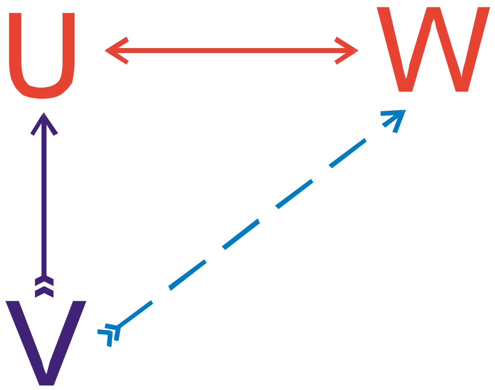


Figure 51. Interrelations between computer configurations

Using published data we may estimate interrelations between computer configurations **U** and **W** as well as between two versions of system **X** run on **U** and **W**. We have to use hardware proportionality constants to make data comparable and to compute the ratio coefficient of software growth by dividing the loading time on **W** by one on **U**.

To make data from experiments on **V** comparable with these on **U** and **W** we assume that **V** and **U** are from the same class of computer power and there is no software growth for a system **Y** in the transition from **V** to **U**. In other words, to estimate interrelations between computer configurations **V** and **U** we need only hardware proportionality constant. After this step we will have data from experiments on **V** transferred for the **U**, i.e. we will have results from system **Y** as if the system **Y** is tested on configuration **U**.

We assume that the possible software growth of system **Y** from computer **U** to **W** is the same as for the system **X**, i.e. we can use the same coefficient for software growth for systems **X** and **Y**. This way we will have comparable data for computer configuration **W**.

Below the algorithm is given in details:

1. Reduce loading time $\mathbf{L_{(X,W,S2)}}$ of program system **X**, run on computer configuration **W** and dataset **S2** with |**S2**|=250 instances, to loading time $\mathbf{L_{(X,W,S2')}}$ of **X** for hypothetical dataset **S2'** with |**S2'**|=|**S1**|=200 instances, using the formula:

$$\mathbf{L_{(X,W,S2')}} = |\mathbf{S2'}| * (\mathbf{L_{(X,W,S2)}} / |\mathbf{S2}|) =$$
$$= |\mathbf{S1}| * (\mathbf{L_{(X,W,S2)}} / |\mathbf{S2}|) = 200*(5/250) = \mathbf{4}$$

2. Compute ratio coefficient of growth $\mathbf{G_{UW}}$ from (**X**,**U**) to (**X**,**W**) by equation:

$$\mathbf{G_{UW}} = \mathbf{L_{(X,U,S1)}}/\mathbf{L_{(X,W,S2')}} = 1000/4 = \mathbf{250}$$

3. Compute loading time $\mathbf{L_{(Y,U,S2)}}$ of system **Y** with dataset **S2** if it is hypothetically ran on configuration **U**, using hardware proportionality constant $\boldsymbol{H_{VU}}$:

$$\mathbf{V \propto U} : \quad \mathbf{H_{VU}} = \quad \mathbf{E_V}/\mathbf{E_U} = \quad 1 / 0.3 \quad = \mathbf{3.33}$$

and formula:

$$\mathbf{L_{(Y,U,S2)}} = \mathbf{H_{VU}} * \mathbf{L_{(Y,V,S2)}} = 3.33 * \mathbf{L_{(Y,V,S2)}} = 3.33*500 = \mathbf{1665}$$

4. Compute loading time $\mathbf{L_{(Y,W,S2)}}$ of system **Y** with dataset **S2** if it is hypothetically ran on configuration **W,** using ratio coefficient of growth $\mathbf{G_{UW}}$, hypothetical loading time $\mathbf{L_{(Y,U,S2)}}$, and formula:

$$\mathbf{L_{(Y,W,S2)}} = \mathbf{L_{(Y,U,S2)}}/\mathbf{G_{UW}} = \mathbf{L_{(Y,U,S2)}} / 250 = 1665/250 = \mathbf{6.66}$$

This way we have received comparable value of loading time of system **Y** with system **X** for computer configuration **W**, i.e.

$$\mathbf{L_{(X,W,S2)}}=5 \text{ sec. and } \mathbf{L_{(Y,W,S2)}} = 6.66 \text{ sec.}$$

and we may conclude that system **X** will have a little better loading time than system **Y** if both are run on computer configuration **W** with dataset **S2**.

One may suppose that we may use directly proportionality constant $\boldsymbol{H_{WV}}$:

$$\mathbf{W \propto V} : \quad \boldsymbol{H_{WV}} = \quad \mathbf{E_W}/\mathbf{E_V} = \quad 3 / 1 \quad = \mathbf{3}$$

and to reduce $\mathbf{L_{(Y,V,S2)}}$=500 sec. three times, i.e. 500/3 = 166.66.

This is not correct because the *software growth* is not taken in account.

We have to calculate possible software growth from **V** to **W** again going through **U** and using $\mathbf{G_{UW}}$ to calculate possible $\mathbf{G_{VW}}$. This may be done by using the proportionality constant $\mathbf{H_{VU}}$ because we need to calibrate growth from **U** to **W** by hardware proportionality of **V** and **U**. In other words, to receive value of growth $\mathbf{G_{VW}}$ from **V** to **W** we have to compute:

$$\mathbf{G_{VW}} = \mathbf{G_{UW}}/\mathbf{H_{VU}}$$

Finally:

$$\mathbf{L_{(Y,W,S2)}} = \mathbf{L_{(Y,V,S2)}}/\mathbf{G_{VW}}$$

Let see it for concrete values:

$$\mathbf{G_{UW}} = \mathbf{L_{(X,U,S1)}}/\mathbf{L_{(X,W,S2')}} = 1000/4 = \mathbf{250}$$
$$\mathbf{H_{VU}} = \mathbf{E_V}/\mathbf{E_U} = \quad 1 / 0.3 \quad = \mathbf{3.33}$$
$$\mathbf{G_{VW}} = (\mathbf{G_{UW}}/\mathbf{H_{VU}}) = (250/3.33) = \mathbf{75.07}$$
$$\mathbf{L_{(Y,W,S2)}} = \mathbf{L_{(Y,V,S2)}}/\mathbf{G_{VW}} = 500 / 75.07 = \mathbf{6.66}$$

We received the same result as algorithm above. This proves that we have equivalent approaches.

The algorithm may be presented by a formula:

$$L_{(Y,W,S2)} = R_{YVW} * L_{(Y,V,S2)}$$

where

$$R_{YVW} = \frac{E_v * | S1 | * L_{(X,W,S2)}}{E_U * | S2 | * L_{(X,U,S1)}}$$

i.e.

$$L_{(Y,W,S2)} = \frac{E_v * | S1 | * L_{(X,W,S2)}}{E_U * | S2 | * L_{(X,U,S1)}} * L_{(Y,V,S2)}$$

where:

- **X**, **Y** - program systems;
- **U**, **V**, **W** – computer configurations;
- (**X**,**U**), (**X**,**W**), (**Y**,**V**) – couples "program system – computer configuration";
- $\mathbf{E_U}$, $\mathbf{E_V}$, $\mathbf{E_W}$ - computer configurations' global scores;
- **S1**, **S2** – datasets;
- $\mathbf{L_{(X,U,S1)}}$, $\mathbf{L_{(X,W,S2)}}$, $\mathbf{L_{(Y,V,S1)}}$, $\mathbf{L_{(Y,V,S2)}}$, $\mathbf{L_{(Y,W,S2)}}$ - loading times of given program system, computer configuration, and dataset;
- $\mathbf{H_{VU}}$ – computer configurations' proportionality constant;
- $\mathbf{G_{UW}}$ – ratio coefficient of growth of software system during migration from a computer configuration to enhanced one.

➢ *Experimental environment*

Our experimental environment includes program systems, computer configurations, datasets and experimental data like published benchmark results, different constants, ratio coefficients, etc. The main concrete elements of our experimental environment are:

- Program systems to be compared are:
  - RDFArM;
  - Virtuoso;
  - Jena;
  - Sesame.

  Virtuoso, Jena and Sesame have several variants depending of database used. These variants have different loading times on the same computer configurations. In our comparisons we will take the best result from the all benchmarks on given configuration.
- Computer configurations used for benchmarking are **A**, **K**, **B**, **C**;
- Couples "program system – computer configuration" are:
  - (RDFArM, **K**);
  - (Virtuoso, **A**), (Virtuoso, **B**), (Virtuoso, **C**);
  - (Jena, **A**), (Jena, **B**), (Jena, **C**);
  - (Sesame, **A**), (Sesame, **B**), (Sesame, **C**).
- Computer configurations' global scores are $\mathbf{E_A}$, $\mathbf{E_K}$, $\mathbf{E_B}$, and $\mathbf{E_C}$;

- Middle-size datasets are:
  - *BSBM 50K;*
  - *homepages-fixed.nt;*
  - *BSBM 250K;*
  - *geocoordinates-fixed.nt;*
  - *BSBM 1M;*
  - *BSBM 5M.*
- Large size datasets are:
  - *infoboxes-fixed.nt;*
  - *BSBM 25M;*
  - *BSBM 100M.*
- Proportionality constant between computer configurations **K** and **A** is $\mathbf{H_{KA}}$;
- Ratio coefficient of growth of software systems during migration from computer configuration A to enhanced ones B and C are $\mathbf{G_{AB}}$ and $\mathbf{G_{AC}}$;
- Corresponded loading times **L** will be presented at the places where they will be used.

➢ *Software proportionality constants*

To provide concrete comparisons of our experimental loading time data, we have to compute $\mathbf{H_{KA}}$, $\mathbf{G_{AB}}$, and $\mathbf{G_{AC}}$.

For purposes of this research it is enough to compute average constants $\mathbf{H_{KA}}$, $\mathbf{G_{AB}}$, and $\mathbf{G_{AC}}$ based on average loading data for all chosen systems. We will use published benchmark results done by Freie Universität Berlin, Web-based Systems Group (BSBM team) and available both as printed publication and free accessible data in the Internet.

✓ *Software proportionality for configurations K, A, and B*

Benchmark results for dataset **S1** (homepages-fixed.nt; 200 036 triples) used for benchmarks on **Configuration A** are published in [Becker, 2008] and reproduced in Table 35.

Table 35. Benchmark results for dataset S1 (homepages-fixed.nt)

system	loading time in seconds	the best time in seconds
Virtuoso (ogps, pogs, psog, sopg)	1327	1327
Jena SDB MySQL Layout 2 Index	5245	3557
Jena SDB Postgre SQL Layout 2 Index	3557	
Jena SDB Postgre SQL Layout 2 Hash	9681	
Sesame Native (spoc, posc)	2404	2404
	Total average time in seconds:	**2429.333**

Benchmark results for dataset **S2** (*BSBM 250K*; 250 030 triples) used for benchmarks on **Configuration B** are published in [BSBMv2, 2008] and reproduced in Table 36.

Table 36. Benchmark results for dataset S2 (BSBM 250K)

system	loading time in seconds
Virtuoso	33
Jena SDB	24
Sesame	18
Total average time in seconds:	**25**

Due to equal systems and range of their loading times on the same computer configuration, we will use total average times as loading times of virtual system **X**, i.e. $\mathbf{L_{(X,A,S1)}}$ = **2429.333** and $\mathbf{L_{(X,B,S2)}}$ = **25**.

Following our algorithm, we reduce loading time $\mathbf{L_{(X,B,S2)}}$ of virtual system **X**, run on computer configuration **B** and dataset **S2** with |**S2**|=250 030 triples, to loading time $\mathbf{L_{(X,B,S2')}}$ of **X** for hypothetical dataset **S2'** with |**S2'**|=|**S1**|=200 036 instances, using the formula

$$\mathbf{L_{(X,B,S2')}} = |\mathbf{S1}| * (\mathbf{L_{(X,B,S2)}}/ |\mathbf{S2}|) = 200036*(25/250030) = \mathbf{20.00}.$$

We compute ratio coefficient of growth $\mathbf{G_{AB}}$ from (**X**,**A**) to (**X**,**B**) by equation:

$$\mathbf{G_{AB}} = \mathbf{L_{(X,A,S1)}}/\mathbf{L_{(X,B,S2')}} = 2429.333/20 = \mathbf{121.46665}.$$

Hardware proportionality constant $\mathbf{H_{AK}}$ is:

$$\mathbf{A \propto K}: \quad \mathbf{H_{AK}} = \quad \mathbf{E_K/E_A} = \quad 1 / 0.32 \quad = \mathbf{3.125}$$

Really measured **RDFArM** loading time on Configuration **K** for dataset **S2** is **575.069** sec. We compute loading time $\mathbf{L_{(RDFArM,A,S2)}}$ using formula:

$$\mathbf{L_{(RDFArM,A,S2)}} = \mathbf{H_{AK}}*\mathbf{L_{(RDFArM,K,S2)}} = 3.125*575.069 = 1797.09.$$

At the end, we compute loading time $\mathbf{L_{(RDFArM,B,S2)}}$ of system **RDFArM** with dataset **S2** if it is hypothetically run on configuration **B,** using ratio coefficient of growth $\mathbf{G_{AB}}$, hypothetical loading time $\mathbf{L_{(RDFArM,A,S2)}}$, and formula:

$$\mathbf{L_{(RDFArM,B,S2)}} = \mathbf{L_{(RDFArM,A,S2)}}/\mathbf{G_{AB}} = 1797.09 / 121.46665= \mathbf{14.796}$$

To **verify** our computations and to show *the easiest way* to find $\mathbf{L_{(RDFArM,B,S2)}}$, we will use our formula

$$L_{(RDFArM,B,S2)} = R_{RDFArM,K,B} * L_{(RDFArM,K,S2)}$$

i.e. we have to compute $\mathbf{R_{RDFArM,K,B}}$ one time and to use it in benchmarks for all datasets. $\mathbf{R_{RDFArM,K,B}}$ may be computed by formula:

$$R_{RDFArM,A,B} = \frac{E_K * | S1 | * L_{(X,B,S2)}}{E_A * | S2 | * L_{(X,A,S1)}}$$

or in linear view:

$$\mathbf{R_{RDFArM,K,B}} = (\mathbf{E_K} * |\mathbf{S1}| * \mathbf{L_{(X,B,S2)}}) / (\mathbf{E_A} * |\mathbf{S2}| * \mathbf{L_{(X,A,S1)}}) =$$
$$= (1 * 200036 * 25) / (0.32 * 250030 * 2429.333) =$$
$$= 5000900 / 194369961.5968 = \mathbf{0.025729}.$$

We compute loading time $\mathbf{L_{(RDFArM,B,S2)}}$ of system **RDFArM** with dataset **S2** if it is hypothetically run on configuration **B,** using ratio coefficient $\mathbf{R_{RDFArM,K,B}}$:

$$\mathbf{L_{(RDFArM,B,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{R_{RDFArM,K,B}} = 575.069 * 0.025729 = \mathbf{14.796}.$$

We receive the same result.

✓ *Software proportionality for configurations K, A, and C*

Software proportionality for configurations **K**, **A**, and **C** will be computed based on the performance of systems Virtuoso and Jena because missing information about Sesame in the benchmark publications.

Benchmark results for dataset **S1** (*infoboxes-fixed.nt*; 15,472,624 triples) used for benchmarks on **Configuration A** are published in [Becker, 2008] and reproduced in Table 37.

Table 37. Benchmark results for dataset S1 (infoboxes-fixed.nt)

system	loading time in seconds	the best time in seconds
Virtuoso	7017	7017
Jena SDB MySQL Layout 2 Index	70851	70851
Jena SDB Postgre SQL Layout 2 Index	73199	
Jena SDB Postgre SQL Layout 2 Hash	734285	
	Total average time:	**38934**

Benchmark results for dataset **S2** (*BSBM 100M*; 100 000 748 triples) used for benchmarks on **Configuration C** are published in [BSBMv6, 2011] and reproduced in Table 38.

Table 38. Benchmark results for dataset S2 (BSBM 100M)

system	loading time in seconds
Virtuoso	6566
Jena TDB	4488
Total average time:	**5527**

Following our algorithm, we reduce loading time $\mathbf{L_{(X,C,S2)}}$ of virtual system **X**, run on computer configuration **C** and dataset **S2** with |**S2**|=100 000 748 triples, to loading time $\mathbf{L_{(X,C,S2')}}$ of **X** for hypothetical dataset **S2'** with |**S2'**|=|**S1**|=15 472 624 instances, using the formula:

$$\mathbf{L_{(X,C,S2')}} = |\mathbf{S1}| * (\mathbf{L_{(X,C,S2)}}/\ |\mathbf{S2}|) =$$

$$= 15472624*(5527/100000748) = \mathbf{855.166}.$$

We compute ratio coefficient of growth $\mathbf{G_{AC}}$ from (**X**,**A**) to (**X**,**C**) by equation:

$$\mathbf{G_{AC}} = \mathbf{L_{(X,A,S1)}}/\mathbf{L_{(X,C,S2')}} = 38934/855.166 = \mathbf{45.528}.$$

Hardware proportionality constant $\boldsymbol{H_{AK}}$ is:

$$\mathbf{A{\propto}K}: \quad \mathbf{H_{AK}} = \quad \mathbf{E_K/E_A} = \quad 1 / 0.32 \quad = \mathbf{3.125}.$$

Really measured **RDFArM** loading time on Configuration **K** for dataset **S2** is 43652.528 sec. We compute loading time $\mathbf{L_{(RDFArM,A,S2)}}$ using formula:

$$\mathbf{L_{(RDFArM,A,S2)}} = \mathbf{H_{AK}}*\mathbf{L_{(RDFArM,K,S2)}} = 3.125*43652.528 = \mathbf{136414.15}.$$

At the end, we compute loading time $\mathbf{L_{(RDFArM,C,S2)}}$ of system **RDFArM** with dataset **S2** if it is hypothetically run on configuration **C,** using ratio coefficient of growth $\mathbf{G_{AC}}$, hypothetical loading time $\mathbf{L_{(RDFArM,A,S2)}}$, and formula:

$$\mathbf{L_{(RDFArM,C,S2)}} = \mathbf{L_{(RDFArM,A,S2)}}/\mathbf{G_{AC}} = 136414.15/45.528 = \mathbf{2996.27}\text{ sec.}$$

To **verify** our computations and to show *the easiest way* to find $\mathbf{L_{(RDFArM,C,S2)}}$, we will use our formula

$$L_{(RDFArM,C,S2)} = R_{RDFArM,K,C} * L_{(RDFArM,K,S2)}$$

i.e. we have to compute $\mathbf{R_{RDFArM,K,C}}$ one time and to use it in benchmarks for all datasets. $\mathbf{R_{RDFArM,K,C}}$ may be computed by formula:

$$R_{RDFArM,A,C} = \frac{E_K * |\, S1\, | * L_{(X,C,S2)}}{E_A * |\, S2\, | * L_{(X,A,S1)}}$$

or in linear view:

$$\mathbf{R_{RDFArM,K,C}} = (\mathbf{E_K} * |\mathbf{S1}| * \mathbf{L_{(X,C,S2)}}) / (\mathbf{E_A} * |\mathbf{S2}| * \mathbf{L_{(X,A,S1)}}) =$$

$$= (1 * 15472624 * 5527) / (0.32 * 100000748 * 38934) =$$

$$= 85517192848 / 1245897319242.24 = \mathbf{0.068639}.$$

We compute loading time $\mathbf{L_{(RDFArM,C,S2)}}$ of system **RDFArM** with dataset **S2** if it is hypothetically run on configuration **C,** using ratio coefficient $\mathbf{R_{RDFArM,K,C}}$:

$$\mathbf{L_{(RDFArM,C,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{R_{RDFArM,K,C}} = 43652.528 * 0.068639 = \mathbf{2996.27}.$$

We receive same result.

✓ *Ratio coefficients*

To compare our results from experiments on computer configuration **K** we will use ratio coefficients:

- For published results received on computer configuration **A**:

  $$\mathbf{L_{(RDFArM,A,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{3.125};$$

- For published results received on computer configuration **B**:

  $$\mathbf{L_{(RDFArM,B,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{0.025729};$$

- For published results received on computer configuration **C**:

  $$\mathbf{L_{(RDFArM,C,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{0.068639}.$$

6.4 Experiments with middle-size datasets

We will compare RDFArM with RDF-stores:

- OpenLink Virtuoso Open-Source Edition 5.0.2 [Virtuoso, 2013];
- Jena SDB Beta 1 on PostgreSQL 8.2.5 and MySQL 5.0.45 [Jena, 2013];
- Sesame 2.0 [Sesame, 2012],

tested by Berlin SPARQL Bench Mark (BSBM) team and connected to it research groups [Becker, 2008; BSBMv2, 2008; BSBMv3, 2009]. More information about latest versions of these systems is given in Appendix B.

We will provide experiments with *middle-size RDF-datasets*, based on selected real datasets from DBpedia [DBpedia, 2007a; DBpedia, 2007b] and artificial datasets created by BSBM Data Generator [BSBM DG, 2013; Bizer & Schultz, 2009].

The real middle-size RDF-datasets which we will use consist of DBpedia's homepages and geocoordinates datasets with minor corrections [Becker, 2008]:

- *Homepages-fixed.nt* (200,036 triples; 24 MB) Based on DBpedia's homepages.nt dated 2007-08-30 [DBpedia, 2007a]. 3 URLs that included line breaks were manually corrected (fixed for DBpedia 3.0);
- *Geocoordinates-fixed.nt* (447,517 triples; 64 MB) Based on DBpedia's geocoordinates.nt dated 2007-08-30 [DBpedia, 2007b]. Decimal data type URI was corrected (DBpedia bug #1817019; resolved);

The RDF stores feature different indexing behaviors: Sesame automatically indexes after each import, while SDB and Virtuoso allow for selective index activation which caouse corresponded limitations or advantages. In order to make load times comparable, the data import by [Becker, 2008] was performed as follows:

- *Homepages-fixed.nt* was imported with indexes enabled;
- *Geocoordinates-fixed.nt* was imported with indexes enabled.

In the case with RDFArM no parameters are needed. The data sets were loaded directly from the source N-triple files.

The artificial middle-size RDF-datasets are generated by BSBM Data Generator [BSBM DG, 2013] and published in N-triple as well as in Turtle format [BSBMv1, 2008; BSBMv2, 2008; BSBMv3, 2009]. We converted Turtle format in N-triple format using “rdf2rdf” program developed by Enrico Minack [Minack, 2010].

We have use four BSBM datasets – 50K, 250K, 1M, and 5M. Details about these datasets are summarized in following Table 39.

Table 39. Details about used artificial middle-size RDF-datasets

Name of RDF-dataset:	**50K**	**250K**	**1M**	**5M**
Exact Total Number of Triples:	**50,116**	**250,030**	**1,000,313**	**5,000,453**
Number of Products	91	666	2,785	9,609
Number of Producers	2	14	60	199
Number of Product Features	580	2,860	4,745	3,307
Number of Product Types	13	55	151	73
Number of Vendors	2	8	34	196
Number of Offers	1,820	13,320	55,700	192,180
Number of Reviewers	116	339	1432	12,351
Number of Reviews	2,275	6,660	27,850	240,225
Total Number of Instances	4,899	23,922	92,757	*458,140*
File Size Turtle (unzipped)	14 MB	22 MB	86 MB	*1,4 GB*

✓ *Loading of BSBM 50K*

RDFArM has loaded all **50116** triples from BSBM 50K for about **113 seconds** (112851 ms) or average time of **2.3 ms** per triple (Figure 52).

Number of Subjects in this dataset was S=4900; number of relations R=40; and number of objects O=50116.

This means that practically we had 40 layers with 4900 NL-locations (containers) which contain 50116 objects. The loading time' results from our experiment and [Bizer & Schultz, 2008] are given in Table 40 and shown on Figure 53.

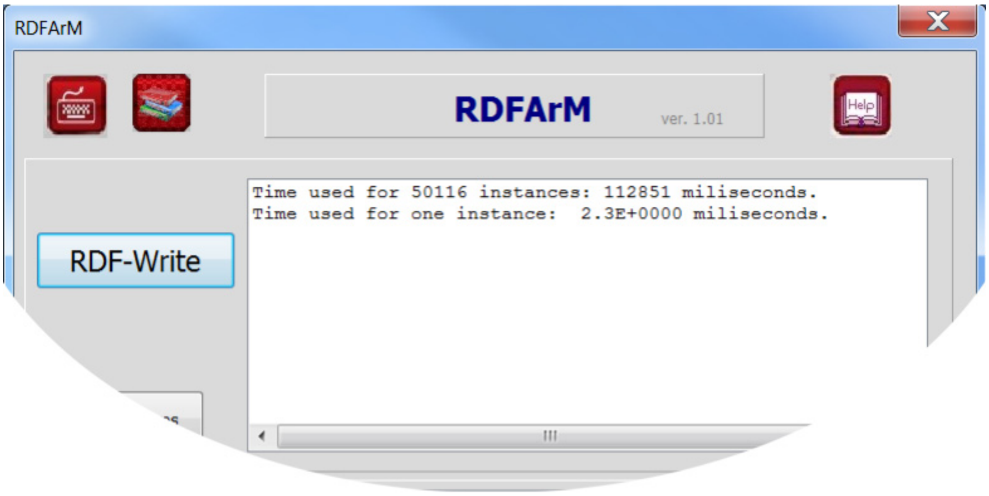
Benchmark configuration used by [Bizer&Schultz, 2008] is **Configuration B**.

Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is

$$\mathbf{L_{(RDFArM,B,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{R_{RDFArM,K,B}}, \text{ and } \mathbf{R_{RDFArM,K,B}} = \mathbf{0.025729};$$

and we compute final loading time as follow: 113 * 0.025729= **2.91 sec**.



```
Time used for 50116 instances: 112851 miliseconds.
Time used for one instance:  2.3E+0000 miliseconds.
```

Figure 52. Screenshot of the report of RDFArM for BSBM 50K

Table 40. Benchmark results for BSBM 50K

system	loading time in seconds
Sesame	3
Jena SDB	5
Virtuoso	2
RDFArM	**3**

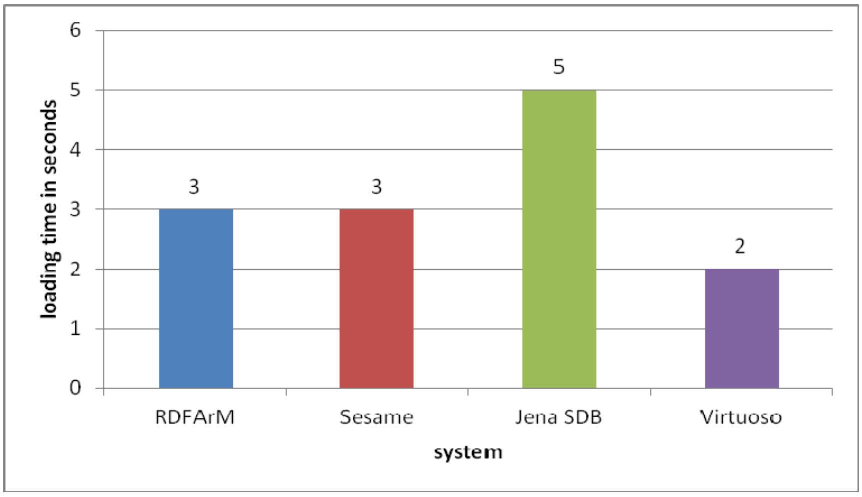
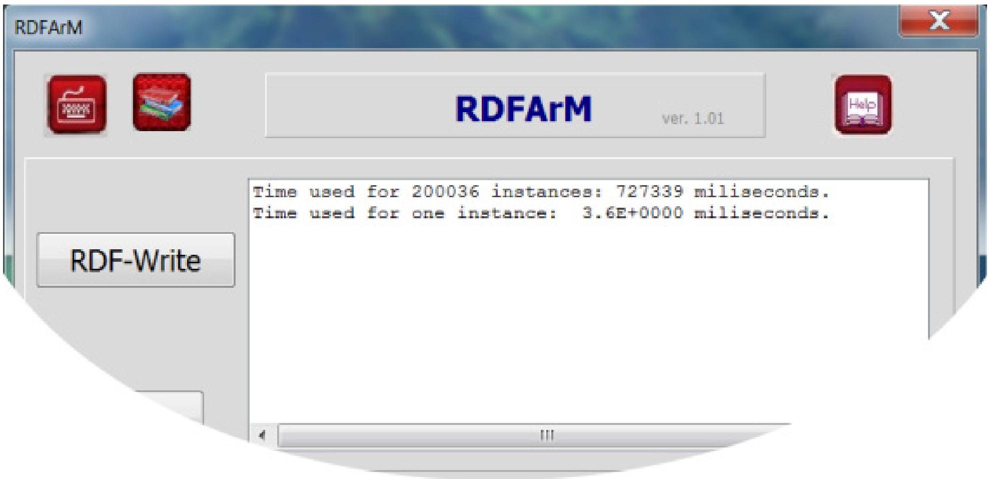


Figure 53. Benchmark results for BSBM 50K

From Table 40 and Figure 53 we may conclude that for **BSBM 50K** Virtuoso has the best time, RDFArM has same loading time as Sesame and 40% better performance than Jena.

✓ *Loading of homepages-fixed.nt*

RDFArM has loaded all **200036** triples from *homepages-fixed.nt* for about **727 seconds** (727339 ms) or average time of **3.6 ms** per triple (Figure 54).



```
Time used for 200036 instances: 727339 miliseconds.
Time used for one instance:  3.6E+0000 miliseconds.
```

Figure 54. Screenshot of the report of RDFArM for homepages-fixed.nt

More detailed information is given in Table 41. Every row of this table contains data for storing of one hundred thousand triples. Total stored triples were 200036 and Table 41 contains three rows.

Table 41. RDFArM results for homepages-fixed.nt

part	triples stored	ms for all	ms for one	Subjects	Relations	Objects
1	100000	360955	3.6	100000	1	100000
2	100000	366275	3.7	100000	1	100000
3	36	109	3.0	36	1	36
Total:	**200036**	**727339**	**3.6**	**200036**	**1**	**200036**

Number of Subjects in this dataset was S=200036; number of relations R=1; and number of objects O=200036.

This means that practically we had only one layer with 200036 NL-locations (containers) which contain the same number of objects. The loading time' results from our experiment and [Becker, 2008] are given in Table 42 and Figure 55.

Benchmark configuration used by [Becker, 2008] is **Configuration A**.

Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is

$$\mathbf{L}_{\mathbf{(RDFArM,A,S2)}} = \mathbf{H}_{\mathbf{AK}}*\mathbf{L}_{\mathbf{(RDFArM,K,S2)}}, \text{ where } \mathbf{H}_{\mathbf{AK}} = \mathbf{3.125};$$

and we compute final loading time as follow: 727 x 3.125 = **2271.875** sec.

Table 42. ***Benchmark results for homepages-fixed.nt***

system	loading time in seconds
Virtuoso (ogps, pogs, psog, sopg)	1327
Jena SDB MySQL Layout 2 Index	5245
Jena SDB Postgre SQL Layout 2 Index	3557
Jena SDB Postgre SQL Layout 2 Hash	9681
Sesame Native (spoc, posc)	2404
RDFArM	**2272**

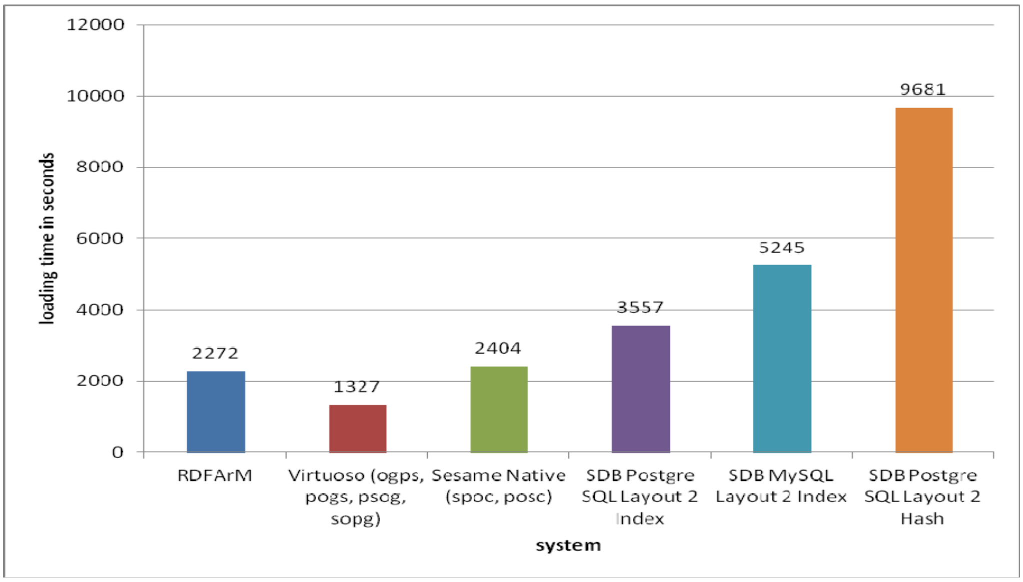


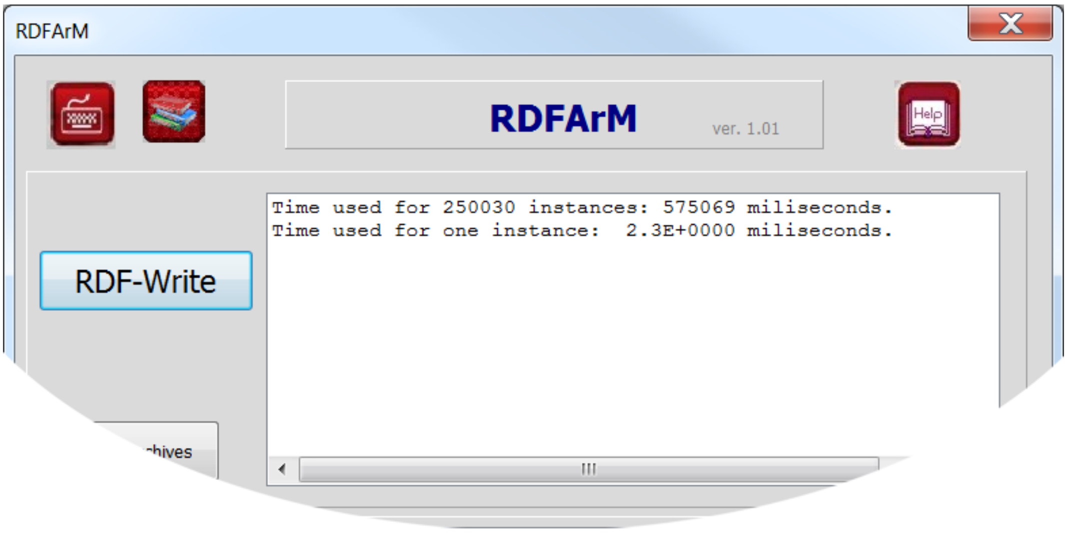
Figure 55. Benchmark results for homepages-fixed.nt

From Table 42 we may conclude that Virtuoso has the best time (about 42% better result than RDFArM); RDFArM has about 5% better time than Sesame and 36% better time than Jena (we take in account only the best results of compared systems, in this case – Jena).

✓ *Loading of BSBM 250K*

RDFArM has loaded all **250030** triples from BSBM 250K for about **575 seconds** (575069 ms) or average time of **2.3 ms** per triple (Figure 56).

More detailed information is given in Table 43. Every row of this table contains data for storing of one hundred thousand triples. Total stored triples were 250030 and Table 43 contains three rows.



```
Time used for 250030 instances: 575069 miliseconds.
Time used for one instance:  2.3E+0000 miliseconds.
```

Figure 56. Screenshot of the report of RDFArM for BSBM 250K

Table 43. RDFArM results for BSBM 250K

part	triples stored	ms for all	ms for one	Subjects	Relations	Objects
1	100000	238525	2.4	19854	6	100000
2	100000	228854	2.3	26505	22	100000
3	50030	107690	2.1	14525	22	50030
Total:	**250030**	**575069**	**2.3**	**60884**	**22**	**250030**

Number of Subjects in this dataset was S=60884; number of relations R=22; and number of objects O=250030.

This means that practically we had 22 layers with 60884 NL-locations (containers) which contain 250030 objects. The loading time' results from our experiment and [BSBMv2, 2008] are given in Table 44 and shown on Figure 57.

Benchmark configuration used by [BSBMv2, 2008] is **Configuration B**.

Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is

$\mathbf{L_{(RDFArM,B,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{R_{RDFArM,K,B}}$, and $\mathbf{R_{RDFArM,K,B}} = \mathbf{0.025729}$;

and we compute final loading time as follow: 575 x 0.025729= **14.79 sec**.

Table 44. ***Benchmark results for BSBM 250K***

system	**loading time in seconds**
Sesame	19
Jena TDB	13
Virtuoso TS	05
Virtuoso RDF views	09
Virtuoso SQL	09
RDFArM	**14.79**

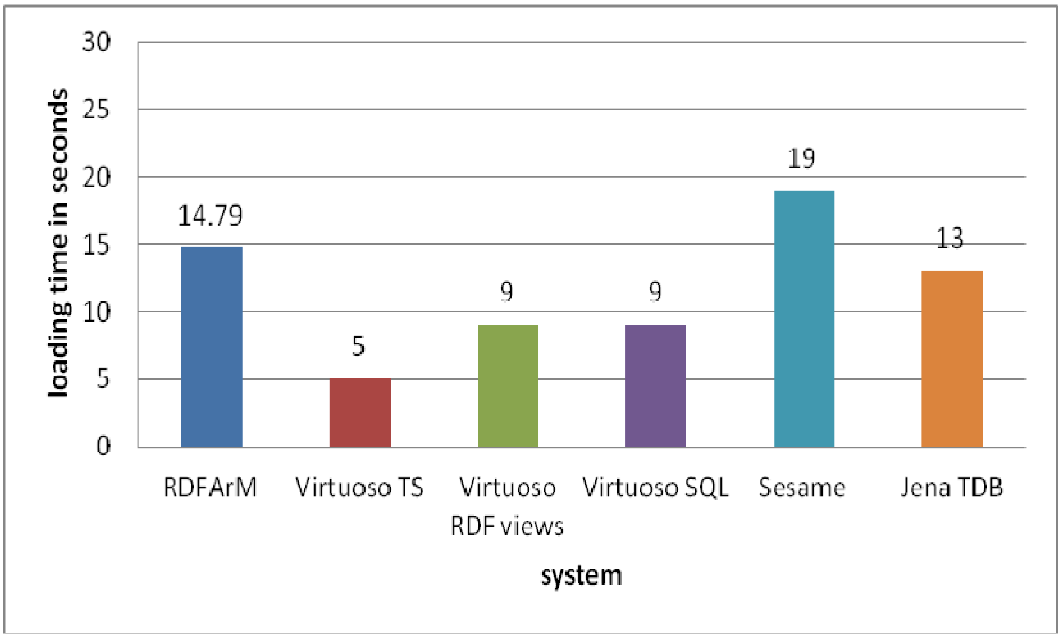


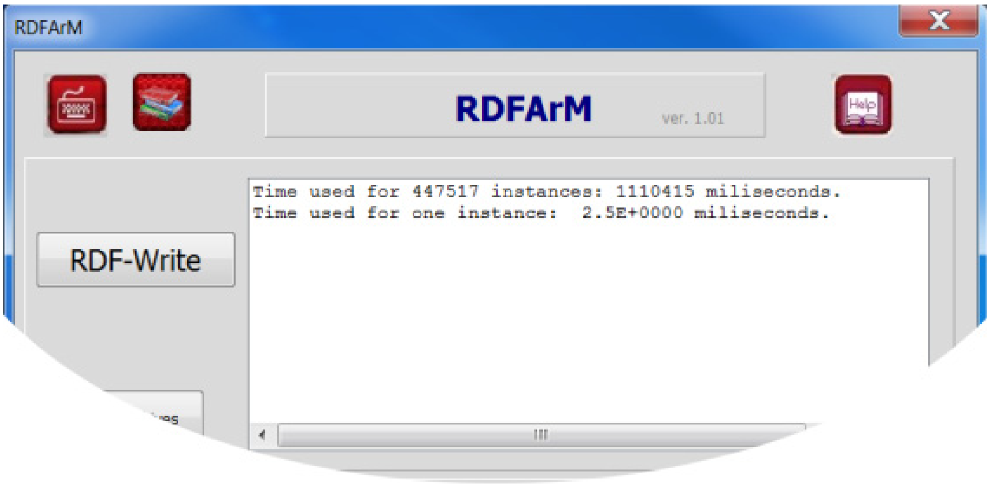
Figure 57. Benchmark results for BSBM 250K

From Table 44 and Figure 57 we may conclude that Virtuoso has 66% and Jena has 12% better performance than RDFArM. RDFArM has 22% better performance than Sesame.

✓ ***Loading of geocoordinates-fixed.nt***

RDFArM has loaded all 447517 triples from geocoordinates-fixed.nt for about 1110 seconds (1110415 ms) or average time of 2.5 ms per triple (Figure 58).

More detailed information is given in Table 45. Every row of this table contains data for storing of one hundred thousand triples. Total stored triples were 447517 and Table 45 contains five rows.



```
Time used for 447517 instances: 1110415 miliseconds.
Time used for one instance:  2.5E+0000 miliseconds.
```

Figure 58. Screenshot of the report of RDFArM for geocoordinates-fixed.nt

Table 45. RDFArM results for geocoordinates-fixed.nt

part	triples stored	ms for all	ms for one	Subjects	Relations	Objects
1	100000	244453	2.4	34430	6	100000
2	100000	246747	2.5	34909	6	100000
3	100000	245530	2.5	33863	6	100000
4	100000	248198	2.5	33678	6	100000
5	47517	47517	2.6	16095	6	47517
Total:	**447517**	**1110415**	**2.5**	**152975**	**6**	**447517**

Number of Subjects in this dataset was S=152975; number of relations R=6; and number of objects O=447517.

This means that practically we had six layers with 152975 NL-locations (containers) which contain 447517 objects, i.e. some containers in some layers are empty. The loading time' results from our experiment and [Becker, 2008] are given in Table 46 and Figure 59.

Benchmark configuration used by [Becker, 2008] is **Configuration A**.

Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is

$$\mathbf{L}_{\mathbf{(RDFArM,A,S2)}} = \mathbf{H}_{\mathbf{AK}}{*}\mathbf{L}_{\mathbf{(RDFArM,K,S2)}},\text{ where } \mathbf{H}_{\mathbf{AK}} = \mathbf{3.125};$$

and we compute final loading time as follow: 1110 x 3.125= **3468.75 sec**.

Table 46. Benchmark results for geocoordinates-fixed.nt

system	loading time in seconds
Virtuoso (ogps, pogs, psog, sopg)	1235
Jena SDB MySQL Layout 2 Index	6290
Jena SDB Postgre SQL Layout 2 Index	3305
Jena SDB Postgre SQL Layout 2 Hash	9640
Sesame Native (spoc, posc)	2341
RDFArM	**3469**

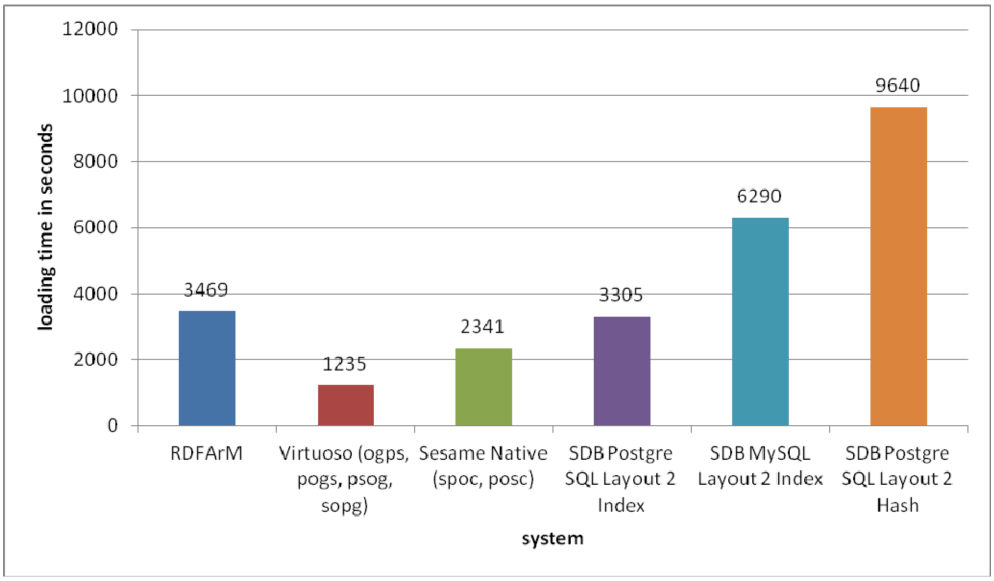
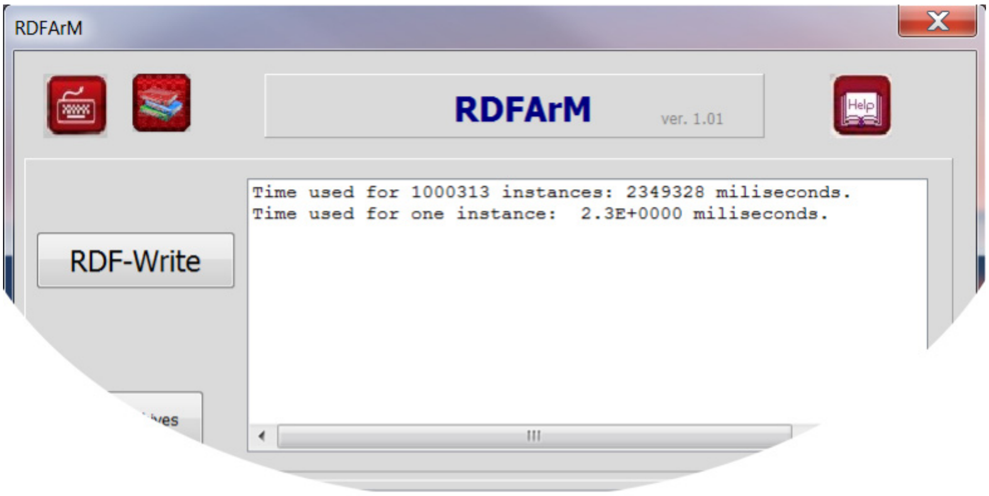


Figure 59. Benchmark results for geocoordinates-fixed.nt

From Table 46 and Figure 59 we may conclude that RDFArM has the worst performance (we take the best time of Jena). Virtuoso has 64%, Sesame has 33%, and Jena has 5% better performance.

✓ *Loading of BSBM 1M*

RDFArM has loaded all **1000313** triples from BSBM 1M for about **2349 seconds** (2349328 ms) or average time of **2.3 ms** per triple (Figure 60).



```
Time used for 1000313 instances: 2349328 miliseconds.
Time used for one instance:  2.3E+0000 miliseconds.
```

Figure 60. Screenshot of the report of RDFArM for BSBM 1M

More detailed information is given in Table 47. Every row of this table contains data for storing of one hundred thousand triples. Total stored triples were 1000313 and Table 47 contains 11 rows. This table has new structure. It contains number of stored triples to corresponded part including it and in separate columns the time for storing the last 100000 triples and average time for one triple from this part.

Table 47. RDFArM results for BSBM 1M

part	triples stored	ms for all	ms for one	ms for last 100000	ms for one	Subjects	Relations	Objects
1	100000	241099	2.4	241099	2.4	6859	22	100000
2	200000	480265	2.4	239166	2.4	14363	29	200000
3	300000	714453	2.4	234188	2.3	24365	29	300000
4	400000	962994	2.4	248541	2.5	34366	29	400000
5	500000	1194344	2.4	231350	2.3	44368	29	500000
6	600000	1423665	2.4	229321	2.3	54370	29	600000

part	triples stored	ms for all	ms for one	ms for last 100000	ms for one	Subjects	Relations	Objects
7	700000	1655420	2.4	231755	2.3	64324	40	700000
8	800000	1892074	2.4	236654	2.4	73799	40	800000
9	900000	2116590	2.4	224516	2.2	83269	40	900000
10	1000000	2348501	2.3	231911	2.3	92729	40	1000000
11	1000313	2349328	2.3	827	2.6	92757	40	1000313

Number of Subjects in this dataset was S=92757; number of relations R=40; and number of objects O=1000313.

This means that practically we had 40 layers with 92757 NL-locations (containers) which contain 1000313 objects. The loading time' results from our experiment and [BSBMv2, 2008; BSBMv3, 2009] are given in Table 48 and shown on Figure 61.

Benchmark configuration used by [BSBMv2, 2008; BSBMv3, 2009] is **Configuration B**.

Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is

$$\mathbf{L_{(RDFArM,B,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{R_{RDFArM,K,B}}, \text{ and } \mathbf{R_{RDFArM,K,B}} = \mathbf{0.025729};$$

and we compute final loading time as follow: 2349 x 0.025729 = **60.437421 sec**.

Table 48. Benchmark results for BSBM 1M

system	**loading time in min:sec**	
	(a) [BSBMv2, 2008]	(b) [BSBMv3, 2009]
Sesame	02:59	03:33
Jena TDB	00:49	00:41
Jena SDB	02:09	-
Virtuoso TS	00:23	00:25
Virtuoso RV	00:34	00:33
Virtuoso SQL	00:34	00:33
RDFArM	**01:00**	**01:00**

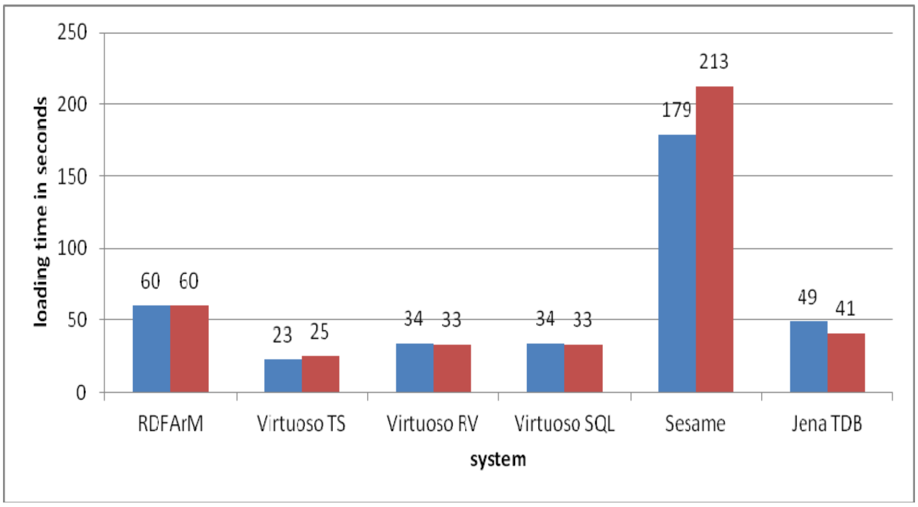
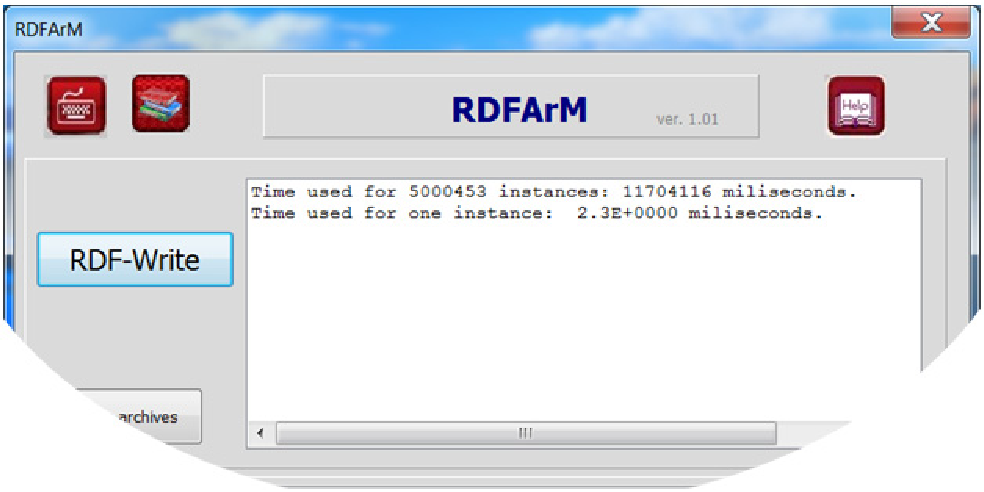


Figure 61. Benchmark results for BSBM 1M

From Table 48 and Figure 61 we may conclude that Virtuoso has 62% and Jena has 32% better performance than RDFArM. RDFArM has 67% better performance than Sesame.

- ***Loading of BSBM 5M***

RDFArM has loaded all **5000453** triples from BSBM 5M for about **11704 sec.** (11704116 ms) or average time of **2.3 ms** per triple (Figure 62).



```
Time used for 5000453 instances: 11704116 miliseconds.
Time used for one instance:  2.3E+0000 miliseconds.
```

Figure 62. Screenshot of the report of RDFArM for BSBM 5M

Number of Subjects in this dataset was S=458142; number of relations R=55; and number of objects O=5000453.

This means that practically we had 55 layers with 458142 NL-locations (containers) which contain 5000453 objects. The loading time' results from our experiment and [Bizer & Schultz, 2008] are given in Table 50 and shown on Figure 63.

More detailed information is given in Table 49. Every row of this table contains data for storing of one hundred thousand triples. Total stored triples were 5000453 and Table 49 contains 51 rows.

This table contains number of stored triples to corresponded part including it and in separate columns the time for storing the last 100000 triples and average time for one triple from this part.

Table 49. RDFArM results for BSBM 5M

part	triples stored	ms for all	ms for one	ms for last 100000	ms for one	Subjects	Relations	Objects
1	100000	250023	2.5	250023	2.5	5463	22	100000
2	200000	506660	2.5	256637	2.6	7973	22	200000
3	300000	751254	2.5	244594	2.4	10471	22	300000
4	400000	983196	2.5	231942	2.3	12974	22	400000
5	500000	1227104	2.5	243908	2.4	22353	29	500000
6	600000	1468063	2.4	240959	2.4	32357	29	600000
7	700000	1708663	2.4	240600	2.4	42360	29	700000
8	800000	1956034	2.4	247371	2.5	52363	29	800000
9	900000	2190644	2.4	234610	2.3	62366	29	900000
10	1000000	2430043	2.4	239399	2.4	72369	29	1000000
11	1100000	2666041	2.4	235998	2.4	82372	29	1100000
12	1200000	2910230	2.4	244189	2.4	92375	29	1200000
13	1300000	3143529	2.4	233299	2.3	102377	29	1300000
14	1400000	3371618	2.4	228089	2.3	112381	29	1400000
15	1500000	3605136	2.4	233518	2.3	122384	29	1500000
16	1600000	3838139	2.4	233003	2.3	132387	29	1600000
17	1700000	4070830	2.4	232691	2.3	142390	29	1700000
18	1800000	4298155	2.4	227325	2.3	152393	29	1800000

19	1900000	4527367	2.4	229212	2.3	162396	29	1900000
20	2000000	4758030	2.4	230663	2.3	172399	29	2000000
21	2100000	4985698	2.4	227668	2.3	182402	29	2100000
22	2200000	5212742	2.4	227044	2.3	192405	29	2200000
23	2300000	5439692	2.4	226950	2.3	202408	29	2300000
24	2400000	5685347	2.4	245655	2.5	212043	40	2400000
25	2500000	5922328	2.4	236981	2.4	221512	40	2500000
26	2600000	6155331	2.4	233003	2.3	230972	40	2600000
27	2700000	6391610	2.4	236279	2.4	240447	40	2700000
28	2800000	6630417	2.4	238807	2.4	249912	40	2800000
29	2900000	6855511	2.4	225094	2.3	259371	40	2900000
30	3000000	7078545	2.4	223034	2.2	268831	40	3000000
31	3100000	7305979	2.4	227434	2.3	278290	40	3100000
32	3200000	7533928	2.4	227949	2.3	287754	40	3200000
33	3300000	7773608	2.4	239680	2.4	297240	40	3300000
34	3400000	8006782	2.4	233174	2.3	306704	40	3400000
35	3500000	8239629	2.4	232847	2.3	316145	40	3500000
36	3600000	8464536	2.4	224907	2.2	325609	40	3600000
37	3700000	8693202	2.3	228666	2.3	335077	40	3700000
38	3800000	8919248	2.3	226046	2.3	344557	40	3800000
39	3900000	9150254	2.3	231006	2.3	354009	40	3900000
40	4000000	9383912	2.3	233658	2.3	363472	40	4000000
41	4100000	9616120	2.3	232208	2.3	372924	40	4100000
42	4200000	9850090	2.3	233970	2.3	382383	40	4200000
43	4300000	10073842	2.3	223752	2.2	391847	40	4300000
44	4400000	10305832	2.3	231990	2.3	401308	40	4400000
45	4500000	10536619	2.3	230787	2.3	410763	40	4500000
46	4600000	10769997	2.3	233378	2.3	420233	40	4600000
47	4700000	11004030	2.3	234033	2.3	429699	40	4700000

48	4800000	11242836	2.3	238806	2.4	439169	40	4800000
49	4900000	11474107	2.3	231271	2.3	448643	40	4900000
50	5000000	11702852	2.3	228745	2.3	458099	40	5000000
51	5000453	11704116	2.3	1264	2.8	458142	55	5000453

Benchmark configuration used by [Bizer & Schultz, 2008] is **Configuration B**.

Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is

$\mathbf{L_{(RDFArM,B,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{R_{RDFArM,K,B}}$, and $\mathbf{R_{RDFArM,K,B}} = \mathbf{0.025729}$;

and we compute final loading time as follow: 11704 * 0.025729= **301.13** sec.

Table 50. Benchmark results for BSBM 5M

system	**loading time in seconds**
Sesame	1988
Jena SDB	1053
Virtuoso	609
RDFArM	**301**

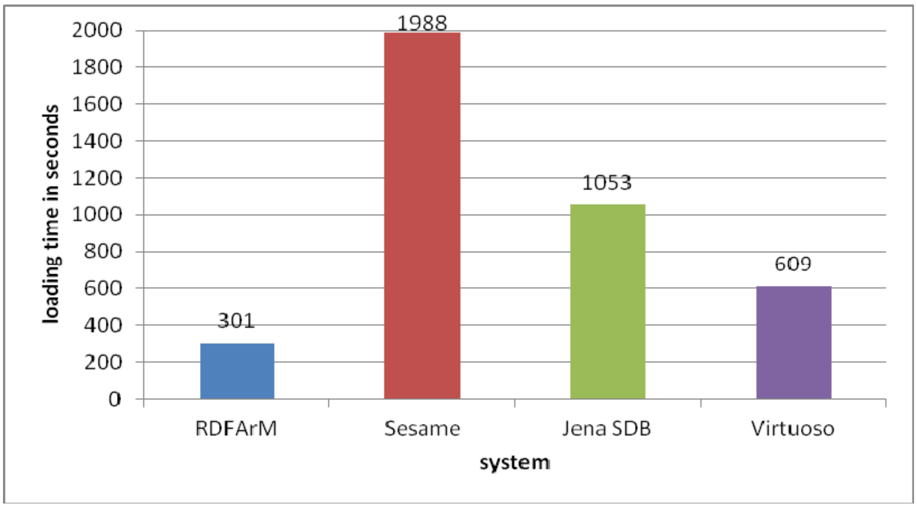


Figure 63. Benchmark results for BSBM 5M

From Table 50 and Figure 63 we may conclude that RDFArM has best loading time (better about 85% than Sesame, 71% than Jena, and 51% than Virtuoso).

6.5 Experiments with large datasets

We provided experiments with real large datasets which were taken from DBpedia's homepages [DBpedia, 2007c] and Billion Triple Challenge (BTC) 2012 [BTC, 2012].

The real dataset from DBpedia's *infoboxes-fixed.nt* (15,472,624 triples; 2.1 GB) is based on DBpedia's infoboxes.nt dated 2007-08-30 [DBpedia, 2007c]. 166 triples from the original set were excluded because they contained excessively large URIs *(> 500 characters)* that caused importing problems with Virtuoso (DBpedia bug #1871653). RDFArM has no such limitation. Infoboxes-fixed.nt was imported with indexes initially disabled in SDB and Virtuoso. Indexes were then activated and the time required for index creation time was factored into the import time. In the case with RDFArM no parameters are needed. The datasets were loaded directly from the source file.

The RDF Stores, tested by [Becker, 2008], are:

- OpenLink Virtuoso Open-Source Edition 5.0.2 [Virtuoso, 2013];
- Jena SDB Beta 1 on PostgreSQL 8.2.5 and MySQL 5.0.45 [Jena, 2013];
- Sesame 2.0 beta 6 [Sesame, 2012].

The RDF stores feature different indexing behaviors: Sesame automatically indexes after each import, while SDB and Virtuoso allow for selective index activation. More information about latest versions of these systems is given in Appendix B.

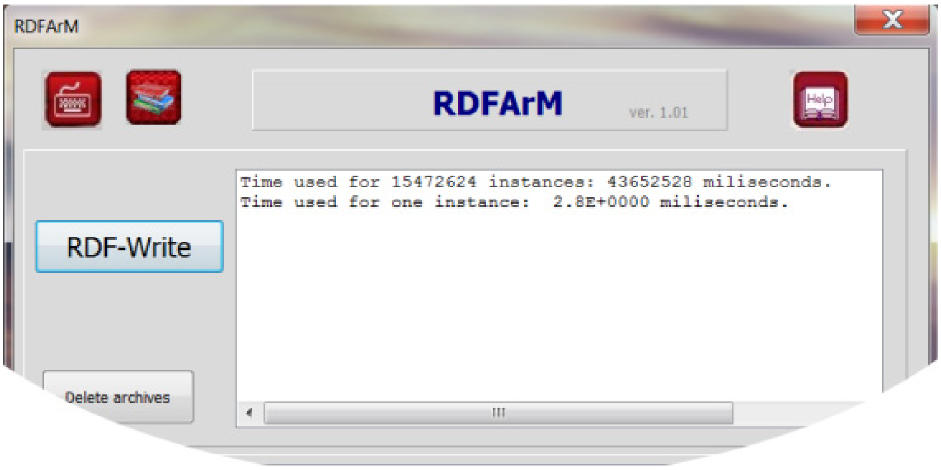
Artificial large datasets are taken from Berlin SPARQL Bench Mark (BSBM) [Bizer & Schultz, 2009; BSBMv3, 2009; BSBMv5, 2009; BSBMv6, 2011]. Details about the benchmark artificial datasets are summarized in the following Table 51:

Table 51. Details about artificial large RDF-datasets

Number of Triples	**25M**	**100M**
Exact Total Number of Triples	**25000244**	**100000112**
Number of Products	70812	284826
Number of Producers	1422	5618
Number of Product Features	23833	47884
Number of Product Types	731	2011
Number of Vendors	722	2854
Number of Offers	1416240	5696520
Number of Reviewers	36249	146054
Number of Reviews	708120	2848260
Total Number of Instances	2258129	9034027
File Size Turtle (unzipped)	*2.1 GB*	8.5 GB

✓ ***Loading of infoboxes-fixed.nt***

RDFArM has loaded all 15 472 624 triples from infoboxes-fixed.nt for about 43652 seconds (43652528 ms) or average time of 2.8 ms per triple (Figure 64).



```
Time used for 15472624 instances: 43652528 miliseconds.
Time used for one instance:  2.8E+0000 miliseconds.
```

Figure 64. Screenshot of the report of RDFArM for infoboxes-fixed.nt

More detailed information is given in Table 70 in Appendix A4. Every row of this table contains data for storing of one hundred thousand triples. Total stored triples were 15,472,624 and Table 70 contains 155 rows.

Number of Subjects in this dataset was S=1354298; number of relations R=56338; and number of objects O=15472624.

This means that practically we had 56338 layers with 1354298 NL-locations (containers) which contain 15472624 objects, i.e. some containers in some layers are empty. The loading time' results from our experiment and [Becker, 2008] are given in Table 52 and Figure 65.

Benchmark configuration used by [Becker, 2008] is **Configuration A**.

Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is
$L_{(RDFArM,A,S2)} = H_{AK}*L_{(RDFArM,K,S2)}$, where H_{AK} =3.125;
and we compute final loading time as follow: 43652 x 3.125= **136412.5** sec.

Table 52. Benchmark results for infoboxes-fixed.nt

system	loading time in seconds
Virtuoso (ogps, pogs, psog, sopg)	7017
Jena SDB MySQL Layout 2 Index	70851
Jena SDB Postgre SQL Layout 2 Index	73199
Jena SDB Postgre SQL Layout 2 Hash	734285
Sesame Native (spoc, posc)	21896
RDFArM	**136412**

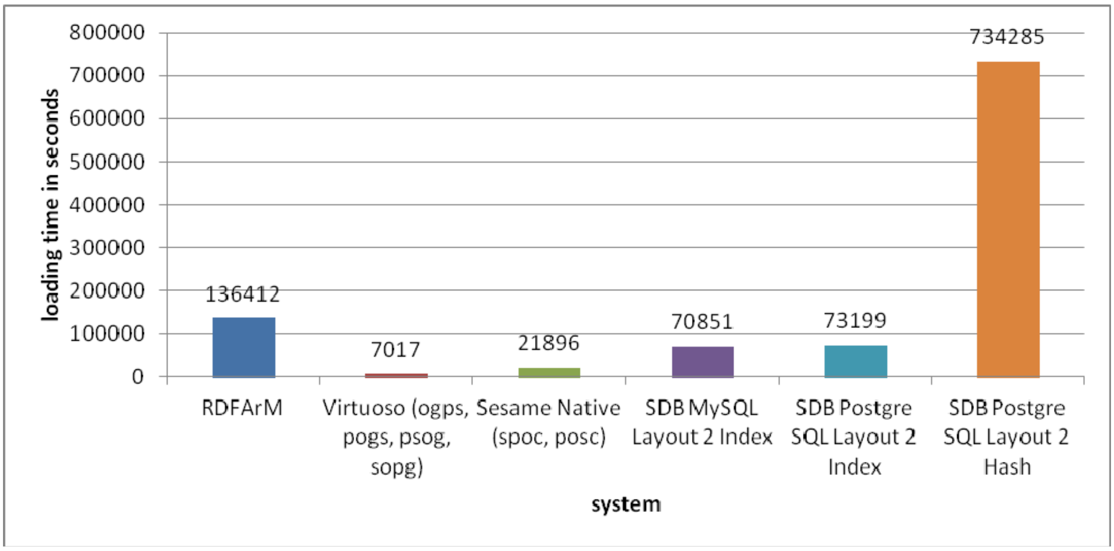


Figure 65. Benchmark results for infoboxes-fixed.nt

From Table 52 and Figure 65 we may conclude that RDFArM has the worst loading time. Virtuoso is 95%, Sesame is 84%, and Jena is 48% better than RDFArM (we take in account only the best results of compared systems).

For large datasets it is very important to support multi-processors' parallel loading of data. RDFArM is developed to support such work. We simulate four processors' configuration by separating dataset on portions of 5 million triples and loading them separately (Table 70).

Table 53 presents final times for different processors.

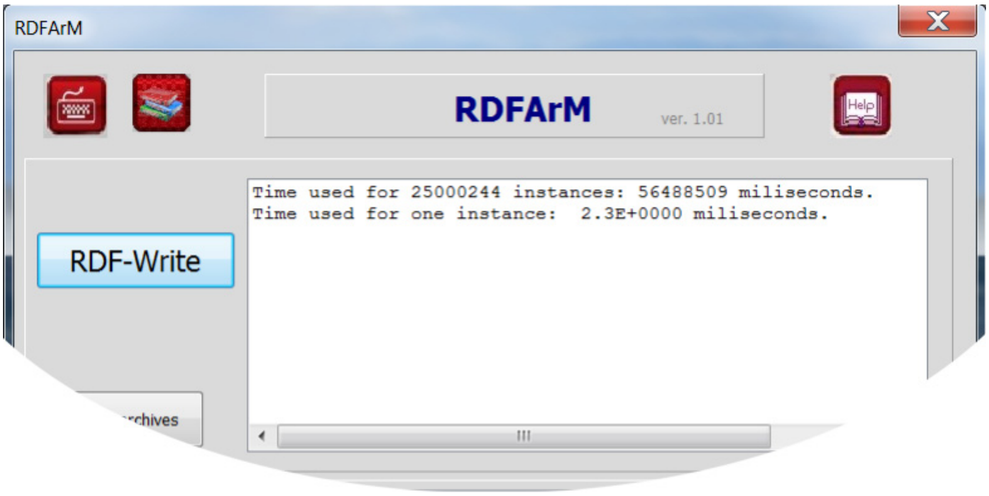
Table 53. Benchmark results for multiprocessor loading of infoboxes-fixed.nt

processor number	triples stored	ms for storing all triples	ms for storing one triple
0	5000000	13394043	2.7
1	5000000	15054986	3.01
2	5000000	14182083	2.8
3	472624	1021416	2.2

As total loading time we assume the largest processor's one, i.e. **15054.986 sec**.

✓ *Loading of BSBM 25M*

RDFArM has loaded all **25000244** triples from **BSBM 25M** for about **56488** seconds (56488509ms) or average time of **2.3** ms per triple (Figure 66).



```
Time used for 25000244 instances: 56488509 miliseconds.
Time used for one instance:  2.3E+0000 miliseconds.
```

Figure 66. Screenshot of the report of RDFArM for BSBM 25M

Number of Subjects in this dataset was S=2258132; number of relations R=112; and number of objects O=25000244.

This means that practically we had 112 layers with 2258132 NL-locations (containers) which contain 25000244 objects, i.e. some containers in some layers are empty.

The loading time' results from our experiment and [Bizer & Schultz, 2009; BSBMv3, 2009] are given in Table 54 and Figure 67.

Benchmark configuration used by [Bizer & Schultz, 2009; BSBMv3, 2009] is **Configuration B**. Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is

$\mathbf{L_{(RDFArM,B,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{R_{RDFArM,K,B}}$, and $\mathbf{R_{RDFArM,K,B}} = \mathbf{0.025729}$.

We compute final loading time as follow: 56488*0.025729= **1453.38** sec.

Table 54. Benchmark results for BSBM 25M

system	loading time in seconds
Sesame	44225
Jena TDB	1013
Jena SDB	14678
Virtuoso TS	2364
Virtuoso RV	1035
Virtuoso SQL	1035
RDFArM	**1453**

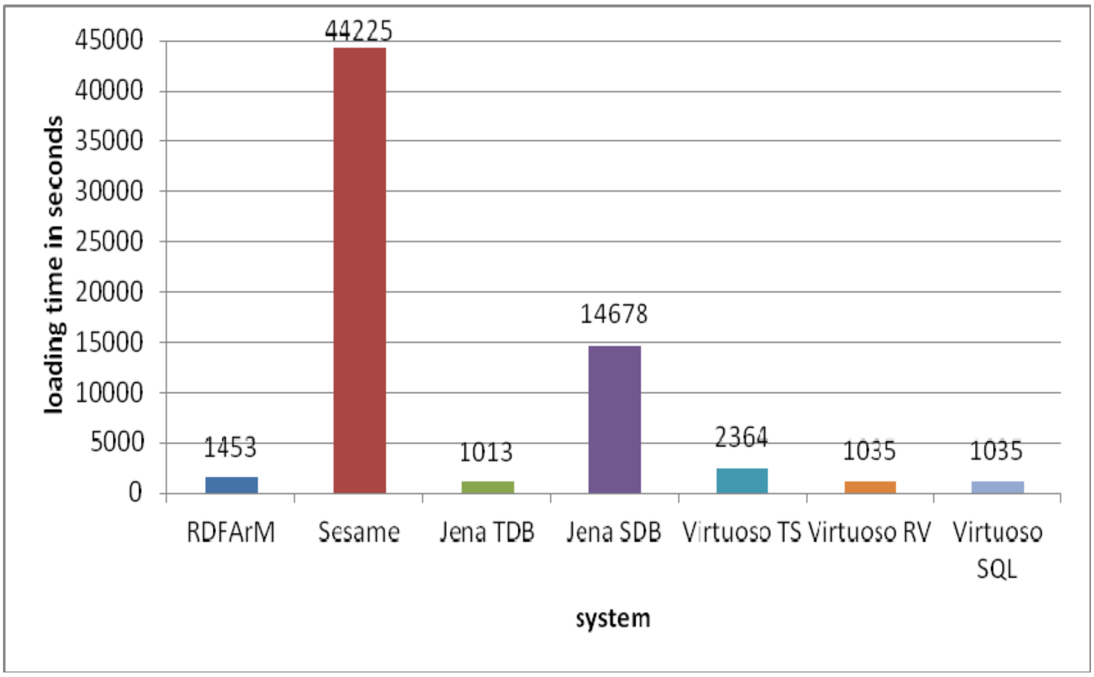


Figure 67. Benchmark results for BSBM 25M

From Table 54 and Figure 67 we may conclude that Jena (with 30%) and Virtuoso (with 29%) are better than RDFArM. RDFArM has 97% better performance than Sesame.

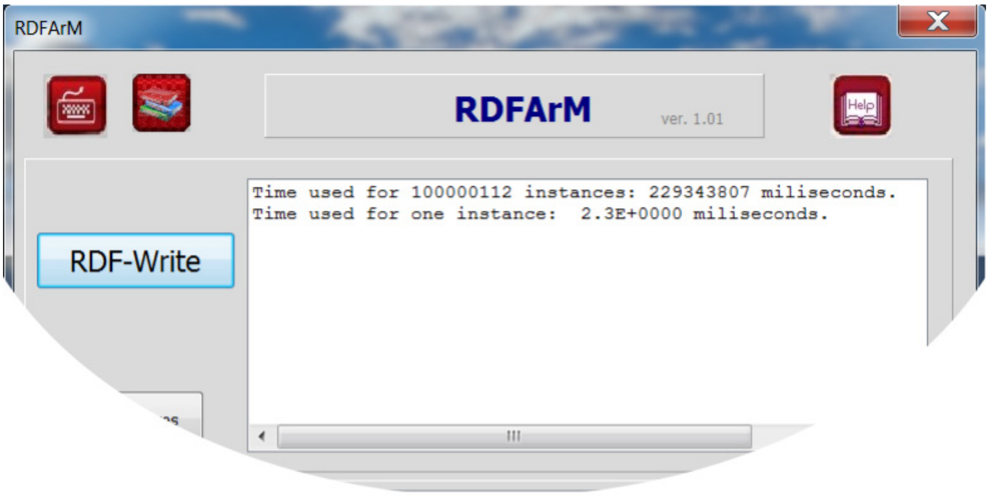
We simulate multi-processors' configuration by separating dataset on portions of 5 million triples and loading them separately. Table 55 presents final times for different processors.

Table 55. Benchmark results for multiprocessors' loading of BSBM 25M

processor number	triples stored	ms for storing all triples	ms for storing one triple
0	5000000	11353862	2.3
1	5000000	11570875	2.3
2	5000000	11529651	2.3
3	5000000	11107771	2.2
4	5000000	10925646	2.2
5	244	704	2.9

✓ *Loading of BSBM 100M and BSBM 200M*

RDFArM has loaded all **100000112** triples from **BSBM 100M** for about **229344** seconds (229343807 ms) or average time of **2.3** ms per triple (Figure 68).



```
Time used for 100000112 instances: 229343807 miliseconds.
Time used for one instance:  2.3E+0000 miliseconds.
```

Figure 68. Screenshot of the report of RDFArM for BSBM 100M

Number of Subjects in this dataset was S=9034046; number of relations R=341; and number of objects O=100000112.

This means that practically we had 341 layers with 9034046 NL-locations (containers) which contain 100000112 objects, i.e. some containers in some layers contain more than one object. The loading time' results from our experiment and [Bizer & Schultz, 2009; BSBMv3, 2009] are given in Table 56 and Figure 69.

Benchmark configuration used by [Bizer & Schultz, 2009; BSBMv3, 2009] is **Configuration B**. Our benchmark configuration is **Configuration K**.

The loading times proportionality formula is

$$\mathbf{L}_{(\mathbf{RDFArM,B,S2})} = \mathbf{L}_{(\mathbf{RDFArM,K,S2})} * \mathbf{R}_{\mathbf{RDFArM,K,B}}, \text{ and } \mathbf{R}_{\mathbf{RDFArM,K,B}} = \mathbf{0.025729}.$$

We compute final loading time as follow:

$$229344 * 0.025729 = \mathbf{5900.79} \text{ sec.}$$

Table 56. Benchmark results for BSBM 100M

system	loading time in seconds
Sesame	282455
Jena TDB	5654
Jena SDB	139988
Virtuoso TS	28607
Virtuoso RV	3833
Virtuoso SQL	3833
RDFArM	**5901**

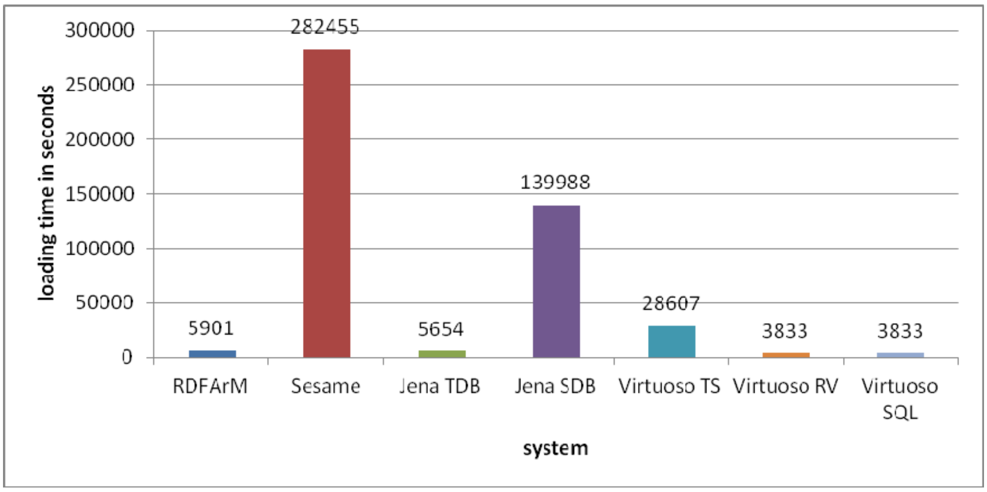


Figure 69. Benchmark results for BSBM 100M

From Table 56 and Figure 69 we may conclude that Virtuoso is 35% better than RDFArM and Jena is 4% better than RDFArM. RDFArM is 98% better than Sesame.

In [BSBMv6, 2011] results received from benchmarks on computer configuration C are published. The N-Triples version of the dataset was used. For Virtuoso, the dataset was split into 100 respectively 200 Turtle files and loaded with the DB.DBA.TTLP function consecutively.

Benchmark configuration used by [BSBMv6, 2011] is **Configuration C**.

Our benchmark configuration is **Configuration K**.

The loading times proportionality formula for **Configuration C** is

$\mathbf{L_{(RDFArM,C,S2)}} = \mathbf{L_{(RDFArM,K,S2)}} * \mathbf{R_{RDFArM,K,C}}$ and $\mathbf{R_{RDFArM,K,C}} = \mathbf{0.068639}$.

We compute RDFArM final loading time for BSBM 100M as follow:

229344 * 0.068639 = **15741.94** sec.

We compute RDFArM final loading time for BSBM 200M as follow:

2 * 229344 * 0.068639 = **31483.88** sec.

The loading time' results from our experiment and [BSBMv6, 2011] are given in Table 57 and Figure 70.

Table 57. Benchmark results for BSBM 100M and 200M on Configuration C

system	loading time in seconds	
	100M	**200M**
Jena TDB	4488	9913
Virtuoso	6566	14378
RDFArM	**15742**	**31484**

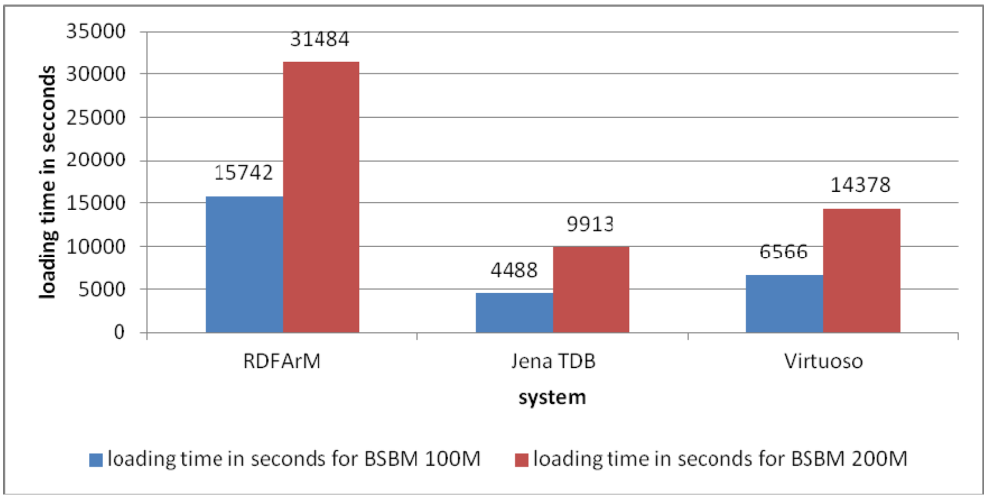


Figure 70. Benchmark results for BSBM 100M and 200M on Configuration C

We have no benchmarks for Sesame. Because of this experiment will not be used for the analysis. From Table 57 and Figure 70 we may conclude that RDFArM has to be improved for big datasets to be comparable to Virtuoso and Jena. This is done in its multi-processors' version RDFArM-MP.

We simulate multi-processors' configuration by separating dataset on portions of 5 million triples and loading them separately. Table 58 presents final times for different processors.

Table 58. Benchmark results for multiprocessors' loading of BSBM 100M

processor number	triples stored	ms for storing all triples	ms for storing one triple
0	5000000	11271727	2.3
1	5000000	11251369	2.3
2	5000000	11514715	2.3
3	5000000	11318091	2.3
4	5000000	11484496	2.3
5	5000000	11571904	2.3
6	5000000	11524854	2.3
7	5000000	11541593	2.3
8	5000000	11547395	2.3
9	5000000	11582902	2.3
10	5000000	11508093	2.3
11	5000000	11461596	2.3

12	5000000	11588535	2.3
13	5000000	11597551	2.3
14	5000000	11565899	2.3
15	5000000	11367450	2.3
16	5000000	11379821	2.3
17	5000000	11337077	2.3
18	5000000	11420647	2.3
19	5000000	11507826	2.3
20	112	266	2.4

As total loading time of multi-processors' configuration we assume the largest processor's time, i.e. 11597.551 sec.

We compute final loading time as follow: 11597.551 * 0.068639 = **796.04** sec.

➢ *Conclusion of chapter 6*

We have presented results from series of experiments which were needed to estimate the storing time of NL-addressing for middle-size and very large RDF-datasets.

We described the experimental storing models and special algorithm for NL-storing RDF instances. Estimation of experimental systems was provided to make different configurations comparable. Special proportionality constants for hardware and software were proposed. Using proportionality constants, experiments with middle-size and large datasets become comparable.

Experiments were provided with both real and artificial datasets. Experimental results were systematized in corresponded tables. For easy reading visualization by histograms was given.

Experimental results will be analyzed in the next chapter.

The goal experiments for NL-storing of middle-size and large RDF-datasets were to estimate possible further development of NL-ArM. We assumed that its "software growth" will be done in the same grade as one of the known systems like Virtuoso, Jena, and Sesame. In the next chapter we will analyze what will be the place of NL-ArM in this environment but already we may see that NL-addressing have good performance and NL-ArM has similar results as Jena and Sesame.

7 Analysis of experiments

Abstract

In this research we have provided series of experiments to identify any trends, relationships and patterns in connection to NL-addressing and its implementations. We have realized three types of experiments:

- *Basic experiments to compare our access method with two main types of organization and access to information – sequential and relational;*
- *Experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of structured datasets;*
- *Experiments to examine the applicability of NL-addressing for middle-size and very large RDF-datasets.*

Below we will analyze these experiments. Special attention will be paid to analyzing of storing times of NL-ArM access method and its possibilities for multi-processing.

7.1 Analysis of basic experiments

We started our experiments in *Chapter 4* with two main types of basic experiments: NL-ArM has been compared with:

- Sequential text file of records;
- Relational data base management system Firebird.

The need to compare NL-ArM access method with text files was determined by practical considerations – in many applications the text files are main approach for storing information. To investigate the size of files and speed of their generation we compared writing in a sequential text file and in a NL-ArM archive.

The experiments have sown:

- NL-ArM is constantly about two and half times slower than writing in sequential text file because of building the hash tables in the NL-ArM archive. This means that including new records in NL-ArM archive take the same time (about 0.013 ms) per record irrespective of the number of already stored records;
- The comparison of file sizes showed that, for great number of elements, text file became longer than NL-ArM archive. This leads to the following conclusion - bulky text files whose records are attached to long “keywords” would be saved more compactly in

NL-ArM archives instead as records in text files with explicitly given keywords in each record.

Important consideration in this case was that reading sequential text file to find concrete keyword is very slow operation. Every indexed approach is quicker. Indexed text files are typical for databases and this case was analyzed in experiments with a database.

To provide experiments with a relational database, we have chosen the system "Firebird". It should be noted that Firebird and NL-ArM have fundamentally different physical organization of data and the tests cover small field of features of both systems.

Firebird is not access method but system for managing databases. It is important that all queries in Firebird are in SQL and interpreted by the system, which requires appropriate time. The primary goal for new Version 2.5 of Firebird is to establish the basics for a new threading architecture which will speed up the Firebird on multi-processors' computer systems [Firebird, 2013].

It is important to underline that the experiments were provided for data with fixed length. If keywords' length is variable, we will have problems to work with any RDBMS. NL-ArM supports variable length of the keywords.

Calculating the hash function is faster than searching keywords in the index. But searching operations in hash structures is slow operation; in this case search in index structures is more convenient.

Another disadvantage of NL-ArM is connected to its special type of realization of the internal index structure. In relational model all keys have same influence on the writing time – they are written in the plain file by the same manner (as parts of records) and extend the balanced index in one or other its section which takes logarithmic time. In NL-ArM the different values of co-ordinates cause various archive structures which take corresponded time. Practically, NL-ArM creates hyper-matrix and large empty zones need additional resources – time and disk space, which are not so great due to smart internal index organization but really exists.

Experiments have shown that regarding NL-ArM:

- In writing, Firebird is on average **90.1 times slower**. This is due to two reasons:
  1. Balanced indexes of Firebird need reconstruction for including of every new keyword. This is time consuming process;
  2. The speed of updating NL-ArM hash tables which do not need recompilation after including new information.

  Due to specific of realization, for small values of co-ordinates NL-ArM is not effective as for the great ones. Nevertheless, NL-ArM is always many times faster than Firebird.
- In reading, Firebird is on average **29.8 times slower** due to avoiding search operations in NL-ArM hash tables which speeds the access.

Finally, for large dynamic data sets more convenient are hash based tools like NL-ArM because of random direct access to all stored records immediately after writing it without any additional indexing. For storing big semi-structured datasets like large ontologies and RDF-graphs this advantage is crucial.

7.2 Analysis of experiments with structured datasets

In Chapter 5 we have presented several experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of structured datasets.

We have provided three main experiments - for NL-storing of dictionaries, thesauruses, and ontologies.

Analyzing results from the experiment with a real dictionary data we may conclude that it is possible to use NL-addressing for storing such information.

Next experiment was aimed to answer to question: "What we gain and loss using NL-Addressing for storing thesauruses?"

Analyzing results from the experiment we point that the loss is additional memory for storing hash structures which serve NL-addressing. But the same if no great losses we will have if we will build balanced search trees or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published. The benefit is in two main achievements:

- High speed for storing and accessing the information;
- The possibility to access the information immediately after storing without recompilation the database and rebuilding the indexes.

The third experiment considered the complex graph structures such as ontologies. The presented survey of the state of the art in this area has shown that main models for storing ontologies are sequential files and relational databases (i.e. sets of interconnected indexed sequential files with fixed records' structure).

Our experiment confirmed the conclusion about losses and benefits from using NL-addressing given above for thesauruses. The same is valid for more complex structures.

For static structured datasets it is more convenient to use standard utilities and complicated indexes. NL-addressing is suitable for dynamic processes of creating and further development of structured datasets due to avoiding recompilation of the database index structures and high speed access to every data element.

7.3 Analysis of experiments with semi-structured datasets

In Chapter 6 the applicability of NL-addressing for middle-size and large semi-structured RDF-datasets was concerned.

We have provided twelve experiments with middle-size and large RDF-datasets, based on selected datasets from DBpedia's homepages and Berlin SPARQL Bench Mark (BSBM) to make comparison with published benchmarks of known RDF triple stores.

➢ *Rank-based multiple comparison*

We will use the Friedman test to detect statistically significant differences between the systems [Friedman, 1940]. The Friedman test is a non-parametric test, based on the ranking of the systems on each dataset. It is equivalent of the repeated-measures ANOVA [Fisher, 1973]. We will use Average Ranks ranking method, which is a simple ranking method, inspired by Friedman's statistic [Neave & Worthington, 1992]. For each dataset the systems are ordered according to the time measures and are assigned ranks accordingly. The best system receives rank 1, the second – 2, etc. If two or more systems have equal value, they receive equal rank which is mean of the virtual positions that had to receive such number of systems if they were ordered consecutively each by other.

Let *n* is the number of observed datasets; *k* is the number of systems.

Let i_{rj} be the rank of system *j* on dataset *i*. The average rank for each system is calculated as

$$R_j = \frac{1}{n}\sum_{i=1}^{k} r_j^i \cdot$$

The null-hypothesis states that if all the systems are equivalent than their ranks R_j should be equal. When null-hypothesis is rejected, we can proceed with the Nemenyi test [Nemenyi, 1963] which is used when all systems are compared to each other. The performance of two systems is significantly different if the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$. Some of the values of q_α are given in Table 59 [Demsar, 2006].

Table 59. Critical values for the two-tailed Nemenyi test

systems	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\mathbf{q_{0.05}}$	1.960	2.343	2.569	2.728	2.850	2.949	3.031	3.102	3.164
$\mathbf{q_{0.10}}$	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920

The results of the Nemenyi test are shown by means of critical difference diagrams.

Experiments which we will take in account were presented in corresponded tables of Chapter 6 as follow (Table 60):

Table 60. Information about tests and results

test No:	**results**
1	Table 40
2	Table 42
3	Table 44
4	Table 46
5a	Table 48 (a)
5b	Table 48 (b)

6	Table 50
7	Table 52
8	Table 54
9	Table 56
10a	Table 57 (a)
10b	Table 57 (b)

Benchmark values from our 12 experiments and corresponded published experimental data from BSBM team are given in Table 61. Published results do not cover all table, i.e. we have no values for some cells. To solve this problem we will take in account only the best result for given system on concrete datasets (Table 62). Sesame had no average values for tests 10a and 10b. Because of this we will not use these test in our comparison. They were useful to see the need of further refinement of RDFArM for big data.

The ranks of the systems for the ten tests are presented below in Table 63.

Table 61. Benchmark values for middle size datasets

system	**TEST**											
	1	**2**	**3**	**4**	**5a**	**5b**	**6**	**7**	**8**	**9**	**10a**	**10b**
RDFArM	3	2272	14.79	3469	60	60	301	136412	1453	5901	15742	31484
Sesame Native (spoc, posc)	3	2404	19	2341	179	213	1988	21896	44225	282455		
Virtuoso (ogps, pogs, psog, sopg)	2	1327		1235			609	7017			6566	14378
Virtuoso TS			05		23	25			2364	28607		
Virtuoso RDF views			09									
Virtuoso SQL			09		34	33			1035	3833		
Virtuoso RV					34	33			1035	3833		
Jena SDB	5		13		129		1053		14678	139988		
Jena TDB					49	41			1013	5654	4488	9913
Jena SDB MySQL Layout 2 Index		5245		6290				70851				
Jena SDB Postgre SQL Layout 2 Hash		3557		3305				73199				
Jena SDB Postgre SQL Layout 2 Index		9681		9640				734285				

Table 62. Chosen benchmark values for middle size datasets

system	TEST											
	1	2	3	4	5a	5b	6	7	8	9	10a	10b
RDFArM	3	2272	14.79	3469	60	60	301	136412	1453	5901	15742	31484
Sesame	3	2404	19	2341	179	213	1988	21896	44225	282455		
Virtuoso	2	1327	05	1235	23	25	609	7017	1035	3833	6566	14378
Jena	5	3557	13	3305	49	41	1053	70851	1013	5654	4488	9913

Table 63. Ranking of tested systems

system	ranks for the tests										average rank
	1	2	3	4	5a	5b	6	7	8	9	
RDFArM	2.5	2	3	4	3	3	1	4	3	3	**2.85**
Sesame	2.5	3	4	2	4	4	4	2	4	4	**3.35**
Virtuoso	1	1	1	1	1	1	2	1	2	1	**1.2**
Jena	4	4	2	3	2	2	3	3	1	2	**2.6**

All average ranks are different. The null-hypothesis is rejected and we can proceed with the Nemenyi test. Following [Demsar, 2006], we may compute the critical difference by formula:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

where q_α we take as $q_{0.10}$ = 2.291 (from Table 59 [Demsar, 2006; Table 5a]); *k* will be the number of systems compared, i.e. *k*=4; N will be the number of datasets used in benchmarks, i.e. N=10. This way we have:

$$CD_{0.10} = 2.291 * \sqrt{\frac{4*5}{6*10}} = 2.291 * \sqrt{\frac{20}{60}} = 2.291 * 0.577 = 1.322$$

We will use for critical difference $CD_{0.10}$ the value 1.322.

At the end, average ranks of the systems and distance to average rank of the first one are shown in Table 64.

Table 64. Average ranks of systems and distance to average rank of the first one

place	system	average rank	Distance between average rank of the system and average rank of the first one
1	**Virtuoso**	**1.2**	**0**
2	**Jena**	**2.6**	**1.4**
3	**RDFArM**	**2.85**	**1.65**
4	**Sesame**	**3.35**	**2.15**

The visualization of Nemenyi test results for tested systems is shown on Figure 71.

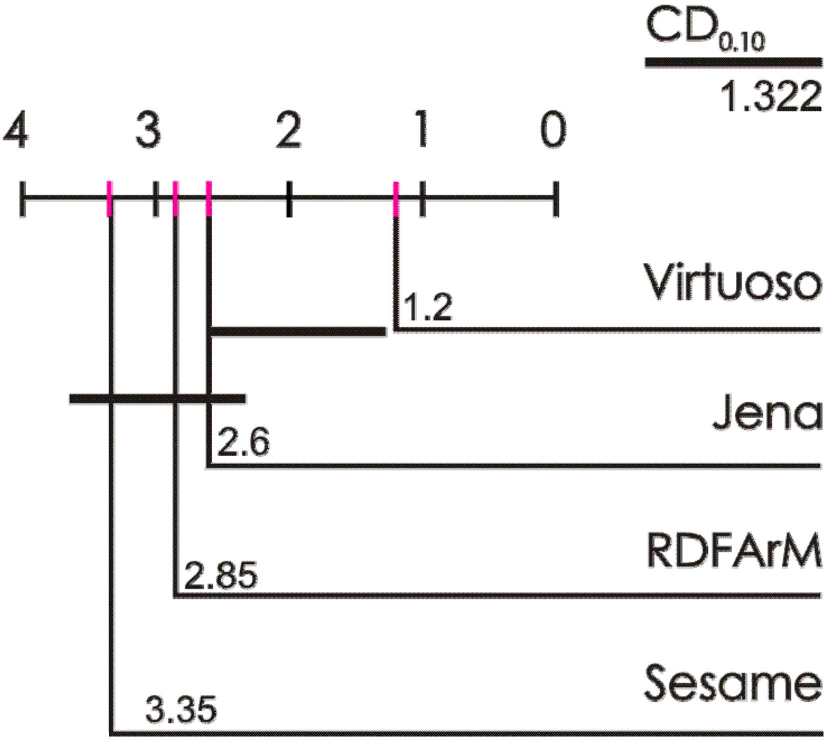


Figure 71. Visualization of Nemenyi test results

Analyzing these experiments we may conclude that RDFArM is at critical distances to Jena and Sesame. RDFArM is nearer to Jena than to Sesame. RDFArM, Jena, and Sesame are significantly different from Virtuoso.

Some recommendations to RDFArM may be given. RDF triple datasets has different characteristics depending of their origination. This causes the need to adapt NL-ArM storage engine to specifics of concrete datasets. For instance, important parameters are length of strings and quantity of repeating values of subject, relation, and object.

7.4 Storing time and multi-processing

The NL-ArM characteristic, which we will analyze now, is NL-access time dependence on growing of dataset size and possibility for multi-processing. Below we will outline data for NL-storing instances with one-, two- and three-elements. Graphical illustrations will be given for loading selected datasets. For two datasets will be given graphical comparison between times consumed by different processors.

✓ *NL-storing two-element instances*

Two-element instances we use in experiments for NL-storing dictionaries (Table 25) and thesauruses (Table 27). Two-element instance is couple: *(name, value)*, where “name” is “one-dimension” co-ordinate of NL-location where “value” (a string) has to be stored.

Measured times are gathered in Table 65.

Table 65. Access times for two-element instances

dataset	number of instances	time for all instances in ms	time for one instance in ms
NL-writing of one-element instances			
SA dictionary	23412	22105	0.94
WordNet as thesaurus	125062	107157	0.86
Total:	**148474**	**129262**	**0.87**
NL-reading of one-element instances			
SA dictionary	23412	20826	0.89
WordNet thesaurus	117641	91339	0.78
Total:	**141053**	**112165**	**0.79**

The average times are **0.87 ms** for writing and **0.79 ms** for reading.

➢ *NL-storing instances with three elements (RDF-triple datasets)*

Three-element instances we use in experiments for NL-storing RDF-triple datasets. Three-element instance is triple: *(subject, relation, object)*, where “subject” and “relation” are NL-locations, and “object” is a string to be stored at NL-location given by two-dimensional co-ordinates *(subject, relation)* where we store only one value (object) which is a string.

Measured times are gathered in Table 66.

Table 66. Loading times for three-element instances

dataset	number of triples	loading time in ms	
		for all triples	for one triple
BSBM 50K	50116	112851	2.3
homepages-fixed.nt	200036	727339	3.6
BSBM 250K	250030	575069	2.3
geocoordinates-fixed.nt	447517	1110415	2.5
BSBM 1M	1000313	2349328	2.3
BSBM 5M	5000453	11704116	2.3
infoboxes-fixed.nt	15472624	43652528	2.8
BSBM 25M	25000244	56488509	2.3
BSBM 100M	100000112	229343807	2.3
total:	**147421445**	**346063962**	**2.34745**

The average loading time is **2.35 ms**.

➢ *NL-storing four-element instances (RDF-quadruple datasets)*

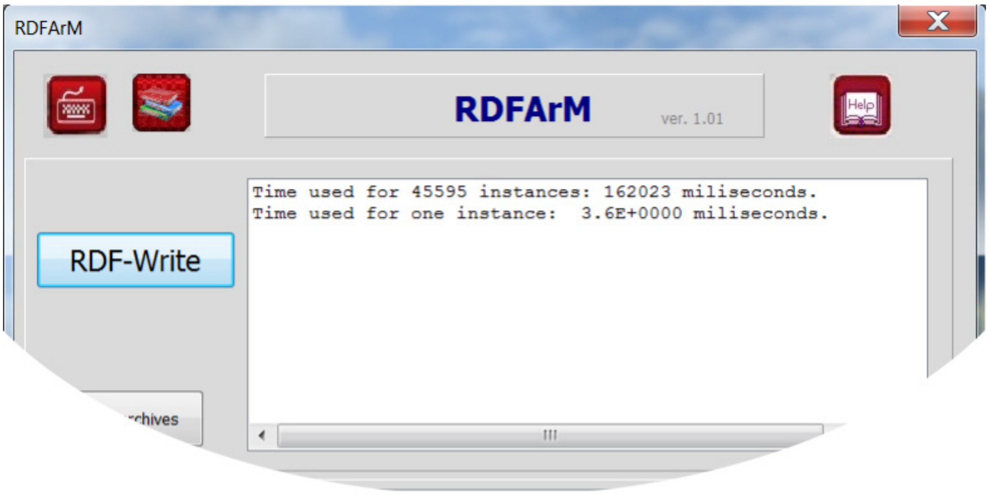
We done series of experiments based on real data from the BTC datasets [BTC, 2012]. Here we will outline only results for dataset http://km.aifb.kit.edu/projects/btc-2012/datahub/data-0.nq.gz.

Dataset “data-0.nq” contains 45595 quadruples. Information about its structure and results from the experiments with it are shown in Table 67. A screenshot from the RDFArM program is shown at Figure 72.

The average loading time is **3.6 ms**.

Table 67. Results for storing datahub/data-0.nq

number of				storing time for all instances in ms	storing time for one instance in ms
instances	subjects	relations	contexts		
45595	7325	894	101	162023	**3.6**



```
Time used for 45595 instances: 162023 miliseconds.
Time used for one instance:  3.6E+0000 miliseconds.
```

Figure 72. A screenshot from the RDFArM program

The time for NL-storing on **Configuration K** is about:

– 0.7 ms for two-element instances;

– 2.35 ms for three-element instances;

– 3.6 ms for four-element instances.

The conclusion is that every new element of the instance takes about one millisecond additional time. This means that ***NL-storing time is depended on the number of elements*** in the instances.

➢ *Graphical illustrations*

Graphical illustrations of loading selected datasets are given below.

Firstly (Figure 73), we show the graphic of time used for storing of one instance from BSBM 250K dataset. At the beginning RDFArM takes more time due to initialization of the hash structures.

The next graphic (Figure 74) illustrates the variation of storing time due to specifics of the dataset, i.e. the size of elements to be used as NL-addresses – as long are the strings of subject and relation, so long time it takes the string of object to be stored in the archive.

Nevertheless, storing time varies between 2.2 and 2.5 milliseconds.

The next two graphics are aimed to illustrate independence from size of the datasets. On Figure 75 the storing times of BSBM 25M dataset is shown, and on Figure 76 storing times of BSBM 100M dataset are illustrated. On both graphics we see the same regularity – constant time for storing of one triple independently of the number of already stored ones.

Simulating multi-processors work we are interested of regularity of used times from different processors. It is expectable to have similar times because of constant complexity of NL-addressing. This is seen of Figure 77 and Figure 78 where graphical comparison between times consumed by different processors is given.

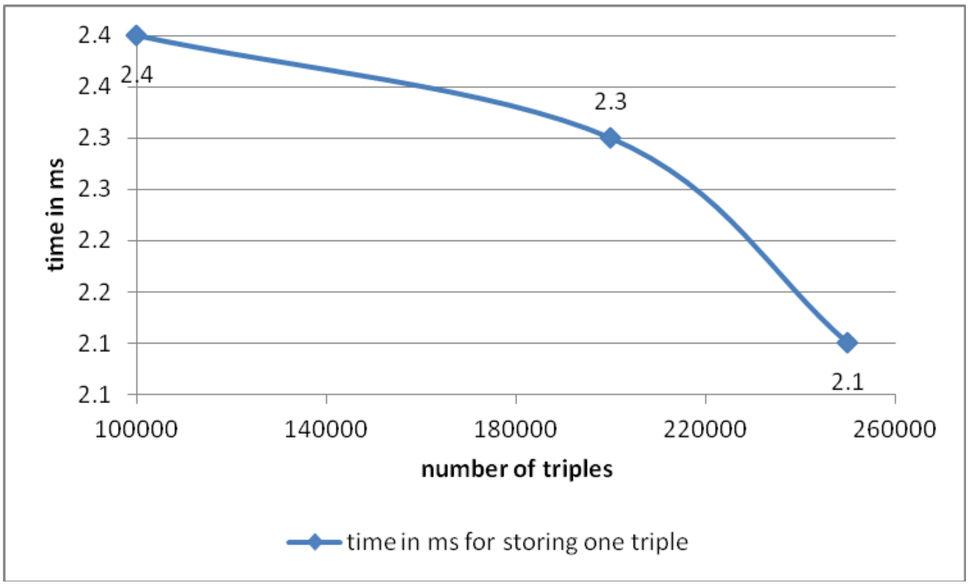


Figure 73. Storing time for one instance of BSBM 250K

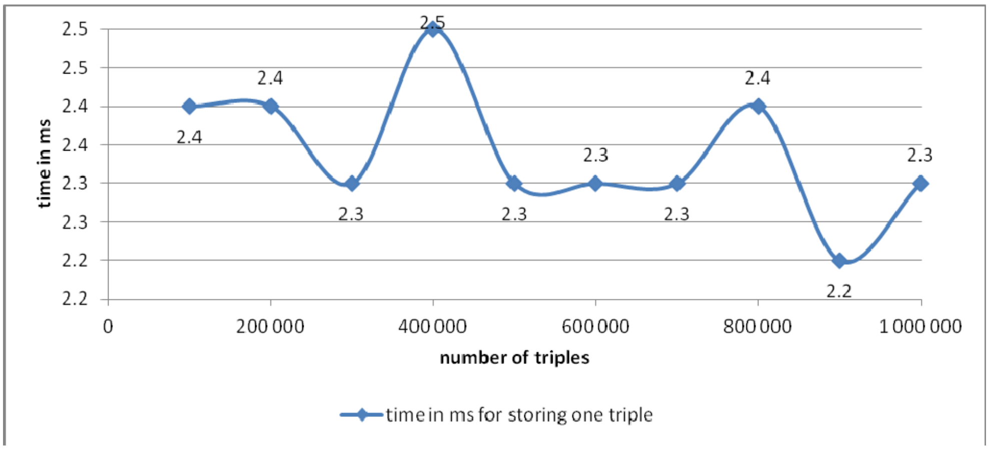


Figure 74. Storing time for one instance of BSBM 1M

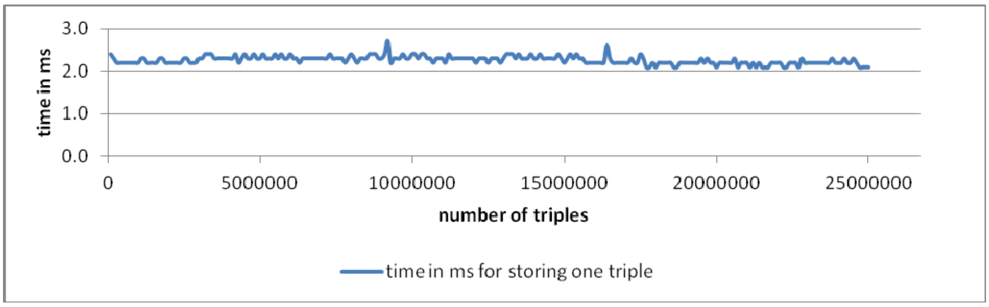


Figure 75. Storing time for one instance of BSBM 25M

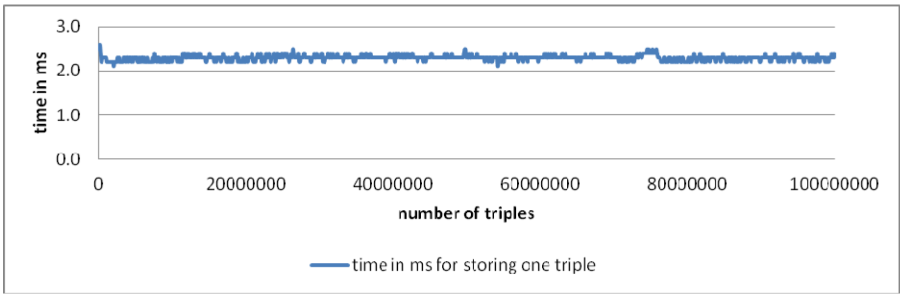


Figure 76. Storing time for one instance of BSBM 100M

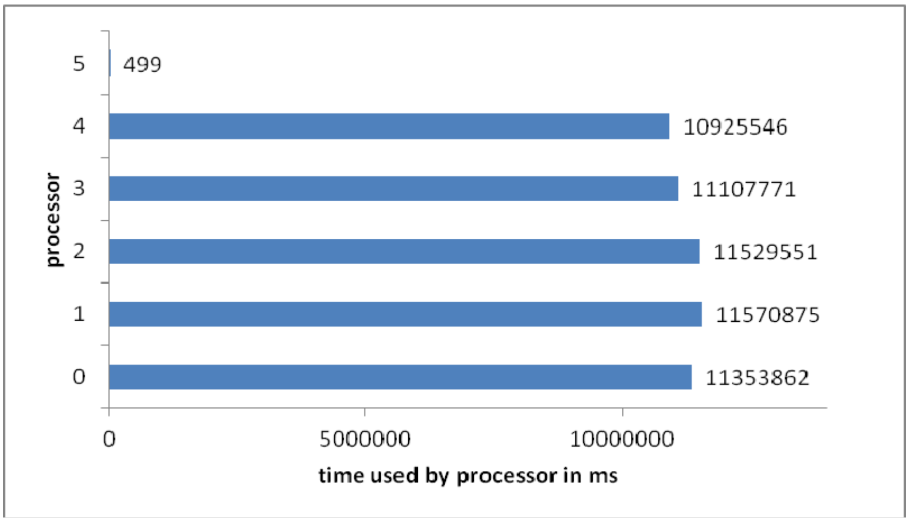


Figure 77. Comparison of time used by processors for BSBM 25M

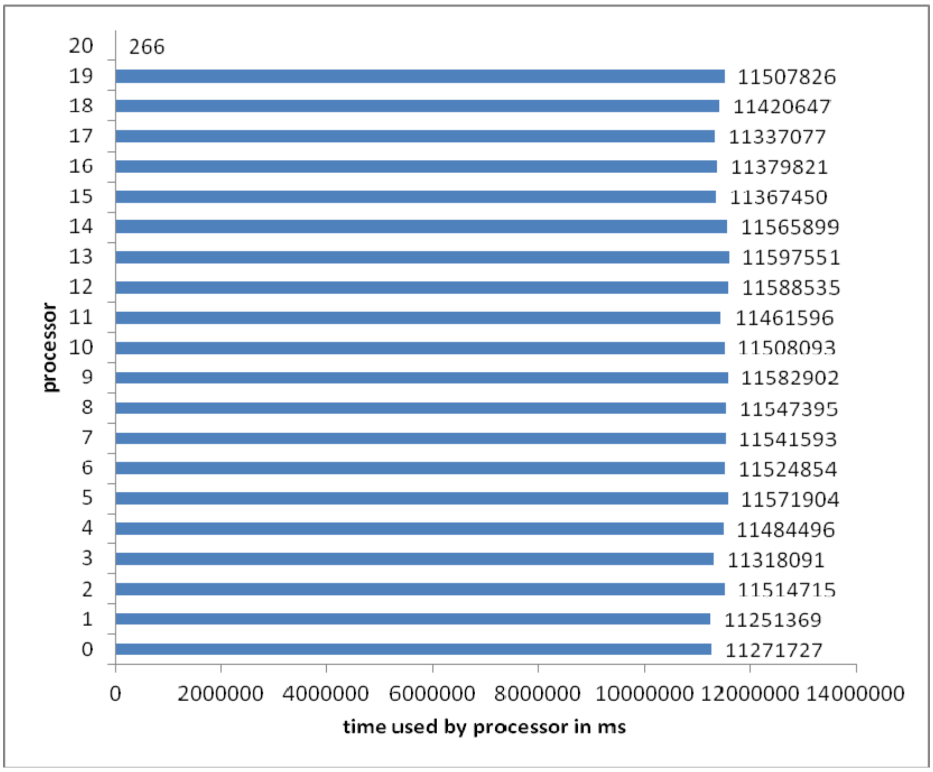


Figure 78. Comparison of time used by processors for BSBM 100M

From experimental data and visualizations we may conclude that the NL-access time:

- ***Depends on number of elements in a dataset's instances***, which have to be stored on the disk;
- ***Not depends on number of instances*** in the dataset.

The second is very important for multi-processing because it means linear reverse dependence on number of processors.

This time does not depend on the size of data sets, i.e. of the number of instances. Detailed information for storing 100 millions triples is given in the Appendix A.5 (Table 71). Table 71 contains results from an experiment for storing 100 millions triples from BSBM 100M [BSBMv3, 2009]. The check points were on every 100 000 triples. For every check point in Table 71, the average time in ms for writing one triple is shown. For comparison, the corresponded value of log n is given in third column. Data from Table 71 are visualized in Figure 79.

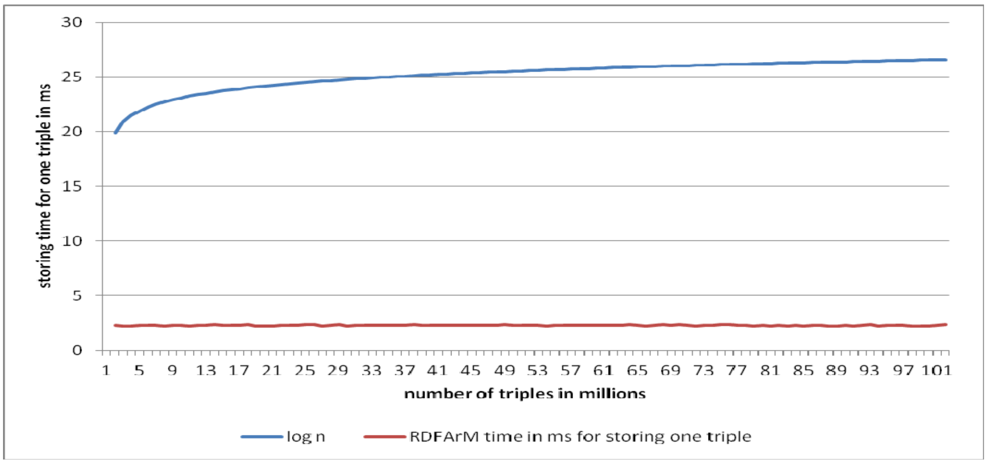


Figure 79. Comparison of log n and average time in ms for storing one triple from BSBM 100M

➢ *Conclusion of chapter 7*

In this chapter we have analyzed experiments presented in previous chapters 4, 5, and 6, which contain respectively results from (1) basic experiments; (2) experiments with structured datasets; (3) experiments with semi-structured datasets. Special attention was paid to analyzing of storing times of NL-ArM access method and its possibilities for multi-processing.

From experimental data and visualizations we concluded that the NL-access time:

- *Depends on number of elements in a dataset's instances, which have to be stored on the disk;*
- *Not depends on number of instances in the dataset.*

The second is very important for multi-processing because it means linear reverse dependence on number of processors.

In Appendix B we outlined some systems which we have analyzed in accordance of further development and implementing of NL-addressing. Two main groups of systems we have selected are:

- *DBMS based approaches (non-native RDF data storage): Oracle [Oracle, 2013], 3Store [AKT Project, 2013], Jena [Jena, 2013], RDF Suite [RDF Suite, 2013], Sesame [Sesame, 2012], 4store [4store, 2013];*
- *Multiple indexing frameworks (native RDF data storage): YARS [YARS, 2013], Kowari [Kowari, 2004], Virtuoso [Virtuoso, 2013], RDF-3X [Neumann & Weikum, 2008], Hexastore [Weiss et al, 2008], RDFCube [Matono et al, 2007], BitMat [Atre et al, 2009], Parliament [Kolas et al, 2009].*

Taking in account our experiments with relational data base we may conclude that for group of DBMS based approaches we will have similar proportions if we realize NL-addressing for more qualitative hardware platforms, for instance cluster machines.

Our approach is analogous to multiple indexing frameworks. The main difference is in reducing the information via NL-addressing and avoiding its duplicating in indexes. Again, if we realize NL-addressing for more qualitative hardware platforms, we will receive results which will outperform the analyzed systems.

What gain and loss using NL-Addressing for RDF storing?

The loss is additional memory for storing internal hash structures. But the same if no great losses we will have if we will build balanced search trees or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published.

The benefit is in two main achievements:

- *High speed for storing and accessing the information;*
- *The possibility to update and access the information immediately after storing without recompilation the database and rebuilding the indexes. This is very important because half or analyzed systems do not support updates (see Table 77).*

The main conclusion is optimistic. The future realization of NL-addressing for cluster machines and corresponded operation systems is well-founded.

8 Practical aspects

Abstract

Some practical aspects of implementation and using of NL-addressing will be discussed in this chapter.

NL-addressing is approach for building a kind of so called "post-relational databases". In accordance with this the transition to non-relational data models will be outlined.

The implementations have to be done following corresponded methodologies for building and using of ontologies. Such known methodology will be discussed in the chapter. It is called "METHONTOLOGY" and guides in how to carry out the whole ontology development through the specification, the conceptualization, the formalization, the implementation and the maintenance of the ontology.

Special case is creating of ontologies of text documents which are based on domain ontologies. It consists of Document annotation and Ontology population which we will illustrate following the OntoPop platform [Amardeilh, 2006].

The software realized in this research was practically tested as a part of an instrumental system for automated construction of ontologies "ICON" ("Instrumental Complex for Ontology designatioN") which is under development in the Institute of Cybernetics "V.M.Glushkov" of National Academy of Sciences of Ukraine.

In this chapter we briefly will present ICON and its structure. Attention will be paid to the storing of internal information resources of ICON realized on the base of NL-addressing and experimental programs WordArM and OntoArM.

8.1 The transition to non-relational data models

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. The graph oriented approach for storing ontologies became one of the preferred. Perhaps the most telling example is the system AllegroGraph® 4.9 [AlegroGraph, 2012] of the FRANZ Inc.

Franz Inc. is an innovative technology company with expert knowledge in developing and deploying Semantic Web technologies (i.e. Web 3.0) and providing Common Lisp based tools that offer an ideal environment to create complex, mission-critical applications [Franz Inc., 2013].

AllegroGraph and Allegro CL with AllegroCache are distinct platforms that provide scalable technology infrastructures which offer start-ups and fortune 100 companies the ability to realize new knowledge-rich applications for enhanced business intelligence.

AllegroGraph is a modern, high-performance, persistent graph database. AllegroGraph uses efficient memory utilization in combination with disk-based storage, enabling it to scale to billions of quads while maintaining superior performance. AllegroGraph supports SPARQL, RDFS++, and Prolog reasoning from numerous client applications [AlegroGraph, 2012].

Franz Inc. announced at the June 2011 Semtech conference a load and query of 310 Billion triples as part of a joint project with Intel. In August 2011, with the help of Stillwater SC (http://www.stillwater-sc.com/) and Intel, they achieved the industry's first load and query of *1 Trillion RDF Triples*. Total load was 1,009,690,381,946 triples in just over 338 hours for an average rate of 829,556 triples per second.

The driving force has been AIDA platform of Amdocs Product Enabler Group (Amdocs). The "Amdocs Intelligent Decision Automation" (AIDA) is an engine that is powered by Franz AllegroGraph 4.0 real-time semantic technology [Guinn & Aasman, 2010].

AllegroGraph provides dynamic reasoning and DOES NOT require materialization. AllegroGraph's RDFS++ engine dynamically maintains the ontological entailments required for reasoning; it has no explicit materialization phase. (*Materialization* is the pre-computation and storage of inferred triples so that future queries run more efficiently.)

The central problem with materialization is its maintenance: changes to the triple-store's ontology or facts usually change the set of inferred triples. In *static* materialization, any change in the store *requires complete re-processing before new queries can run*. AllegroGraph's dynamic materialization simplifies store maintenance and reduces the time required between data changes and querying. AllegroGraph also has RDFS++ reasoning with built in Prolog.

Let remember shortly some comments that concern one of the problems connected with RDF triple stores.

In April, 2011, Dr. Jans Aasman, CEO[†] of Franz Inc., the leading supplier of Graph Database technology for the Semantic Web wrote [Aasman, 2011]:

People ask me all the time, "*Will triple stores replace relational databases in three or five years?*" and I usually give two answers:

Answer 1: **Yes**, because triple stores provide 100 times more flexibility. For example, triple stores make it so much easier to add new predicates (think columns in relational databases) and write complicated ad hoc queries or perform inference and rule processing. Triple stores will soon be as robust, user-friendly and manageable as relational databases. Relational databases may continue to perform a bit better on simple joins, but triple stores already produce better performance when it comes to complicated queries, rule handling and inference. Given this robustness and usability – if the speed is roughly the same – many people will make the choice to switch to the more flexible solution.

[†] CEO: Chief Executive Officer - the corporate executive responsible for the operations of the firm; reports to a board of directors; may appoint other managers (including a president).

Answer 2: **No**, triple stores will continue to be used in conjunction with relational databases for the near future. Many installed legacy systems took millions of dollars to implement, and it's impractical to replace these systems in the near term. In these cases, triple stores can enable smart integration of databases by adding intelligent metadata on top of databases. Many companies are already using triple stores as a "smart brain" on top of their legacy systems [Aasman, 2011].

Post-relational data bases give new possibilities but are not aimed to replace RDBMS. Both have one main goal – ***to store data effectively***.

Because of this, it is not correct to claim one against another.

In addition, many new approaches are built over the RDBMS platforms. In the same time, it is important to point main features of RDF triple stores which make them preferable.

Steve Harris, the CTO* of a company that extensively uses RDF triple stores commercially, has outlined the "***five main features***" of RDF triple stores which make them preferable [TSRD, 2012]:

- ***Schema flexibility*** - it's possible to do the equivalent of a schema change to an RDF store live, and without any downtime, or redesign - it's not a free lunch, you need to be careful with how your software works, but it's a pretty easy thing to do;
- ***More modern*** - RDF stores are typically queried over HTTP it's very easy to fit them into Service Architectures without performance penalties. Also they handle internationalized content better than typical SQL databases - e.g. you can have multiple values in different languages;
- ***Standardization*** - the level of standardization of implementations using RDF and SPARQL is much higher than SQL. It's possible to swap out one triple store for another, though you have to be careful you're not stepping outside the standards. Moving data between stores is easy, as they all speak the same language;
- ***Expressivity*** - it's much easier to model complex data in RDF than in SQL, and the query language makes it easier to do things like LEFT JOINs (called OPTIONAL in SPARQL). Conversely though, if you data are very tabular, then SQL is much easier;
- ***Provenance*** - SPARQL lets you track where each piece of information came from, and you can store metadata about it, letting you easily do sophisticated queries, only taking into account data from certain sources, or with a certain trust level, on from some date range etc.

There are downsides though. SQL databases are generally much more mature, and have more features than typical RDF databases. Things like transactions are often much more crude, or nonexistent. Also, the cost per unit information stored in RDF vs. SQL is noticeably higher. It's hard to generalize, but it can be significant if you have a lot of data - though at least in our case it's an overall benefit financially given the flexibility and power [TSRD, 2012].

* CTO: *Chief Technology Officer* or *Chief Technical Officer* is an executive-level position in a company or other entity whose occupant is focused on scientific and technological issues within an organization.

8.2 Building and using of ontologies

The flexibility of triple stores is very important for solving of two considerable practical problems: building and using of domain ontologies and, directly connected to it, building and using of ontologies of text documents.

➢ *Domain ontologies*

Domain ontologies are formal descriptions of the classes of concepts and the relationships among those concepts that describe an application area. In other words, domain ontology models concepts and relationships that are relevant to the given domain (e.g., biology, architecture, software engineering) [Witte et al, 2010]. Building domain ontologies is not a simple task when domain experts have no background knowledge on engineering techniques and/or they have not much time to invest in domain conceptualization.

In order to develop domain ontology, some methodology has to be followed. For instance, such methodology is the "METHONTOLOGY Framework" developed within the Ontological Engineering group at Universidad Politécnica de Madrid [Fernández et al, 1997].

This methodology enables the construction of ontologies at the knowledge level, and has its roots in the main activities identified by the IEEE software development process and in other knowledge engineering methodologies. METHONTOLOGY guides in how to carry out the whole ontology development through the specification, the conceptualization, the formalization, the implementation and the maintenance of the ontology [Corcho et al, 2005].

The "METHONTOLOGY Framework" reduced the existing gap between ontological art and ontological engineering [Fernández et al, 1997] mainly by:

- *Identifying* a set of activities to be done during the ontology development process. They are: plainly, specify, acquire knowledge, conceptualize, formalize, integrate, implement, evaluate, document, and maintain;

- Proposing the *evolving prototype* as the life cycle that better fits with the ontology life cycle. The life of ontology moves on through the following states: specification, conceptualization, formalization, integration, implementation, and maintenance. The evolving prototype life cycle allows the ontologies to go back from any state to other if some definition is missed or wrong. So, this life cycle permits the inclusion, removal or modification of definitions *anytime* of the ontology life cycle. Knowledge acquisition, documentation and evaluation are support activities that are carried out during the majority of these states;

- METHONTOLOGY highly recommends the *reuse* of existing ontologies.

The METHONTOLOGY framework provides the idea of support activities: Knowledge Acquisition and Validation/Verification. It is divided into three main phases: *Specification*, *Conceptualization* and *Implementation*. These phases constitute an iterative process[Brusa et al, 2006].

✓ *Specification phase*

The specification activity states why the ontology is being built, what its intended uses are and who the end-users are.

The goal of the *specification phase* is to acquire informal knowledge about the domain, i.e. to produce either an informal, semi-formal or formal ontology specification document written in natural language, using a set of intermediate representations or using competency questions, respectively.

It is important to bear in mind that knowledge acquisition is an independent activity in the ontology development process. However, it is coincident with other activities. Most of the acquisition is done simultaneously with the requirements specification phase, and decreases as the ontology development process moves forward [Fernández et al, 1997].

✓ *Conceptualization phase*

In this activity, developer will structure the domain knowledge in a conceptual model that describes the problem and its solution in terms of the domain vocabulary identified in the ontology specification activity.

The goal of the *conceptualization phase* is to organize and structure this knowledge using external representations that are independent of the implementation languages and environments The conceptualization activity in METHONTOLOGY organizes and converts an informally perceived view of a domain into a semi-formal specification using a set of intermediate representations based on tabular and graph notations that can be understood by domain experts and ontology developers. The result of the conceptualization activity is the *ontology conceptual model*. The formalization activity transforms the conceptual model into a formal or semi-computable model [Corcho et al, 2005].

With the goal of speeding up the construction of ontology, one might consider reuse of definitions already built into other ontologies instead of starting from scratch [Fernández et al, 1997].

✓ *Implementation phase*

The goal of *implementation phase* is to evaluate, i.e. validate/verification, developed ontologies. The implementation activity builds computable models in an ontology language (Ontolingua, RDF Schema, OWL, etc.). The maintenance activity updates and corrects the ontology if needed [Corcho et al, 2005].

Ontologies' implementation requires the use of an environment that supports the meta-ontology and ontologies selected at the integration phase. The result of this phase is the ontology codified in a formal language such us: CLASSIC, BACK, LOOM, Ontolingua, Prolog, C++, etc.

Evaluation means to carry out a technical judgment of the ontologies, their software environment and documentation with respect to a frame of reference (the requirements' specification document) during each phase and between phases of their life cycle. *Evaluation* subsumes the terms Verification and Validation [Fernández et al, 1997]:

- *Verification* refers to the technical process that guarantees the correctness of ontology, its associated software environments, and documentation with respect to a frame of reference during each phase and between phases of their life cycle;
- *Validation* guarantees that the ontologies, the software environment and documentation correspond to the system that they are supposed to represent.

➢ *Ontologies of text documents*

Creating of ontologies of text documents is based on domain ontology and consists of *Document annotation* and *Ontology population* [Amardeilh, 2006]:

- *Document Annotation* consists in (semi-)automatically adding metadata to documents, i.e. providing descriptive information about the content of a document such as its title, its author but mainly the controlled vocabularies as the descriptors of a thesaurus or the instances of a knowledge base on which the document has to be indexed;
- *Ontology Population* aims at (semi-)automatically inserting new instances of concepts, properties and relations to the knowledge base as defined by the domain ontology.

Once Document Annotation and Ontology Population are performed, the final users of an application can exploit the resulting annotations and instances *to query, to share, to access, to publish documents, metadata and knowledge.*

Document Annotation and Ontology Population can be seen as similar tasks.

- Firstly, they both rely on the modeling of terminological and ontological resources (ontologies, thesaurus, taxonomies…) to normalize the semantic of the documentary annotations as well as the concepts of the domain;
- Secondly, as human language is a primary mode of knowledge transfer, they both make use of text-mining methods and tools such as Information Extraction to extract the descriptive structured information from documentary resources or categorization to classify a document into predefined categories or computed clusters;
- Thirdly, they both more and more rely on the Semantic Web standards and languages such as RDF for annotating and OWL for populating [Amardeilh, 2006].

The document annotation and ontology population we will illustrate following the OntoPop platform [Amardeilh, 2006] (Figure 80).

We have three phases (Figure 80):

(1) Extracting information from semi-structured texts - the text-mining solutions parse a textual resource, creating semantic tags to mark up the relevant content with regard to the domain of concern.

(2) Mapping between the results of the Information Extraction tool and the ontology model - the mediation layer maps the semantic tags produced by the text mining tools into formal representations, being the content annotations (RDF) or the ontology instances (OWL).

(3) Representing and managing the domain ontology, the thesaurus and the knowledge base - the semantic tags are used either to semantically annotate the content with

metadata or to acquire *knowledge*, i.e. to semi-automatically construct and maintain domain terminologies or to semi-automatically enrich knowledge bases with the named entities and semantic relations extracted.

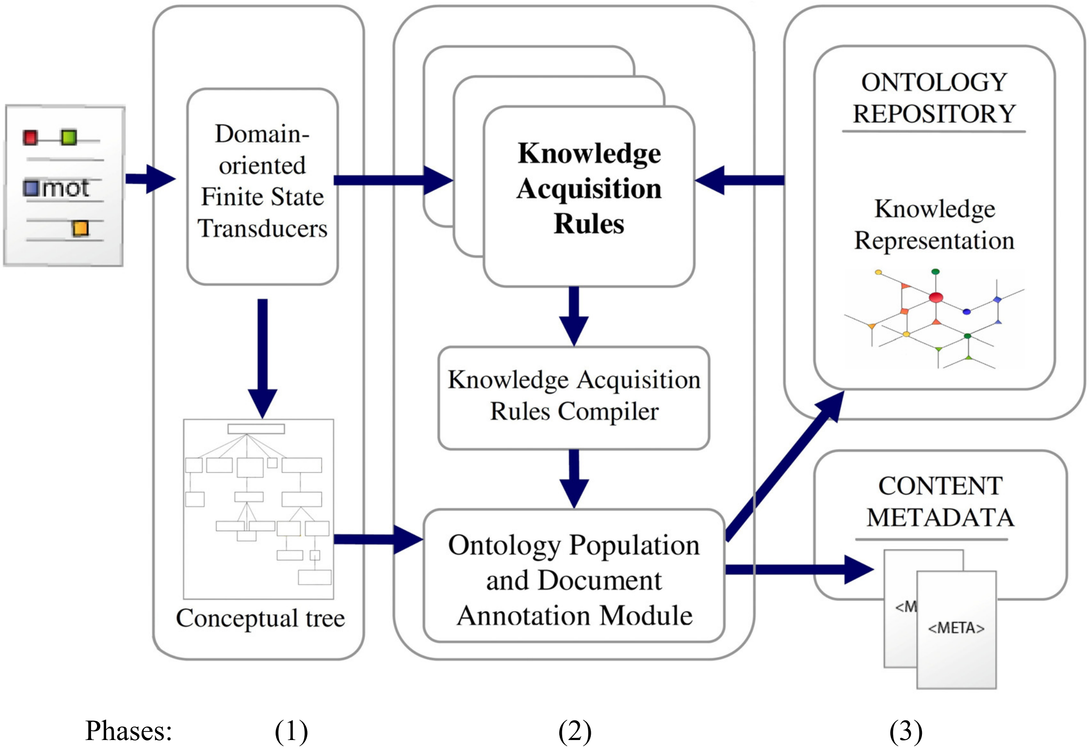

mot
Domain-oriented Finite State Transducers
Conceptual tree
Knowledge Acquisition Rules
Knowledge Acquisition Rules Compiler
Ontology Population and Document Annotation Module
ONTOLOGY REPOSITORY
Knowledge Representation
CONTENT METADATA
<M
<META>
Phases: (1) (2) (3)

Figure 80. The OntoPop's platform [Amardeilh, 2006]

➢ *Operations with ontologies stored by NL-addressing*

Operations for maintenance and integration of ontologies may be facilitated by using NL-addressing. It permits ontology operations to be realized by operations with corresponded layers of ontologies. It is possible to create a "virtual" ontology by combining only the paths to ontologies without any "real" creation a new one. In this case, the consistency has to be supported dynamically.

For instance, after merging ontologies irrespective of the kind of operation result (virtual or real), new ontology will contain a union of the layers of source ontologies.

When same relation (layer) exists in both ontologies, the process of merging may be provided in depth for all existing cells of layers. The problem to be solved is what to do if in different archives exist cells with equal location but different content.

Here we have three variants:

(1) To select cell content of the first ontology.

(2) To select cell content of the second ontology.

(3) To keep both contents and dynamically to make decision what is appropriate.

Our preference is to create virtual ontologies because this will save resources (time and space) and will give new possibilities based on dynamical selection of the content.

Using natural language addressing for storing dictionaries, thesauruses and ontologies facilitate its realization.

Let remember that not all of operations for maintenance and integration of ontologies can be made for all ontologies [Kalfoglou & Schorlemmer, 2003]. In general, these are very difficult tasks that are in general not solvable automatically [Obitko, 2007].

What is common and may be realized is developing of new generation tools for storing ontologies. At the first place, such tools are RDF-stores.

8.3 Building RDF-stores using NL-addressing

The Semantic Web and RDF triple stores are important research themes. Taking in account that NL-addressing is a possibility which may be used in addition to all already existing tools and approaches, below we will outline the main areas of its applicability. It is not correct to claim that NL-addressing will replace one or another tool. It has to be used where it is really effective.

In Chapter 1 we presented main approaches for creating RDF-triple stores. Below, following that explanation, we will sketch some practicable solutions [Ivanova et al, 2012b].

➢ *NL-Addressing for ontology generic schemas*

✓ *Vertical representation*

It is easy to realize vertical representation of a triple store via NL-addressing.

The values of *Subject* will be the addresses and all couples (*Predicate*, *Object*) for given value may be stored at one and the same address. This way with one operation all edges of a node of the graph will be received.

In the multi-layer variant, values of *Predicate* may be names of the layers (archives). In this case, additional operations for reading edges will be needed. The advantage is possibility to work only with selected layers and to reduce the time for access.

Nevertheless, in all cases the NL-addressing has constant complexity O(C), where C=max_L is the maximal length of the words or phrases, used for NL-addressing.

In the same time, the relational table has complexity at least $O(\log_d n)$, where “n” is number of all indexed elements (words) and “d” is the base of supporting (d-)balanced indexing and search.

The memory for balanced indexes exceeds the NL-addressing memory for indexes of hash tables.

The time for direct access is many times less than for access via search operations and updating the information. Let remember the speed experiments with *Firebird* relation data base, which had shown about 30-ty times for reading and more than 90-ty times for writing in NL-addressing’s favor.

✓ *Normalized triple store (vertical partitioning)*

The normalized triple store is ready for representing via NL-addressing.

We may use *multi-layer variant* where values of *Predicate* may be names of the layers (archives). In this case, additional operations for reading edges will be needed. The advantage is possibility to work only with selected layers and to reduce the time for access.

The *Subject* will be the NL-address and only *Object* will be saved. Possibility to concatenate all *Objects* for a given *Subject* reduces the size of memory and access time.

In addition, the vertical partitioning approach may be realized directly by the Multi-domain Information Model because it *directly supports the column-oriented DBMS (one column = one information space)*.

In all cases, the NL-addressing has constant complexity O(*m*C), where *m* is number of layers and C=max_L is the maximal length of the word or phrases, used for NL-addressing.

➢ *NL-Addressing for ontology specific schemas*

✓ *Horizontal representation*

The horizontal representation is an example of a set of layers. Storing every class in a separate layer (archive) gives possibility to add properties without restructuring existing tables.

Again, NL-addressing has constant complexity O(*m*C), where *m* is number of layers and C=max_L is the maximal length of the word or phrases, used for NL-addressing.

✓ *Decomposition storage model*

The decomposition storage model is memory and time consuming due to duplicating the information and generation of too much search indexes. In the same time, it is very near to the NL-addressing style and may be directly implemented using NL-addressing but *this will be not efficient*.

NL-addressing permits new possibilities due to omitting of explicit given information – names as well as balanced indexes. The feature tables may be replaced by NL-addressing access to corresponded points of the information space where all information about given *Subject* will exist. This way we will reduce the needed memory and time.

✓ *Multiple indexing frameworks*

The NL-addressing directly supports idea of multi-indexing because of the multi-layer structures and direct access to the *Object* values by NL-address computed on the base of the *Subject* and *Relation* values. Only the *Object*'s index has to be generated if it is really needed.

The above outlined ideas give basis for experiencing in a real software implementation of NL-addressing.

8.4 ICON - Instrumental Complex for Ontology designatioN

Design of ontologies, i.e. the formation sets of concepts, relations, axioms, and functions for interpretation, is a laborious process. Manual construction of these sets needs both time and many highly qualified specialists. This determines the development of tools (instrumental complexes) for automation of process of ontology design and distribution. The instrumental complexes for automated construction of ontologies are aimed to be used for the analysis and processing of large volumes of semi-structured data, such as linguistic corpuses in English, Dutch, Russian, Ukrainian, Bulgarian, and others languages.

Such instrumental complex is under development at the Kiev Institute of Cybernetics "V.M.Glushkov" of the National Academy of Sciences of Ukraine with the participation of Bulgarian experts from the Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences. This research is a part of this project and continues work for intelligent systems memory structuring [Gladun, 2003] done during the years [Mitov, 2011].

The complex is called "**ICON**" ("**I**nstrumental **C**omplex for **O**ntology designatio**N**", from Russian "ИКОН": "**И**нструментальный **К**омплекс **О**нтологического **Н**азначения") [Palagin et al, 2011].

Information model of ICON is presented in Figure 81 below.

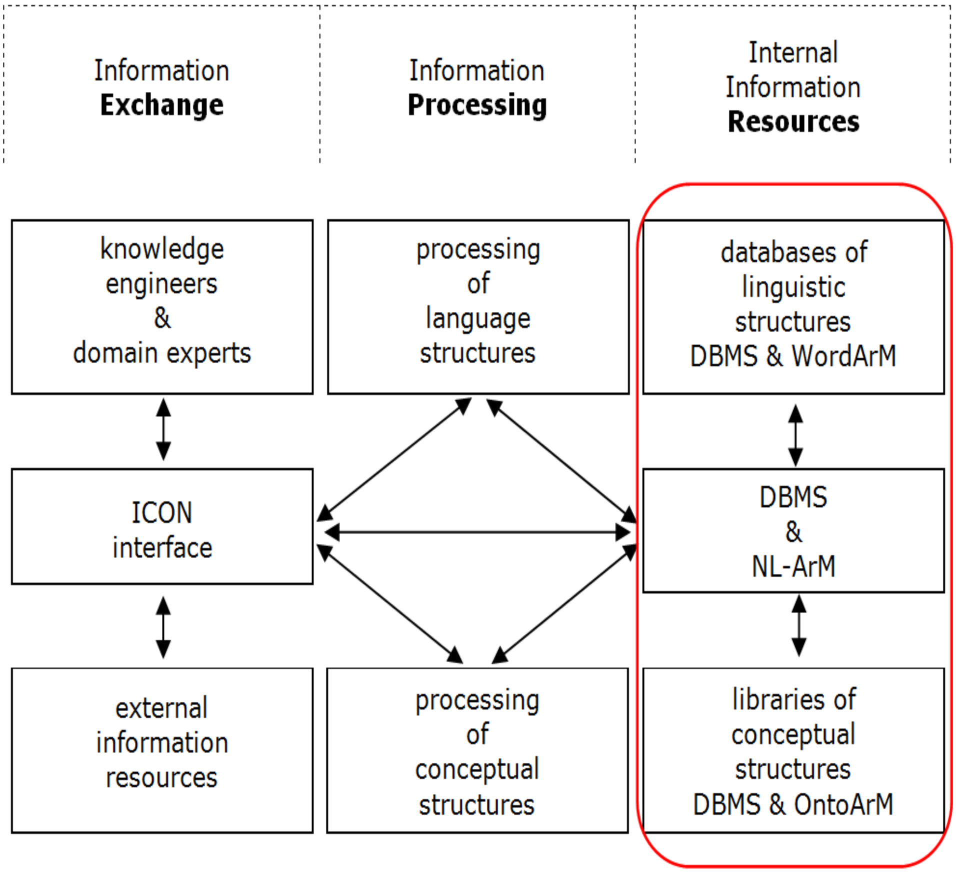

Information **Exchange**
Information **Processing**
Internal Information **Resources**
knowledge engineers & domain experts
processing of language structures
databases of linguistic structures DBMS & WordArM
ICON interface
DBMS & NL-ArM
external information resources
processing of conceptual structures
libraries of conceptual structures DBMS & OntoArM

Figure 81. Information model of ICON

ICON consists of three subsystems: *"Information exchange"*, *"Information processing",* and "*Internal information resources"*:

- "***Information Exchange"*** subsystem is aimed to serve manual or automatic c*ollecting and distributing of information* as well as interface with other subsystems of ICON to support creating, storing, visualization and export of the ontological knowledge. It serves retrieval of relevant to solving problem text documents which are available in the Internet and/or in other electronic collections. It include graphical user interface for knowledge engineers and domain experts, who provide preliminary design of ontologies, control and verification of design results, deciding on degree of completion design and more. Via this subsystem the external information resources can be accessed. They include different sources from local or global information bases and networks, such as:
  - Knowledge resources from given domain - electronic collections of encyclopedic dictionaries, monolingual dictionaries, thesauruses, etc.;
  - Internet resources - sources of text documents and distributed knowledge bases to be used in the process of creating ontologies.

  Collecting information from external sources is served by the ICON information-retrieval system. It is designed to detect and extract textual documents from various external sources and to create linguistic text corpora based on data from these documents;
- "***Information Processing"*** subsystem is a set of *original software modules* that implement relevant algorithms for the ontology' design, and *finished tools*, freely available on the Internet, such as Protégé [protégé, 2012] used as one of the main components in module for visual design. Processing of information includes: automatic natural language processing; knowledge discovery, extraction, representation, construction and verification of semantic structures; integration of ontological knowledge, etc. There are two main groups of processing tools respectively for *Linguistic structures* and *Conceptual structures;*
- "***Internal information resources"*** subsystem is aimed to support storing of large dictionaries, thesauruses, and ontologies in specialized electronic libraries based on NL-addressing tools realized in this research. It contains:
  - ✓ Linguistic libraries - a kind of electronic linguistic corpus which contains various dictionaries and thesauruses as well as document databases with source and/or processed information, for instance, a Linguistic corpuses of texts - a variety of text documents to be processed; and published documents with received results;
  - ✓ Conceptual libraries - they are built during the design or integration of ontologies. They are used to store both source information and finished ontological models.

➢ *Storing of the internal information resources of ICON*

Storing of the internal information resources of ICON is based on several relational DBMS as well as on program modules presented in this research – WordArM and OntoArM, outlined in

Appendix A. The main idea is to extend possibilities of “conventional” tools for semi-structured datasets. Conventional DBMS are used to store some structured information, like sets of descriptions of text documents to be processed.

Some finished tools for processing ontological information have their own databases but they are not appropriate for storing semi-structured information. For instance, such tool is the system Protégé [protégé, 2012]. It is written in Java and allows users to create their own database plug-ins. This choice is also consistent with rest of the Protégé plug-in architecture. Protégé developers chose the simplest schema that one could think of and focused on "maximal change" usage where the class structure and hierarchy is undergoing constant change. In this design, therefore, there is no attention paid to things such as query performance of any type.

Originally, Protégé has a single table that stores entire contents of the knowledge base which is developed as a frame based one [protégé, 2012].

The ***frame table*** has a fixed number of columns which are listed in Appendix B. It includes classes, slots, facets and instances. The Protégé meta-class architecture is used explicitly in the table to simplify things: all classes, slots, and facets are treated as frames. Each entry in the database corresponds to a frame in Protégé.

In the case of the superclass and subclass relations, Protégé stores duplicated information. For example with class A it stores that its subclass is B and with B it stores that its superclass is A. Maintaining separate records for these relations is necessary to maintain the ordering of both subclasses and superclasses. So while the "slot value" information is indeed duplicated in these records, the "index" information is unique (Subclass ordering is a user-interface feature that a number of users have requested. Protégé attaches no meaning to the ordering of superclasses or subclasses.) [protégé, 2012].

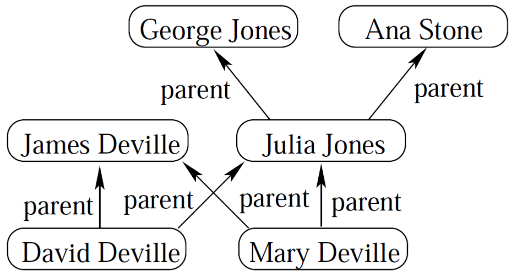
For large ontological structures the Protégé approach is not effective. As we have seen in Chapter 4 “Basic experiments”, the relational data bases are slower than post-relational ones based on NL-addressing, and take much resources for updating the information (especially for updating the indexes). Finally, Protégé does not support functions for dictionaries and thesauruses. The OWL and RDF descriptions are heavy to be parsed by human (see Appendix B).

The proper decision was to integrate Natural Language Addressing together with existing tools and this way to have available all needed functions.

The model which has been chosen is multi-layer storing of graph information. To remember it let's look at an example - the family tree of Figure 12 (its copy is given below).

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



The tree is represented by two tables: “NAME/LASTNAME” and “PERSON/PARENT”. For convenience, the children inherit the father's family.

The "multi-layer" representation of the family tree is given in Table 68.

Table 68. Multi-layer representation of the family tree

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

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 and avoid using of supporting indexes for information retrieval services (B-trees, hash tables, etc.).

➢ *Organization of ICON libraries*

The ICON internal information resources are stored in libraries which may be of two main types:

- Common libraries, which contain general information used practically by all users and models;
- Local libraries, which contain specific information needed only for given user or model.

In addition, these information resources may be linguistic or conceptual. This way we have a simple taxonomy (Figure 82):

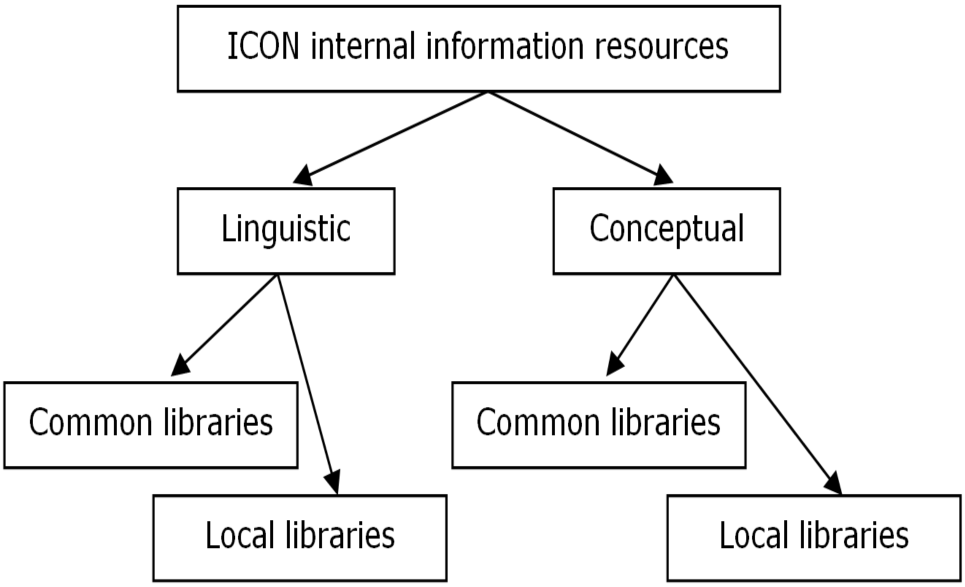


Figure 82. Taxonomy of ICON internal information resources

Libraries may be installed on single computer or distributed on local network.

Special description in a "context" table is used to establish correspondence between names, types, permissions, and allocations (paths) of library archives (files).

Common archives are allocated in shared folders. It is possible to have more than one folder with common archives. Updating of common archives may be done after permission from the administrator.

Local archives are stored in users' folders, which may be shared or not, depending of user preferences. Updating of local archives is under control of end-user.

Main difference between common and local archives is in the permissions for updating. Common archives have more strict discipline for making updates – it is obligation of and may be done only by administrators.

➢ *ICON Libraries of linguistic structures*

Libraries of linguistic structures are organized according different application areas (domains) covered by ICON. The tool for organization of these libraries is WordArM. As a rule there are no interconnections between linguistic archives (files) but there are many connections with conceptual structures where the linguistic information is used.

Common linguistic archives contain dictionaries and thesauruses of general purpose like Ukrainian-English dictionary or WordNet thesaurus of English.

Local linguistic archives contain thematic oriented dictionaries and thesauruses with specific information which concern given practical domain. For instance, it may be Medical thesaurus or Ukrainian-English dictionary of computer science.

One may note that the former ones have same general purposes as previous. This is quite right. What will be declared as common and what as local depends only on decision of administrators about the way of updating. Common archives may be changed only by administrator, but not by end-user.

We have to point to a special "*Data base of text documents"* which consists of original text documents and linguistic corpuses which are sources for creating the ontologies. In addition, we have to mention the common and local archives with metadata about documents and other information resources. The metadata is closely connected to documents and corresponded resources which are source for conceptual structures. All these information sources are organized using the ArMSpeed tool which is not mentioned in this research and because of this it is not discussed here.

➢ *ICON libraries of conceptual structures*

ICON conceptual libraries are built during the design or integration of ontologies. There are two kinds of such libraries:

- Library of domain ontologies;
- Library of ontologies of text documents.

These libraries are supported by OntoArM.

✓ *ICON library of domain ontologies*

Creating and editing domain ontologies in ICON is supported by its original ontological editor (see Appendix A7). It is able to read and store ontologies in OWL and XML formats. An example of the ontological graph generated by the ICON Ontological Editor is presented on Figure 109. This visualization of our sample graph (Figure 18) is created by this editor. In Table 72 the corresponded ICON XML description of sample graph is given. It is generated automatically.

The ICON Ontological Editor uses functions of OntoArM for saving ontologies. 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 of Appendix A8. It permits a preliminary evaluation of the number of layers needed for ICON at the project's first stage (about 50 up to 100).

The domain ontology consists of upper level ontology with a set of sub-ontologies subordinated to it. It is possible sub-ontologies to be stored in subfolders of those of the main ontology but this is not obligatory. Using links (local or global paths) ontology may subordinate several others. This way practically we have ontology network with unlimited size.

Ontology is stored in a separate folder. It contains all archives of all its layers. Link to ontology is the path to folder which contains it.

In addition, ontology may be connected to some linguistic resources – dictionaries and/or thesauruses. Again the connections are links but this time they point the file of the resource, i.e. the path to it.

➢ *ICON library of ontologies of text documents*

A generalized view of OntoArM implementation is shown on Figure 83 (following [Witte et al, 2010]).

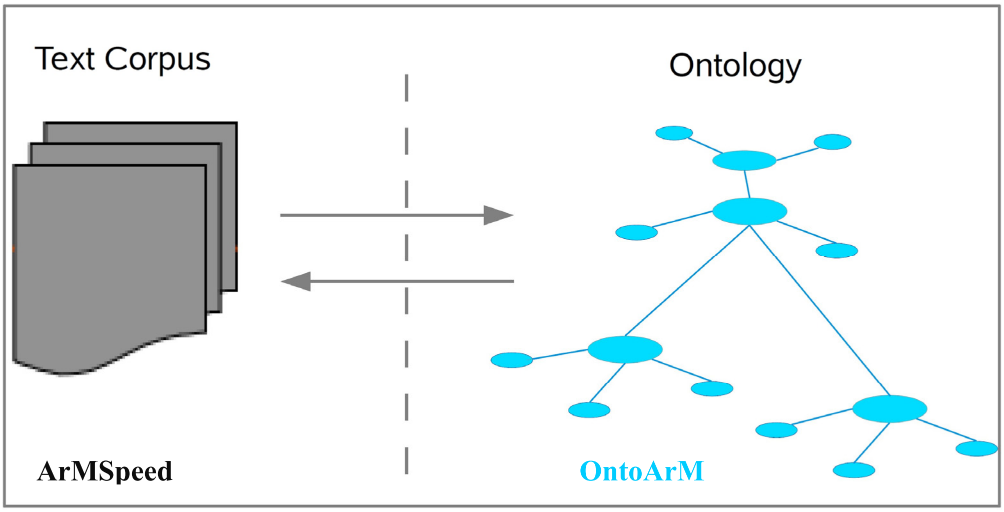


Figure 83. Using OntoArM for storing ontologies of text documents (following [Witte et al, 2010])

Text corpus and its metadata is stored using ArMSpeed module. Beside NL-addressing, in this module is used searching based on balanced trees.

Ontologies are stored by OntoArM.

Creating and editing ontologies of text documents in ICON is supported by its Ontological Editor based on:

- ArMSpeed for storing documents;
- OntoArM for storing ontologies of text documents, using the same storing model as for domain ontologies. It is multi-layer storing of ontology graph based on natural language addressing. In addition to sample list of layers used for storing common and local ontologies of text documents presented in Table 73, some specific for concrete text documents layers are raised up.

Ontology of a text document is stored in a separate folder. It contains all archives of all its layers. Link to ontology is the path to folder which contains it.

In addition, ontology of text document may be connected to some linguistic resources – dictionaries and/or thesauruses. The connections are links (paths) to the files of linguistic resources.

➢ *ICON methodology for construction of ontologies*

ICON follows similar methodology as the "METHONTOLOGY Framework" [Fernández et al, 1997].

It is important to point that ICON methodology permits inclusion, removal or modification of definitions *anytime* of the ontology life cycle. This is very important facility which causes serious problems to conventional databases which have to update permanently their indexing structures and this way to consume large (time and space) resources.

In addition, the processes of document annotation and ontology population ICON are similar to ones of OntoPop platform [Amardeilh, 2006] (Figure 80). NL-addressing is used for knowledge representation in the ontology repository.

NL-addressing facilitates the whole ontology development in ICON through the specification, the conceptualization, the formalization, the implementation and the maintenance of very large ontologies.

➢ *Conclusion of chapter 8*

Some practical aspects of implementation and using of NL-addressing were discussed in this chapter.

NL-addressing is approach for building a kind of so called "post-relational databases". In accordance with this the transition to non-relational data models was outlined.

The implementations have to be done following corresponded methodologies for building and using of ontologies. Such known methodology was discussed in the chapter. It is called "METHONTOLOGY" and guides in how to carry out the whole ontology development through the specification, the conceptualization, the formalization, the implementation and the maintenance of the ontology.

Special case is creating of ontologies of text documents which are based on domain ontologies. It consists of Document annotation and Ontology population which we illustrated following the known OntoPop platform [Amardeilh, 2006].

The software realized in this research was practically tested as a part of an instrumental system for automated construction of ontologies "ICON" ("Instrumental Complex for Ontology designatioN") which is under development in the Institute of Cybernetics "V.M.Glushkov" of NAS of Ukraine.

In this chapter we briefly presented ICON and its structure. Attention was paid to the storing of internal information resources of ICON realized on the base of NL-addressing and experimental programs WordArM and OntoArM.

ICON is still under developing and, during solving concrete problems, new functions based on NL-addressing and NL-ArM rise to be realized. For instance, such problems concern the operations with ontologies; work with very large ontological structures; etc.

Conclusion

The main goal of this research was to study a new approach for storing semi-structured datasets. To achieve this goal, we have studied and analyzed the existing methods and systems for storing semi-structured datasets and we have proposed an information model for storing semi-structured datasets and corresponded access method as well as tools for working in such style, theirs main principles, and storing functions.

We have provided experiments and practical approbation of the proposed model and tools by experimental software realizations and comparative evaluating in order to study their behavior under practical conditions and to compare with other tools from the same class. The main conclusion is optimistic. The future realization of NL-addressing, for instance – for cluster machines and corresponded operation systems, is well-founded.

Our further research will be directed to several interesting areas of implementing the NL-addressing in business applications where flexibility of this approach will give some new possibilities. Implementing the NL-addressing in linguistic systems which work with large linguistic data sets is another direction for further work.

Let point the area of cognitive modeling, too. It is clear; the human brain does not create indexes. The information processing in the brain looks like our model for NL-addressing. It is very interesting to provide research in this area.

Big Data

Maybe the most interesting is the area of so called “Big Data”. The term Big Data applies to information that can’t be processed or analyzed using traditional processes or tools. Increasingly, organizations today are facing more and more “Big Data challenges”. They have access to a wealth of information, but they don’t know how to get value out of it because it is sitting in its most raw form or in a semi-structured or unstructured format [Zikopoulos et al, 2012].

Popular approach for representing Big Data is Resource Definition Framework (RDF). Let remember, RDF is a graph based data format which is schema-less, thus unstructured, and self-describing, meaning that graph labels within the graph describe the data itself. The prevalence of RDF data is due to variety of underlying graph based models, i.e. almost any type of data can be expressed in this format including relational and XML data [Faye et al, 2012].

Big Data created the need for a new class of capabilities to augment the way things are done today to provide better line of site and controls over our existing knowledge domains and the ability to act on them.

BigArM

In the Big Data community, the “MapReduce Paradigm” has been seen as one of the key enabling approaches for meeting the continuously increasing demands on computing resources imposed by massive data sets. MapReduce is a highly scalable programming paradigm capable of processing massive volumes of data by means of parallel execution on a large number of commodity computing nodes. It was recently popularized by Google [Dean & Ghemawat, 2008], but today the MapReduce paradigm has been implemented in many open source projects, the most prominent being the Apache Hadoop [Hadoop, 2014]. The popularity of MapReduce can be accredited to its high scalability, fault-tolerance, simplicity and independence from the programming language or the data storage system.

At the same time, MapReduce faces a number of obstacles when dealing with Big Data including the lack of a high-level language such as SQL, challenges in implementing iterative algorithms, support for iterative ad-hoc data exploration, and stream processing [Grolinger et al, 2014].

A possible solution may be the approach of Natural Language Addressing (NLA) presented in this monograph. It is suitable for storing Big Data. Its main idea is to use internal encoding of letters of a word or phrase as elements of co-ordinate vector which may be used as hyper-space address of the information connected to this word or phrase. As result the standard indexing and recompilation of information base are avoided.

Three main characteristics define Big Data: ***Volume, Variety, and Velocity*** [Zikopoulos et al, 2012]. These characteristics cause corresponded problems of storing Big Data which may be solved by means of NLA [Markov et al, 2014]:

- **Volume** (the sheer volume of data being stored today is exploding) – avoiding additional indexing, duplication of keywords, and corresponded pointers, leads to reducing additional memory needed for accessing information i.e. we may use addressing but not classical search engines;
- **Velocity** (a conventional understanding of velocity typically considers how quickly the data is arriving and stored, and its associated rates of retrieval) – avoiding recompilation of information base permits high speed of storing and immediately readiness of information to be accessed. This is very important possibility for stream data;
- **Variety** (it represents all types of data — a fundamental shift in analysis requirements from traditional structured data to include raw, semi-structured, and unstructured data as part of the decision-making and insight process) – natural language addressing permits creating a special kind of graph information bases which may operate both with structured as well as semi-structured information.

What is needed is to extend possibilities of ArM32 up to 64 bit addressing capabilities and to rationalize the internal hash structures to speed access from milliseconds down to microseconds per one access operation. This will be done in ongoing developing of its new version called “BigArM” for 64 bit machines and operating systems like MS Windows and Linux as well as for Cloud processing.

Collect/Report Paradigm

Realizing BigArM will permit new kind of Cloud processing of Big Data, called "Collect/Report Paradigm" (CRP). Its idea is very simple and because of this it is perspective to be realized.

CRP is based on the possibility of NLA to separate incoming information coded as RDF-triples on many different layers stored in separate archives which may be distributed all over the world. The correspondence between archives is strongly kept by names as addresses which are equal for all layers.

Similar model we may see in the game of chance "Bingo" (Figure 84) for two or more players, who mark off numbers on a grid with unique sequence of numbers printed on their individual cards as they are announced by the Caller corresponding to numbered balls drawn at random; the game is won by the first person to call out "bingo!" or "house!" after crossing off all numbers on the grid or in one line of the grid [YourDictionary, 2013].

Figure 84. Illustration of Collect/Report Paradigm via example of Bingo game

To play Bingo one has to "***collect***" (to buy) one or more individual cards and after starting the game to listen what number the Caller will announce, to find in the individual cards the same numbers and to mark them (i.e. to process the stream of incoming data). After marking every new number, (in real time, before next number will be announced) player has to analyze the configuration of marked cells on the individual cards and to decide if it is the winner configuration. If the configuration is a winner one, the player has to "***report***" (to call out) "Bingo". Only the players with winner configurations have to report, the others must stay silent.

In Collect/Report Paradigm, all nodes have to "listen" in parallel the incoming stream of RDF-data and to "collect" (to store) information only in the layers the nodes have to support. In the same time, nodes have to "listen" incoming stream of requests and only nodes, which have information corresponded to given request has to "report" (to send answer).

As an example let's remember a part from Table 32 (Table 69). Let it represents six nodes numbered from 1 to 6 which may be distributed over the net. Incoming information is in RDF triples (subject, relation, object). Information (objects) for the same subject and relation is concatenated in the corresponded points. Let assume that Table 69 represents the state of nodes at given time moment. If in this moment a request for word "cut" will come, only nodes 1 and 6 will "report" the content (definitions) from corresponded cells. Node 1 will report only the first row which correspond to "cut" with small letters but not its second row which corresponds to word "CUT" with capital letters. Nodes 2, 3, 4, and 5 will rest silent.

Table 69. A part from Table 32

node	layer	NLA	definition
1	**adj_all**	cut	{ cut, shortened, (with parts removed; "the drastically cut film") } { cut, thinned, weakened, (mixed with water; "sold cut whiskey"; "a cup of thinned soup") } { cut, slashed, ((used of rates or prices) reduced usually sharply; "the slashed prices attracted buyers") } { cut, emasculated, gelded, ((of a male animal) having the testicles removed; "a cut horse") }
		CUT	{ [CUT1, UNCUT1,!] (separated into parts or laid open or penetrated with a sharp edge or instrument; "the cut surface was mottled"; "cut tobacco"; "blood from his cut forehead"; "bandages on her cut wrists") } { [CUT2, UNCUT2,!] ((of pages of a book) having the folds of the leaves trimmed or slit; "the cut pages of the book") }

node	layer	NLA	definition
			{ [CUT3, UNCUT3,!] (fashioned or shaped by cutting; "a well-cut suit"; "cut diamonds"; "cut velvet") }
2	**adj_pert**	cut	empty definition
3	**adj_ppl**	cut	empty definition
4	**adv_all**	cut	empty definition
5	**noun_Tops**	cut	empty definition
6	**noun_act**	cut	{ cut6, absence,@ (an unexcused absence from class; "he was punished for taking too many cuts in his math class") } { cut5, reduction,@ (the act of reducing the amount or number; "the mayor proposed extensive cuts in the city budget") } { cut, [cutting, verb.creation:cut11,+] cutting_off1, shortening,@ (the act of shortening something by chopping off the ends; "the barber gave him a good cut") } { cut1, [cutting1, verb.contact:cut10,+ verb.contact:cut,+] division,@ (the act of cutting something into parts; "his cuts were skillful"; "his cutting of the cake made a terrible mess") } { cut2, [cutting2, verb.contact:cut10,+] opening2,@ (the act of penetrating or opening open with a sharp edge; "his cut in the lining revealed the hidden jewels") } { cut9, [cutting9, verb.contact:cut5,+] division,@ card_game,#p (the division of a deck of cards before dealing; "he insisted that we give him the last cut before every deal"; "the cutting of the cards soon became a ritual") } { cut8, [undercut, verb.contact:undercut,+] stroke,@ tennis,;c badminton,;c squash,;c ((sports) a stroke that puts reverse spin on the ball; "cuts do not bother a good tennis player") }

In general, Collect/Report Paradigm is illustrated on Figure 85.

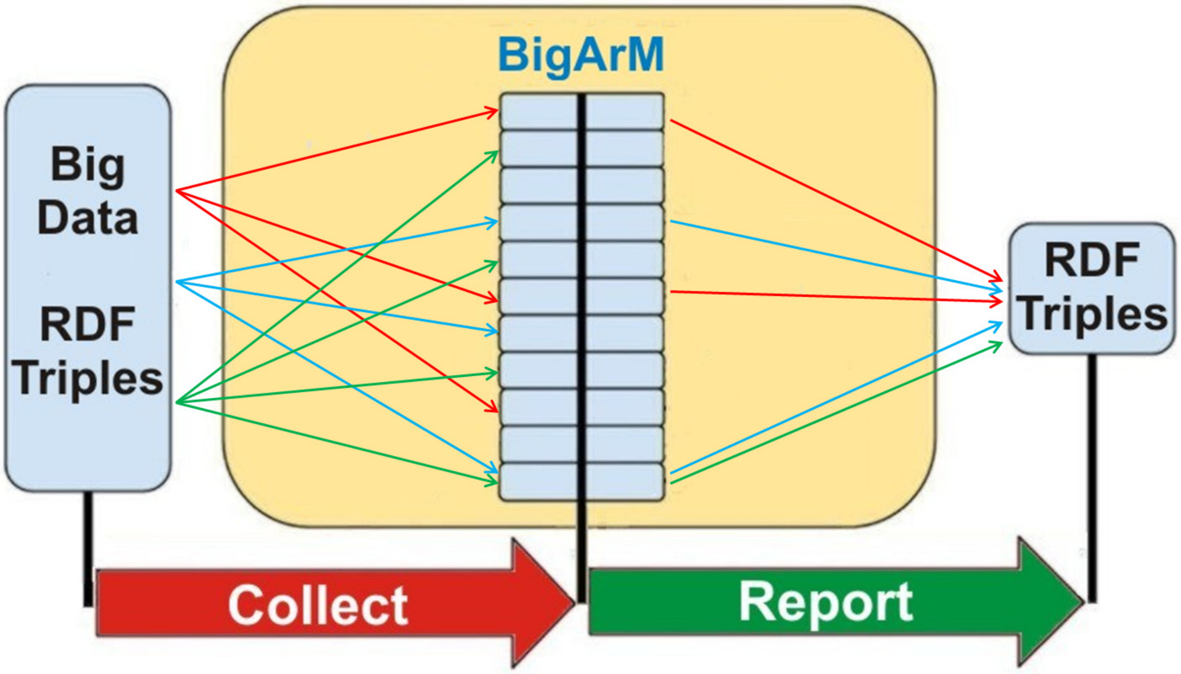


Figure 85. Cloud Collect/Report Scheme for Storing and Accessing Big Data

Main advantages of Collect/Report Paradigm (Figure 85) are:

- Collecting information is done by all nodes independently in parallel. It is possible one node to send information to another;
- Reporting information is provided only by the nodes which really contain information related to the request; the rest nodes do not react, they remain silent;
- Input data as well as results are in RDF-triple or RDF-quadruple format.

Main results presented in the monograph

***Chapter 1** was aimed to introduce the theoretical surroundings of our work.*

Firstly in this chapter, we remembered the needed basic mathematical concepts. Special attention was paid to the Names Sets – mathematical structure which we implemented in our research. We used strong hierarchies of named sets to create a specialized mathematical model for new kind of organization of information bases called "Multi-Domain Information Model" (MDIM). The "information spaces" defined in the model are kind of strong hierarchies of enumerations (named sets).

*We will realize MDIM via special kind of hashing. Because of this, we remembered the main features of hashing and types of hash tables as well as the idea of "Dynamic perfect hashing" and "Trie", especially – the "Burst trie". A **burst trie** is an in-memory data structure, designed for sets of records that each has a unique string that identifies the record and acts as a key. Burst trie consists of three distinct components: a set of records, a set of containers, and an access trie.*

***Chapter 2** introduced the main data structures and storing technologies which further we will use to compare our results. Mainly they are graph data models as well as RDF storage and retrieval technologies.*

Firstly we defined concepts of storage model and data model.

Mapping of the data models to storage models is based on program tools called "access methods". Their main characteristics were outlined.

Graph models and databases were discussed more deeply and examples of different graph database models were presented. The need to manage information with graph-like nature especially in RDF-databases had reestablished the relevance of this area.

There is a real need of efficient tools for storing and querying knowledge using the ontologies and the related resources. In this context, the annotation of unstructured data has become a necessity in order to increase the efficiency of query processing. Efficient data storage and query processing that can scale to large amounts of possibly schema-less data has become an important research topic. The proposed approaches usually rely on (object-) relational database technology or on main-memory virtual machine implementations, while employing a variety of storage schemes [Faye et al, 2012].

In accordance with this, the analyses of RDF databases as well as of the storage and retrieval technologies for RDF structures were in the center of our attention. The analysis of the viewed tools showed that all of them use data storing models which are limited to text files, indexed data or relational databases. These approaches do not conform to the specific structures of the ontologies. This necessitates the development of new models and tools for storing ontologies which correspond to their structure.

Storing models for several popular ontologies and summary of main types of storing models for ontologies and, in particular, RDF data were discussed.

Our attention was paid to addressing and naming (labeling) in graphs with regards to introducing the Natural Language Addressing (NL-addressing) in graphs. A sample graph was analyzed to find its proper representation.

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.

Concluding, let us point on advantages and disadvantages of the multi-layer representation of graphs.

The main disadvantages are:

- *The layers are sparsed;*
- *The number of locations may be very great which causes the need of corresponded number of columns in the table (in any cases hundred or thousand).*

The main advantages are:

- *Reducing the used resources;*
- *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 and avoid using of supporting indexes for information retrieval services (B-trees, hash tables, etc.);*
- *Finally, using NL-addressing, the multi-layer representation is easily understandable by humans and interpretable by the computers.*

If we will use indexed files or relational data bases, the disadvantages are so serious that make the implementation impossible.

We proposed to use the multi-dimensional model for organization of information. For this purpose the "Multi-Domain Infrmation Model"and its realizations were presented. The Multi-Dimensional Numbered Information Spaces are basis for context independed indexing. Because of this they may be used for storing Big Data.

***Chapter 3** was aimed to introduce a new access method based on the idea of Natural Language Addressing.*

*MDIM and its realizations are not ready to support NL-addressing. We upgraded them for ensuring the features of NL-addressing via new access method called **NL-ArM**.*

The program realization of NL-ArM is based on specialized hash functions and two main functions for supporting the NL-addressing access.

In addition, several operations were realized to serve the work with thesauruses and ontologies as well as work with graphs.

*NL-ArM is ready for storing RDF information. It is possible to define tree information models for storing RDF-graphs using NL-ArM: (1) **RSO model** (Relation-Subject-Object model), (2) **SRO model** (Subject-Relation-Object model), and (3) **UNL model ((Subject, Relation) => Object** Universal model) .*

*In **Chapter 4** two main types of basic experiments were presented. NL-ArM has been compared with (1) sequential text file of records and (2) relational database management system Firebird.*

The need to compare NL-ArM access method with text files was determined by practical considerations – in many applications the text files are main approach for storing semi-structured data. To investigate the size of files and speed of their generation we compared writing in a sequential text file and in a NL-ArM archive.

For 8 characters as length of the keywords and small quantity of records, the NL-ArM archive occupies more memory than text file but for the case of very large data the NL-ArM archive is smaller. It is important to underline that these experiments were based on artificial data with fixed length (record of 30 bytes with 8 bytes artificially generated keyword of arbitrary ASCII symbols). If the length of the keywords is variable, the size of NL-ArM archive will be different according of length of the strings of keywords of stored information, i.e. according of number of layers of hash tables (depth of trie).

In sequential storing of records, NL-ArM access method is slower than same operation in text file. For applications where it is important in real time to register incoming information, the text files are preferable than archives with NL-Addressing.

To provide experiments with a relational database, we have chosen the system "Firebird". It should be noted that Firebird and NL-ArM have fundamentally different physical organization of data and the tests cover small field of features of both systems.

We did not compare the sizes of files of NL-ArM and Firebird because of difference of keywords – symbols for Firebird and integer values for NL-ArM.

*In writing experiments, regarding NL-ArM, Firebird is on average **90.1 times slower.** This result is due to two reasons. The first is that balanced indexes of Firebird need reconstruction for including of every new keyword. This is time consuming process. The second reason is the speed of updating NL-ArM hash tables which do not need recompilation after including new information. Due to specific of realization, for small values of co-ordinates NL-ArM is not as effective as for the great ones.*

*In reading experiments, regarding NL-ArM, Firebird is on average **29.8 times slower.** This result is due to the speed of access in NL-ArM hash tables which do not need search operations.*

If we need direct access to large dynamic data sets (via NL-path), than more convenient are hash based tools like NL-ArM. For instance, such cases are large ontologies and RDF-graphs.

*In **Chapter 5** we have presented several experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of structured datasets.*

Firstly we introduced the idea of knowledge representation. Further in the chapter we discussed three main experiments - for NL-storing of dictionaries, thesauruses, and ontologies.

Presentations of every experiment started with introductory part aimed to give working definition and to outline state of the art in storing concrete structures.

The explanation of the experiments begins with the easiest case – storing dictionaries. Analyzing results from the experiment with a real dictionary data we may conclude that it is possible to use NL-addressing for storing such information.

Next experiment was aimed to answer to question: "What we gain and loss using NL-Addressing for storing thesauruses?"

Analyzing results from the experiment we point that the loss is additional memory for storing hash structures which serve NL-addressing. But the same if no great losses we will have if we will build balanced search trees or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published. The benefit is in two main achievements:

(1) High speed for storing and accessing the information.

(2) The possibility to access the information immediately after storing without recompilation the database and rebuilding the indexes.

The third experiment considered the complex graph structures such as ontologies. The presented survey of the state of the art in this area has shown that main models for storing ontologies are files and relational databases.

Our experiment confirmed the conclusion about losses and benefits from using NL-addressing given above for thesauruses. The same is valid for more complex structures.

Here we have to note that for static structured datasets it is more convenient to use standard utilities and complicated indexes. NL-addressing is suitable for dynamic processes of creating and further development of structured datasets due to avoiding recompilation of the database index structures and high speed access to every data element.

The goal of the experiments with different small datasets, like dictionary, thesaurus or ontology, was to discover regularities in the NL-addressing realization. Analyzing Table 25, Table 27, and Table 33 we may see the main two regularities of storing time using NL-addressing:

— *It depends on number of elements in the instances;*

— *It not depends on number of instances in datasets.*

In ***Chapter 6*** *we have presented results from series of experiments which were needed to estimate the storing time of NL-addressing for middle-size and very large RDF-datasets.*

We described the experimental storing models and special algorithm for NL-storing RDF instances. Estimation of experimental systems was provided to make different configurations comparable. Special proportionality constants for hardware and software were proposed. Using proportionality constants, experiments with middle-size and large datasets become comparable.

Experiments were provided with both real and artificial datasets. Experimental results were systematized in corresponded tables. For easy reading visualization by histograms was given.

The goal experiments for NL-storing of middle-size and large RDF-datasets were to estimate possible further development of NL-ArM. We assumed that its "software growth" will be done in the same grade as one of the known systems like Virtuoso, Jena, and Sesame. In the next chapter we will *analyze what will be the place of NL-ArM in this environment but already we may see that NL-addressing have good performance and NL-ArM has similar results as Jena and Sesame.*

In ***Chapter*** *7 we have analyzed experiments presented in previous Chapters 4, 5, and 6, which contain respectively results from (1) basic experiments; (2) experiments with structured datasets; (3) experiments with semi-structured datasets. Special attention was paid to analyzing of storing times of NL-ArM access method and its possibilities for multi-processing.*

From experimental data and visualizations we concluded that the NL-access time:

— *Depends on number of elements in a dataset's instances, which have to be stored on the disk;*

— *Not depends on number of instances in the dataset.*

The second is very important for multi-processing because it means linear reverse dependence on number of processors.

In Appendix B we outlined some systems which we have analyzed in accordance of further development and implementing of NL-addressing. Two main groups of systems we have selected are:

- *DBMS based approaches (non-native RDF data storage): Oracle [Oracle, 2013], 3Store [AKT Project, 2013], Jena [Jena, 2013], RDF Suite [RDF Suite, 2013], Sesame [Sesame, 2012], 4store [4store, 2013];*
- *Multiple indexing frameworks (native RDF data storage): YARS [YARS, 2013], Kowari [Kowari, 2004], Virtuoso [Virtuoso, 2013], RDF-3X [Neumann & Weikum, 2008], Hexastore [Weiss et al, 2008], RDFCube [Matono et al, 2007], BitMat [Atre et al, 2009], Parliament [Kolas et al, 2009].*

Taking in account our experiments with relational data base we may conclude that for group of DBMS based approaches we will have similar proportions if we realize NL-addressing for more qualitative hardware platforms, for instance cluster machines.

Our approach is analogous to multiple indexing frameworks. The main difference is in reducing the information via NL-addressing and avoiding its duplicating in indexes. Again, if we realize NL-addressing for more qualitative hardware platforms, we will receive results which will outperform the analyzed systems.

What gain and loss using NL-Addressing for RDF storing?

The loss is additional memory for storing internal hash structures. But the same if no great losses we will have if we will build balanced search trees or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published.

The benefit is in two main achievements:

- *High speed for storing and accessing the information;*
- *The possibility to update and access the information immediately after storing without recompilation the database and rebuilding the indexes. This is very important because half or analyzed systems do not support updates (see Table 77).*

The main conclusion is optimistic. The future realization of NL-addressing for cluster machines and corresponded operation systems is well-founded.

*In **Chapter 8** some practical aspects of implementation and using of NL-addressing were discussed in this chapter.*

NL-addressing is approach for building a kind of so called "post-relational databases". In accordance with this the transition to non-relational data models was outlined.

The implementations have to be done following corresponded methodologies for building and using of ontologies. Such known methodology was discussed in the chapter. It is called "METHONTOLOGY" and guides in how to carry out the whole ontology development through the specification, the conceptualization, the formalization, the implementation and the maintenance of the ontology.

Special case is creating of ontologies of text documents which are based on domain ontologies. It consists of Document annotation and Ontology population which we illustrated following the known OntoPop platform [Amardeilh, 2006].

The software realized in this research was practically tested as a part of an instrumental system for automated construction of ontologies "ICON" ("Instrumental Complex for Ontology designatioN") which is under development in the Institute of Cybernetics "V.M.Glushkov" of NAS of Ukraine.

In this chapter we briefly presented ICON and its structure. Attention was paid to the storing of internal information resources of ICON realized on the base of NL-addressing and experimental programs WordArM and OntoArM.

ICON is still under developing and, during solving concrete problems, new functions based on NL-addressing and NL-ArM rise to be realized. For instance, such problems concern the operations with ontologies; work with very large ontological structures; etc.

*Finally, in the **Conclusion,** we presented shortly the next steps. Special attention was done on the area of so called "Big Data" and possible implementation of NLA for processing of large semi-structured data sets. New realization of the access method called BigArM and connected to it Collect/Report Paradigm were outlined. Main advantages of Collect/Report Paradigm are (1) Collecting information is done by all nodes independently in parallel. It is possible one node to send information to another; (2) Reporting information is provided only by the nodes which really contain information related to the request; the rest nodes do not react, they remain silent; (3) Results are in RDF-triple or RDF-quadruple format.*

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

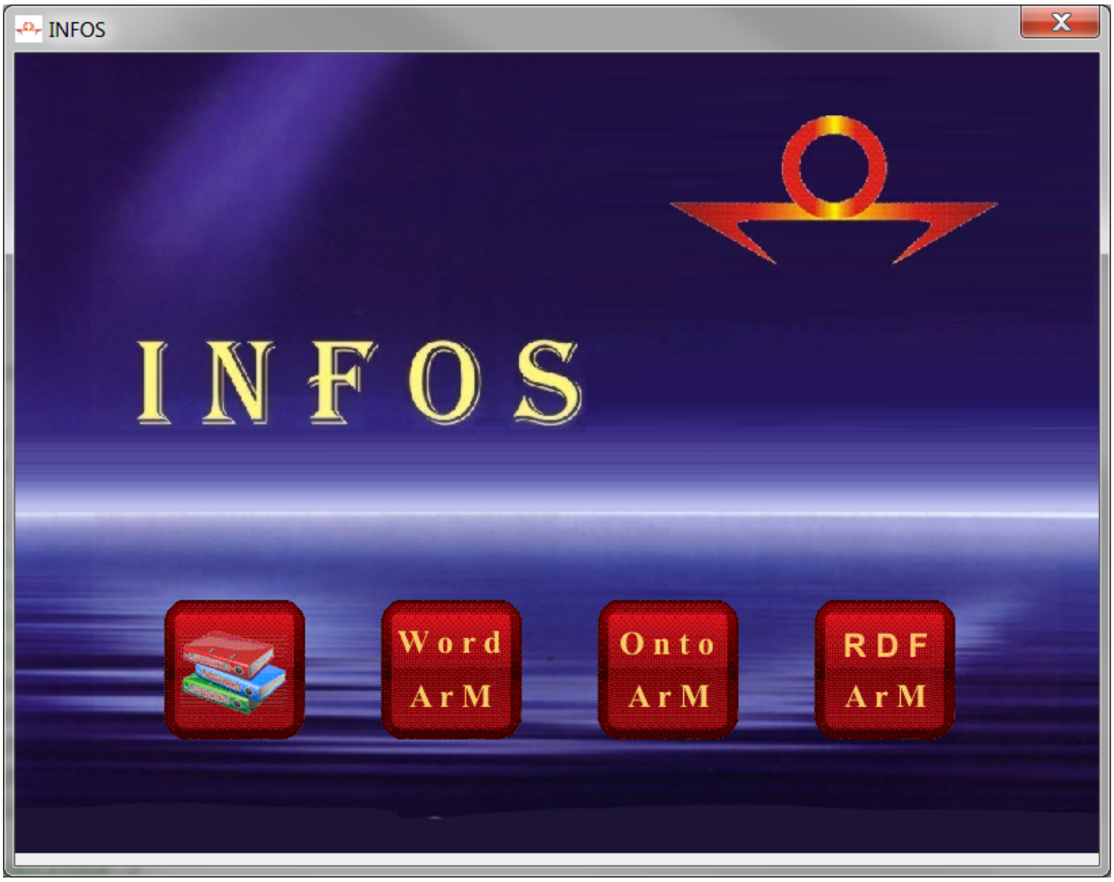


Figure 86. The front panel of system INFOS

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.

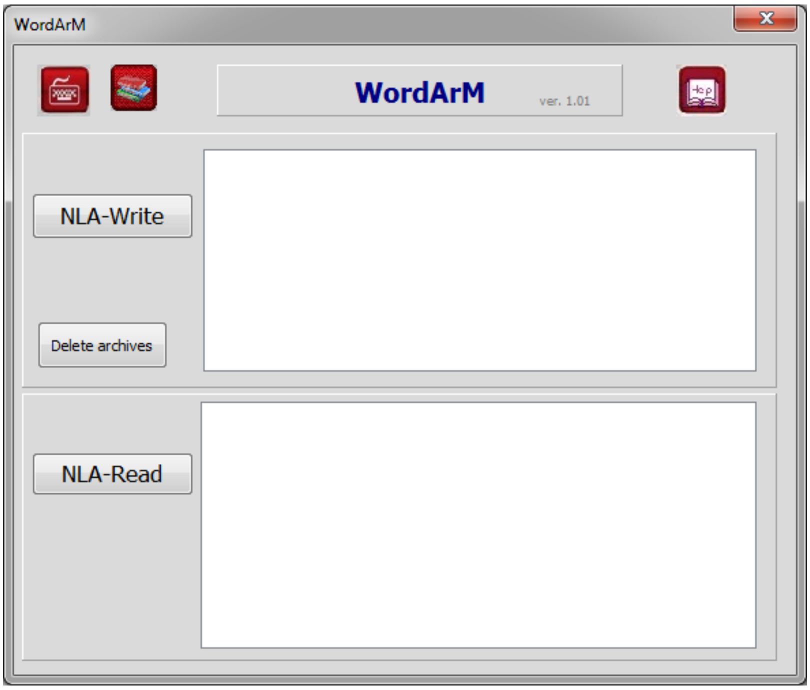
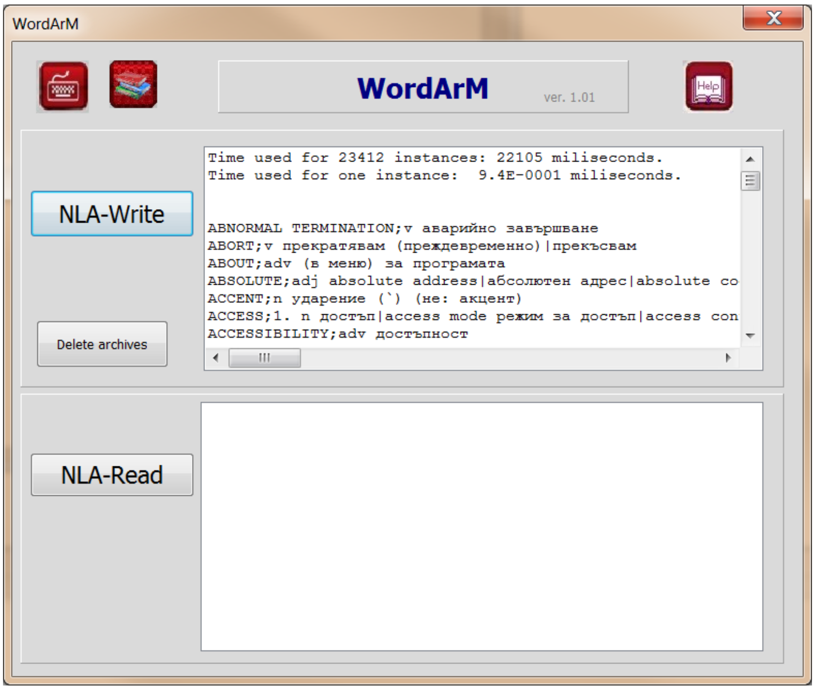


Figure 87. The WordArM panel for working in automated mode

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.



```
Time used for 23412 instances: 22105 miliseconds.
Time used for one instance:  9.4E-0001 miliseconds.


ABNORMAL TERMINATION;v аварийно завършване
ABORT;v прекратявам (преждевременно)|прекъсвам
ABOUT;adv (в меню) за програмата
ABSOLUTE;adj absolute address|абсолютен адрес|absolute co
ACCENT;n ударение (`) (не: акцент)
ACCESS;1. n достъп|access mode режим за достъп|access con
ACCESSIBILITY;adv достъпност
```

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.

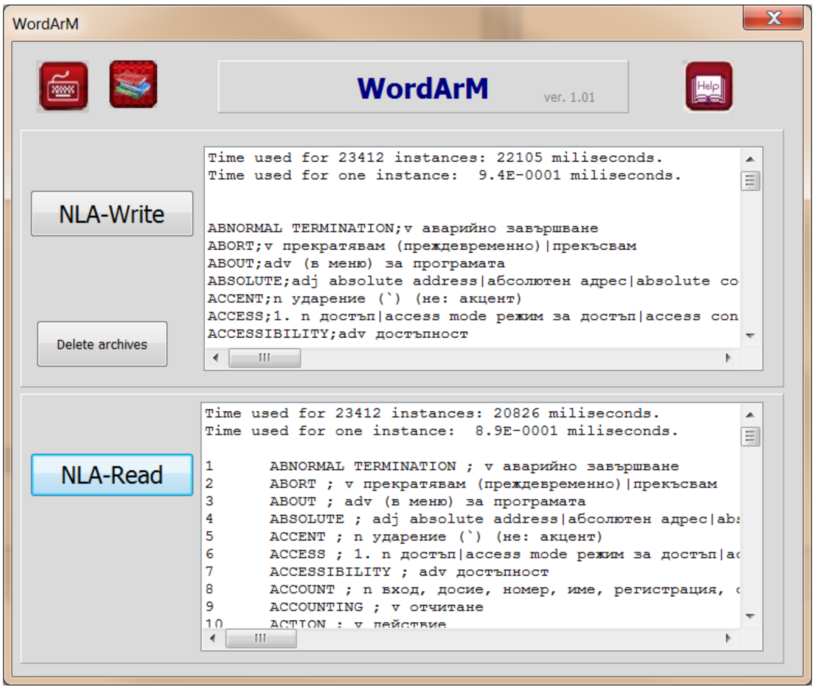
The output file is in CSV file format. Its records have the next format:

<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.



```
Time used for 23412 instances: 22105 miliseconds.
Time used for one instance:  9.4E-0001 miliseconds.


ABNORMAL TERMINATION;v аварийно завършване
ABORT;v прекратявам (преждевременно)|прекъсвам
ABOUT;adv (в меню) за програмата
ABSOLUTE;adj absolute address|абсолютен адрес|absolute co
ACCENT;n ударение (`) (не: акцент)
ACCESS;1. n достъп|access mode режим за достъп|access con
ACCESSIBILITY;adv достъпност
```

```
Time used for 23412 instances: 20826 miliseconds.
Time used for one instance:  8.9E-0001 miliseconds.

1       ABNORMAL TERMINATION ; v аварийно завършване
2       ABORT ; v прекратявам (преждевременно)|прекъсвам
3       ABOUT ; adv (в меню) за програмата
4       ABSOLUTE ; adj absolute address|абсолютен адрес|abs
5       ACCENT ; n ударение (`) (не: акцент)
6       ACCESS ; 1. n достъп|access mode режим за достъп|ac
7       ACCESSIBILITY ; adv достъпност
8       ACCOUNT ; n вход, досие, номер, име, регистрация, 
9       ACCOUNTING ; v отчитане
10      ACTION ; v действие
```

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.

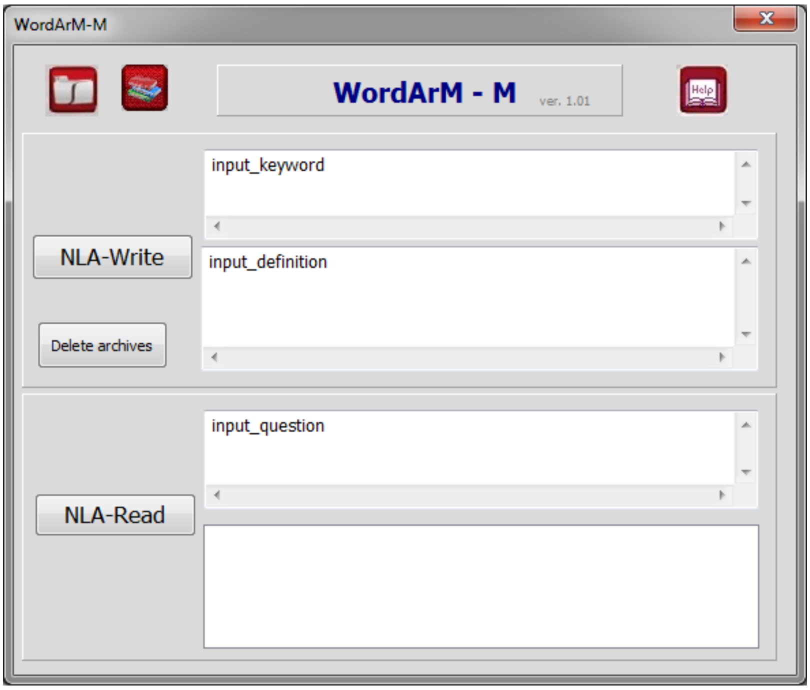


Figure 90. The WordArM panel for working in manual mode

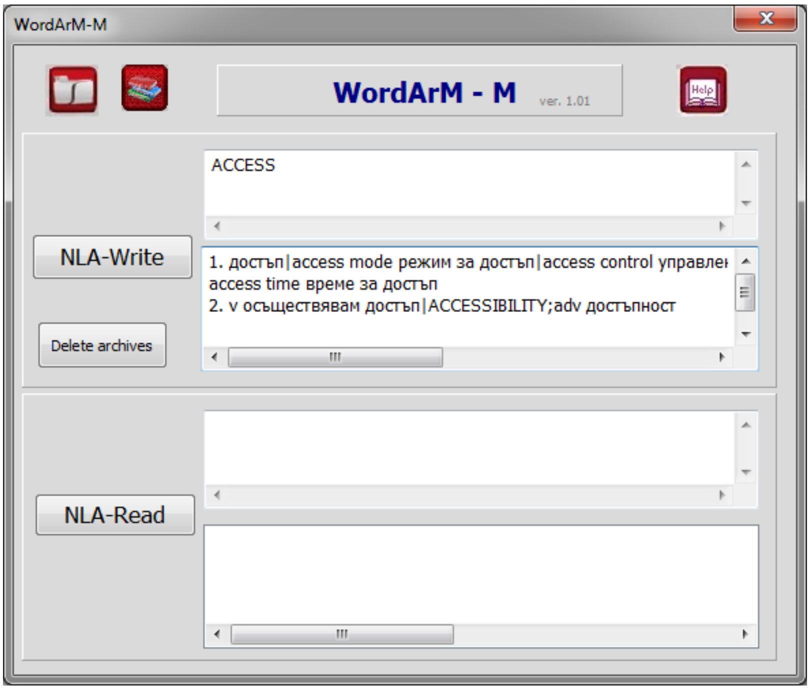
By “**NLA-Write**” button the function for storing definitions from the form can be activated.

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.

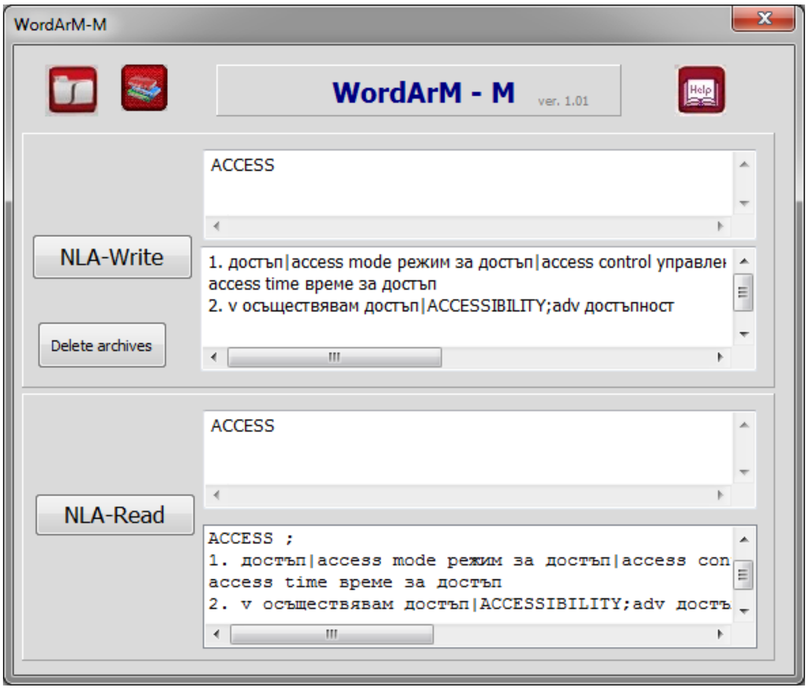


ACCESS

1. достъп|access mode режим за достъп|access control управлен
access time време за достъп
2. v осъществявам достъп|ACCESSIBILITY;adv достъпност

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.



ACCESS

1. достъп|access mode режим за достъп|access control управлен
access time време за достъп
2. v осъществявам достъп|ACCESSIBILITY;adv достъпност

ACCESS

```
ACCESS ;
1. достъп|access mode режим за достъп|access con
access time време за достъп
2. v осъществявам достъп|ACCESSIBILITY;adv достъ
```

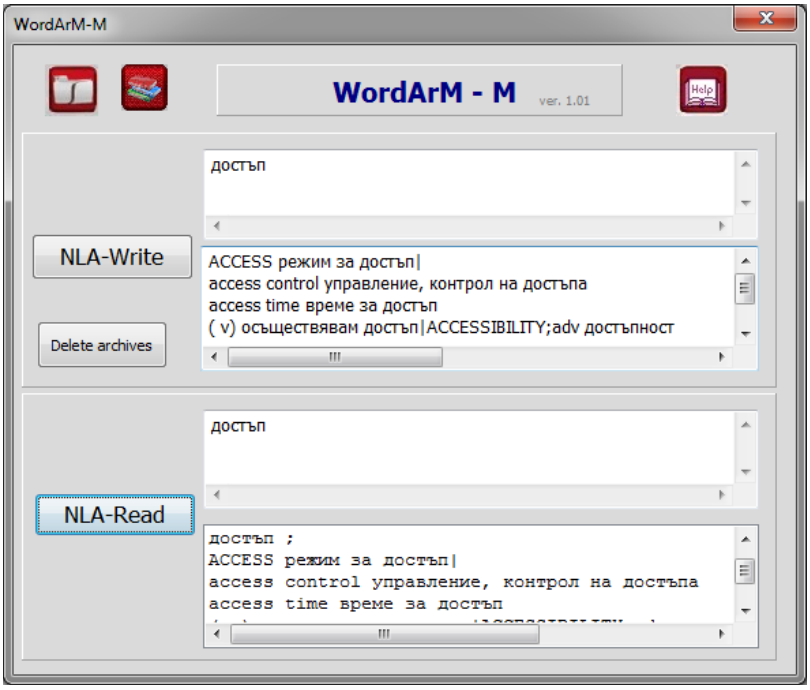
Figure 92. Manual output of the concept and its definition

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).



```
достъп
```

```
ACCESS режим за достъп|
access control управление, контрол на достъпа
access time време за достъп
( v) осъществявам достъп|ACCESSIBILITY;adv достъпност
```

```
достъп
```

```
достъп ;
ACCESS режим за достъп|
access control управление, контрол на достъпа
access time време за достъп
```

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:

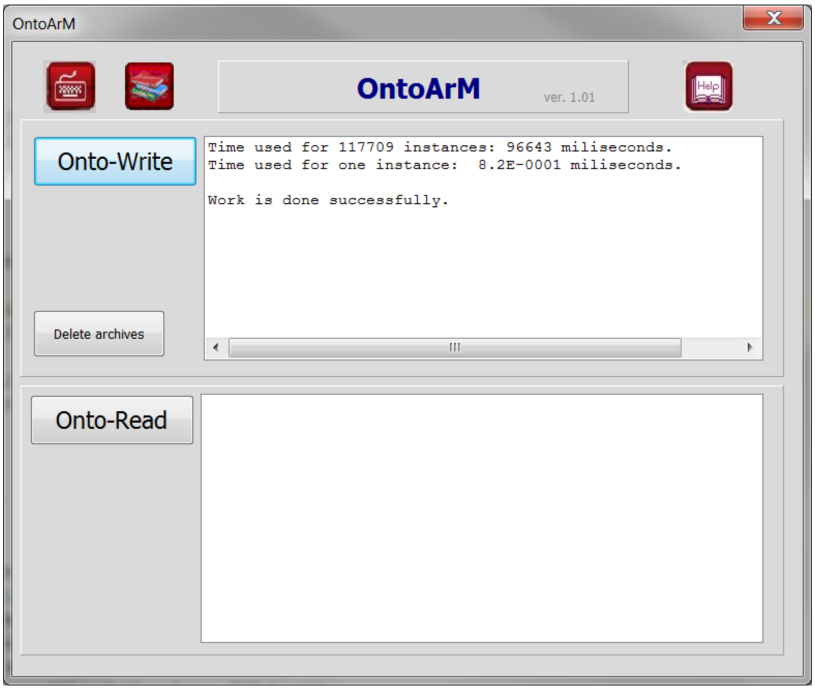
<subject>;<relation>;<object><CR>.

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.



```
Time used for 117709 instances: 96643 miliseconds.
Time used for one instance:  8.2E-0001 miliseconds.

Work is done successfully.
```

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:

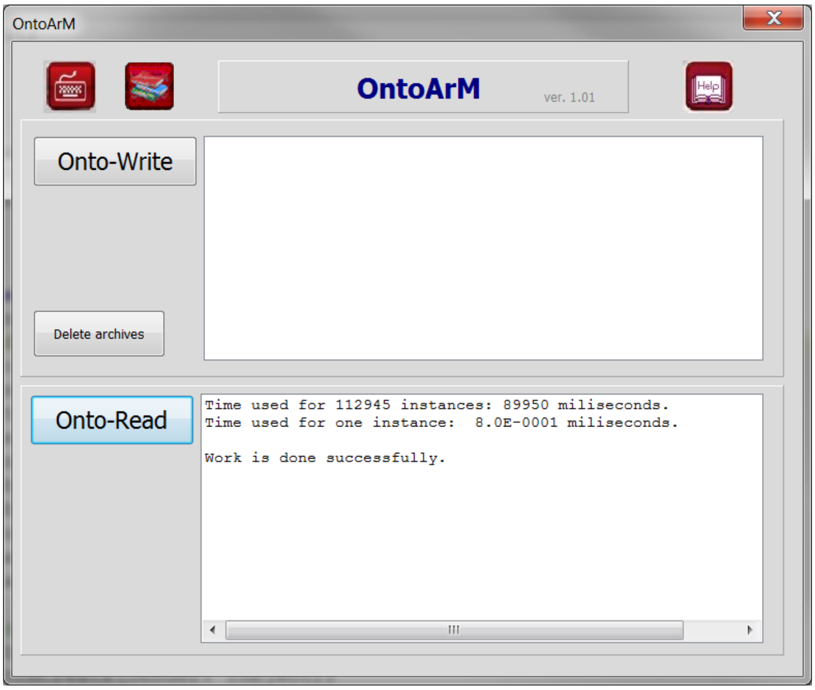
<subject>;<relation>;<object><CR>.

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.



```
Time used for 112945 instances: 89950 miliseconds.
Time used for one instance:  8.0E-0001 miliseconds.

Work is done successfully.
```

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 (Help) 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:

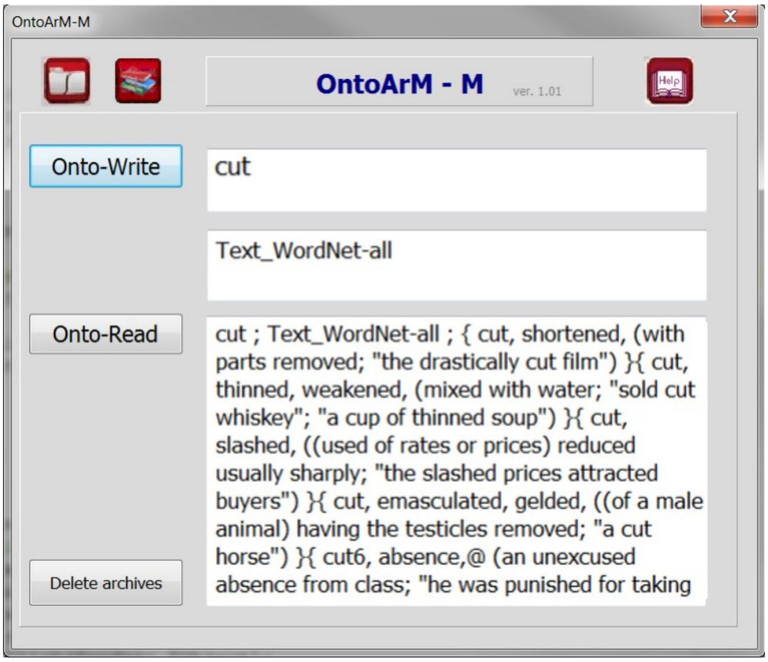
(1) Transform the concept into path;

(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 (Help) 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.

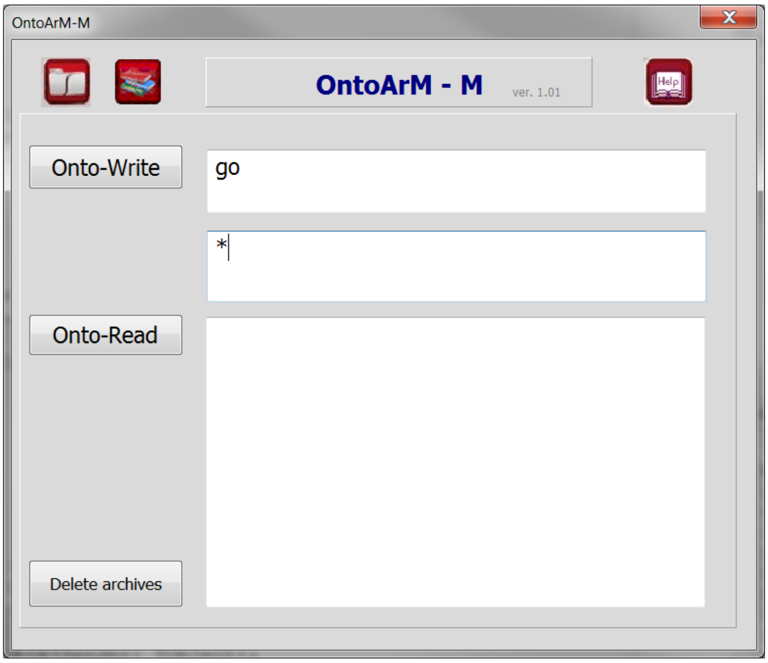


cut

Text_WordNet-all

cut ; Text_WordNet-all ; { cut, shortened, (with parts removed; "the drastically cut film") }{ cut, thinned, weakened, (mixed with water; "sold cut whiskey"; "a cup of thinned soup") }{ cut, slashed, ((used of rates or prices) reduced usually sharply; "the slashed prices attracted buyers") }{ cut, emasculated, gelded, ((of a male animal) having the testicles removed; "a cut horse") }{ cut6, absence,@ (an unexcused absence from class; "he was punished for taking

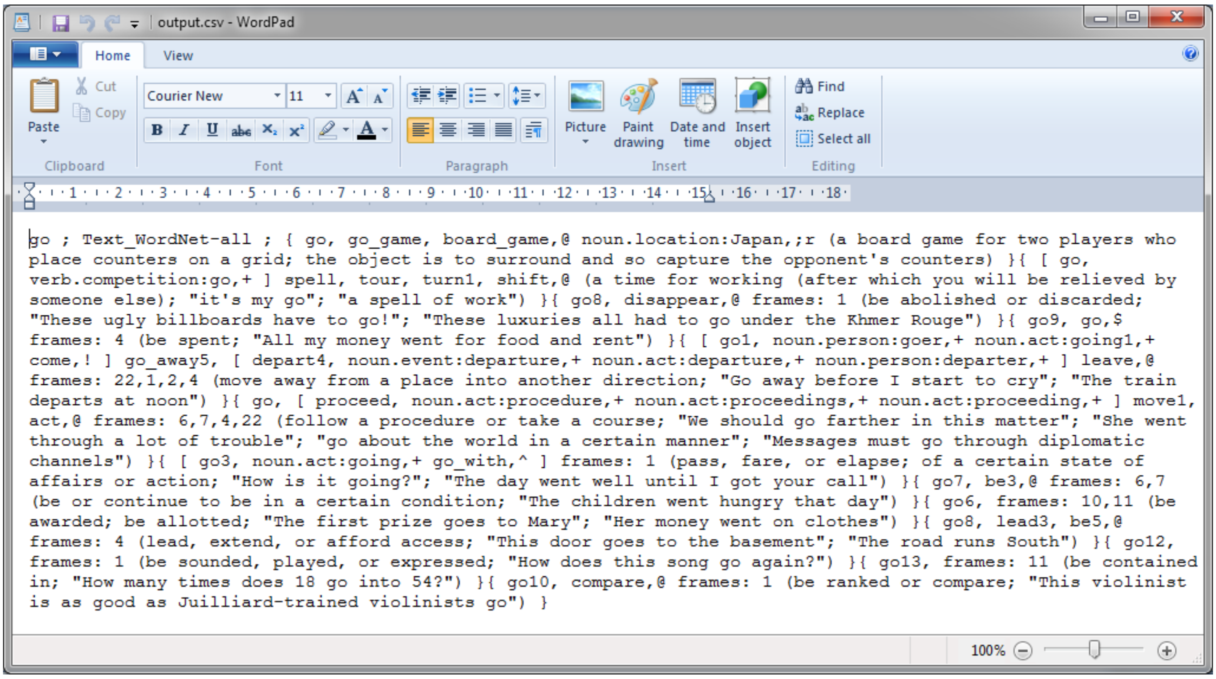
Figure 96. Manual input of the RDF-triple



```
go
```

```
*
```

Figure 97. Manual reading the RDF-triple



```
go ; Text_WordNet-all ; { go, go_game, board_game,@ noun.location:Japan,;r (a board game for two players who
place counters on a grid; the object is to surround and so capture the opponent's counters) }{ [ go,
verb.competition:go,+ ] spell, tour, turn1, shift,@ (a time for working (after which you will be relieved by
someone else); "it's my go"; "a spell of work") }{ go8, disappear,@ frames: 1 (be abolished or discarded;
"These ugly billboards have to go!"; "These luxuries all had to go under the Khmer Rouge") }{ go9, go,$
frames: 4 (be spent; "All my money went for food and rent") }{ [ go1, noun.person:goer,+ noun.act:going1,+
come,! ] go_away5, [ depart4, noun.event:departure,+ noun.act:departure,+ noun.person:departer,+ ] leave,@
frames: 22,1,2,4 (move away from a place into another direction; "Go away before I start to cry"; "The train
departs at noon") }{ go, [ proceed, noun.act:procedure,+ noun.act:proceedings,+ noun.act:proceeding,+ ] move1,
act,@ frames: 6,7,4,22 (follow a procedure or take a course; "We should go farther in this matter"; "She went
through a lot of trouble"; "go about the world in a certain manner"; "Messages must go through diplomatic
channels") }{ [ go3, noun.act:going,+ go_with,^ ] frames: 1 (pass, fare, or elapse; of a certain state of
affairs or action; "How is it going?"; "The day went well until I got your call") }{ go7, be3,@ frames: 6,7
(be or continue to be in a certain condition; "The children went hungry that day") }{ go6, frames: 10,11 (be
awarded; be allotted; "The first prize goes to Mary"; "Her money went on clothes") }{ go8, lead3, be5,@
frames: 4 (lead, extend, or afford access; "This door goes to the basement"; "The road runs South") }{ go12,
frames: 1 (be sounded, played, or expressed; "How does this song go again?") }{ go13, frames: 11 (be contained
in; "How many times does 18 go into 54?") }{ go10, compare,@ frames: 1 (be ranked or compare; "This violinist
is as good as Juilliard-trained violinists go") }
```

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:

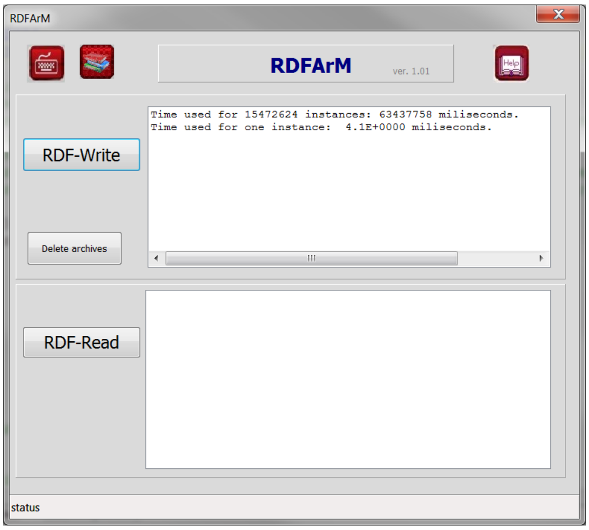
<subject><relation><object><context> . <CR>

- for triples:

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

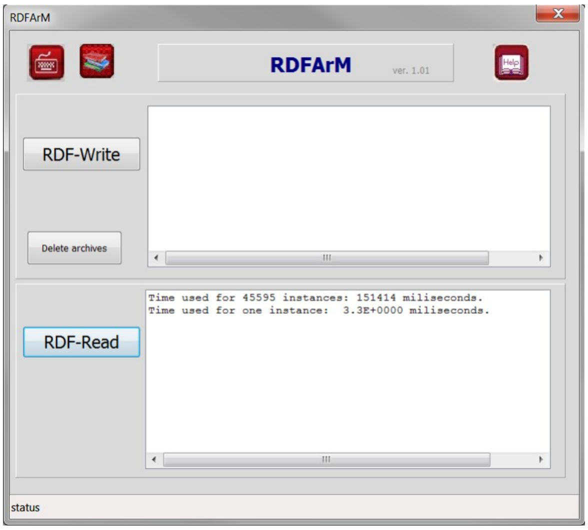
In the window next to the RDF-Read button, two informative lines are shown (Figure 100):

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



```
Time used for 15472624 instances: 63437758 miliseconds.
Time used for one instance:  4.1E+0000 miliseconds.
```

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



```
Time used for 45595 instances: 151414 miliseconds.
Time used for one instance:  3.3E+0000 miliseconds.
```

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 (Help) 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
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
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
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
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
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
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
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
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
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
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
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
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.

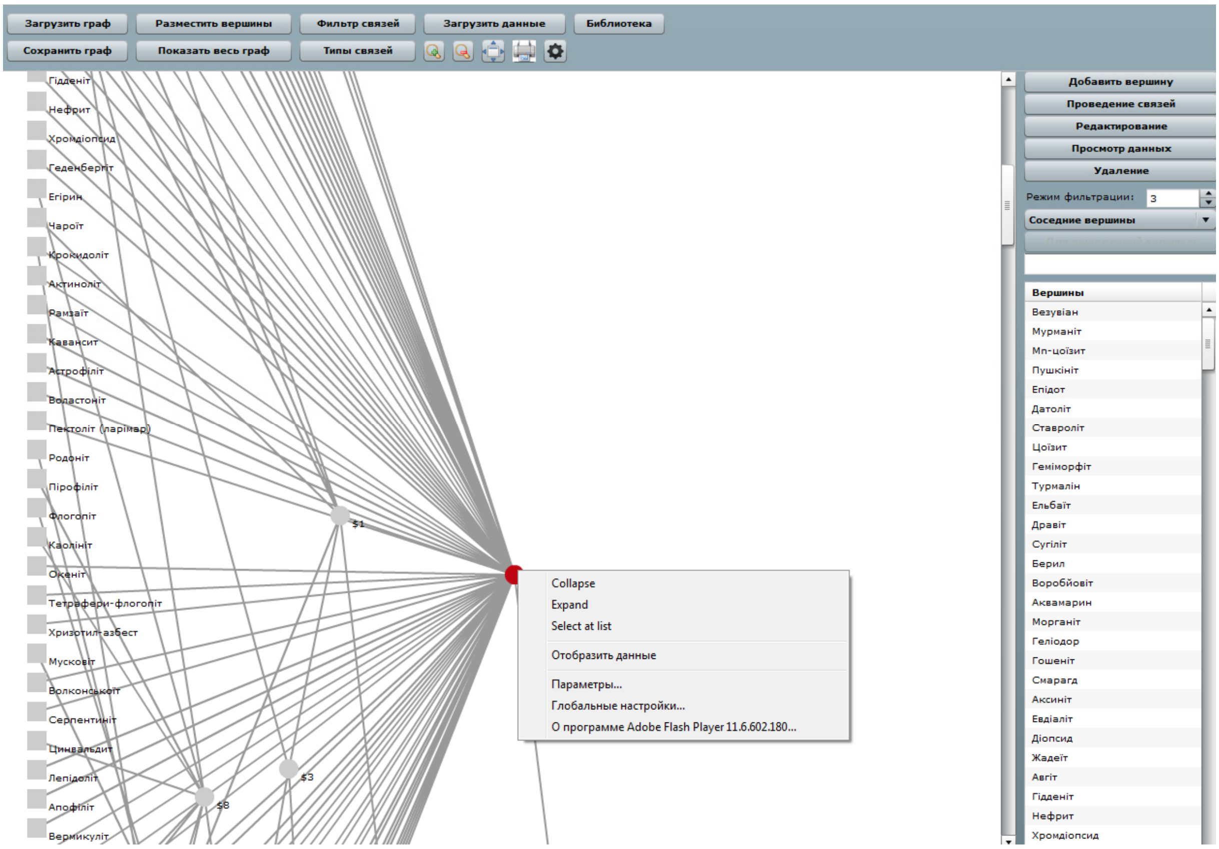


Figure 108. A visualization of a Growing pyramidal network

An example of the ontological graph generated by the ICON Ontological Editor is presented on Figure 109. This visualization of our sample graph (Figure 18) is created by this editor.

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

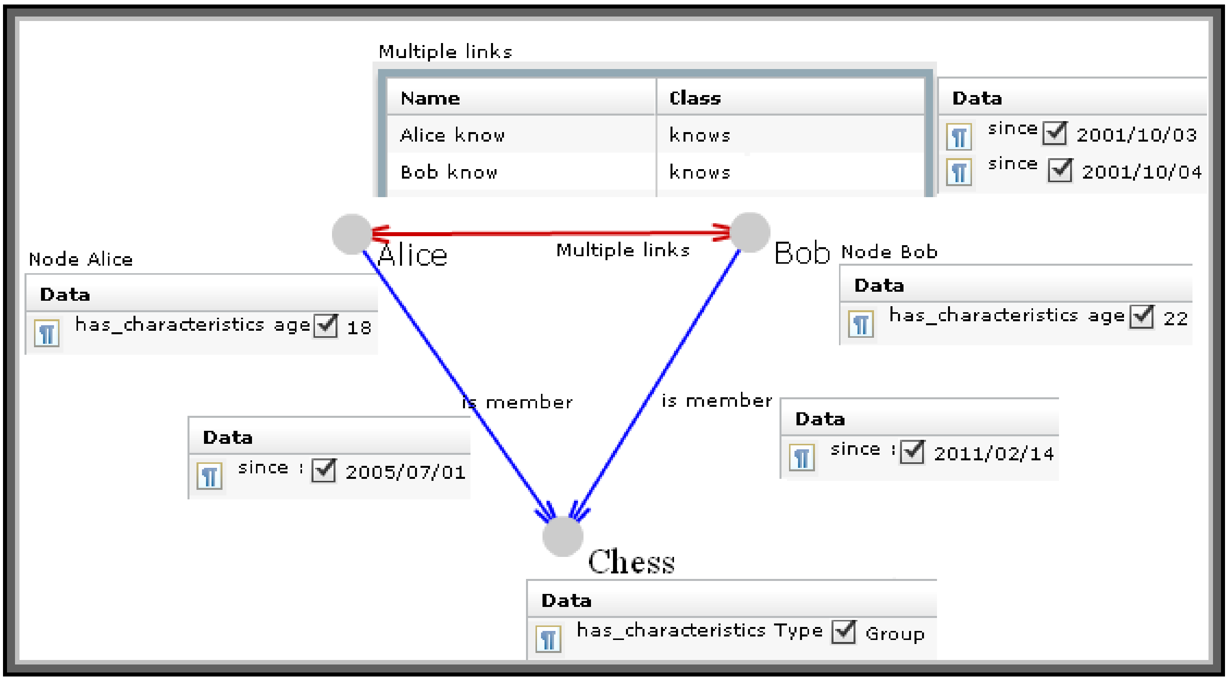


Figure 109. Screenshot from the ICON Ontological Editor

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.”)

Appendix B: Brief descriptions of the main mentioned tools

B1. Protégé 4.2

http://protege.stanford.edu/

Protégé was 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 allows a user to simultaneously work on classes and instances. This is provided for by a uniform GUI whose top level is composed of overlapping tabs for compact representation [protégé, 2012; protégé-owl, 2012].

Protégé platform supports two main ways of modeling ontologies:

- *Protégé-Frames editor*: enables users to build and distribute ontologies, which are based on frame structures corresponding to "Open Knowledge Base Connectivity" (OKBC) protocol;
- *Protégé-OWL editor*: enables users to build ontologies on the Semantic Web, especially the using W3C's Web Ontology Language (OWL) [OWL, 2004].

Protégé Basic features:

- Import format - XML, RDF(S) and XML Schema;
- Export format - XML, RDF(S), XML Schema, FLogic, CLIPS and Java HTML;
- Graph view - Via GraphViz plug-in (browsing of classes and global properties); Via Jambalaya plug-in (nested graph view);
- Consistency check - Via plug-ins (PAL and FaCT);
- Limited multi-user support - Protégé has some multi-user capabilities added to it. It is intended for experienced Protégé users. Multiple users can read the same database and make incremental changes or changes that don't conflict with one another. However, there's no support for multiple users trying to modify the same elements of a knowledge base or notification of changes made by other users. Concurrent changes to the same section will cause severe problems;

- Web support - Via Protégé-OWL plug-in; Protégé doesn't provide direct support for accessing knowledge base from the web, but it can easily be done. A number of users have communicated with Protégé knowledge bases from the Web via servlets. Protégé can be run as an applet.

Additional features:

- Merging - Via Anchor-PROMPT plug-in;
- Not support to add a new basic type;
- Extensible plug-in architecture;
- Ontology storage - File and DBMS (JDBC).

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.

Originally, Protégé has a single table that stores entire contents of the knowledge base which is developed as a frame based one [protégé, 2012].

The frame table has a fixed number of columns which are listed below in Table 74. It includes classes, slots, facets and instances. The Protégé meta-class architecture is used explicitly in the table to simplify things: all classes, slots, and facets are treated as frames.

Each entry in the database corresponds to a frame in Protégé. Classes have slots such as ":DIRECT_SUPERCLASS" to maintain the inheritance hierarchy. All frames have a “:NAME” slot which contains name of frame.

Table 74. Protégé database format

Column	Description
frame - [integer]	frame id Frame ID's < 10000 are reserved for the system. The frame ids for system frames are declared in the file: edu.stanford.smi.protege.model.Model.java
frame_type - [smallint]	same as "value_type" but for the frame column
slot - [integer]	slot frame id
facet - [integer]	facet frame id (0 if not a facet value)
is_template - [smallint]	0 => value is OKBC "own", 1 => value is OKBC "template"
value_index - [integer]	number used to maintain relative ordering of slot_or_facet_value entries for a frame-slot(-facet) combination
value_type - [smallint]	number used to indicate the "type" of the value stored in slot_or_facet_value. The number-to-type conversion is given in the file: edu.stanford.smi.protege.storage.database.DatabaseUtils.java
slot_or_facet_value - [varchar(N)]	facet value if facet is not 0, slot value otherwise. Holds values of length that will fit in a varchar (typically <= 255)
long_slot_or_facet_value - [longvarchar]	same as slot_or_facet_value but holds values too long to fit in slot_or_facet_value

In the case of the superclass and subclass relations, Protégé stores duplicated information.

For example with class A it stores that its subclass is B and with B it stores that its superclass is A.

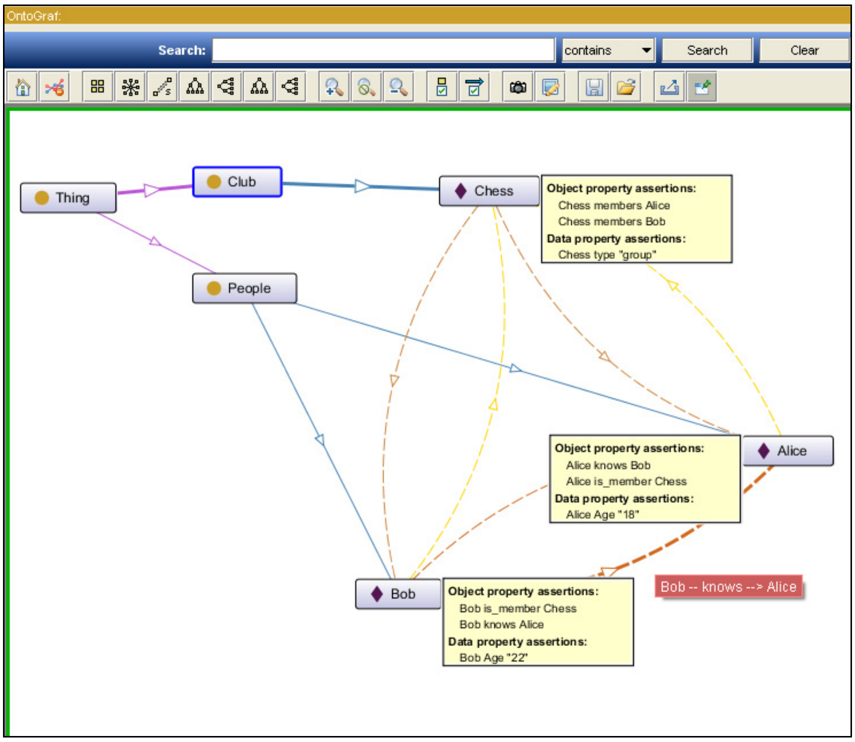
Maintaining separate records for these relations is necessary to maintain the ordering of both subclasses and superclasses. So while the "slot value" information is indeed duplicated in these records, the "index" information is unique (Subclass ordering is a user-interface feature that a number of users have requested. Protégé attaches no meaning to the ordering of superclasses or subclasses.) [protégé, 2012].

To illustrate the using of Protégé we use our sample graph.

For easy reading, below we reproduce the description by triples of the sample graph.

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**

The visualization is shown at Figure 110. Some information could not be viewed (like dates and etc.). The corresponded OWL and RDF descriptions of the sample graph are given in Table 75 and Table 76.



Object property assertions:
Chess members Alice
Chess members Bob
Data property assertions:
Chess type "group"

Object property assertions:
Alice knows Bob
Alice is_member Chess
Data property assertions:
Alice Age "18"

Object property assertions:
Bob is_member Chess
Bob knows Alice
Data property assertions:
Bob Age "22"

Figure 110. Protégé graphical representation of the sample graph

Table 75. 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 76. 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 -->
```

This example show that the sentence "*OWL and RDF are easy readable by humans*" is not the all truth. Linearizing the information is suitable solution for telecommunication and computer processing, but it is not easy understandable by humans.

Let remember multi-layer representation of the sample graph. What is presented in four rows below is the same as one presented on two (OWL) or four and half pages (RDF) above.

	space addresses		
file name	*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**	

The main conclusion is that we still need new approaches for representing the knowledge which will correspond both to human and machine possibilities. Because of this, in addition to Protégé, ICON implements the NL-addressing features for storing the dictionaries, thesauruses and ontologies.

B2. SPARQL

The “Simple Protocol and RDF Query Language” (SPARQL) is a SQL-like language for querying RDF data. SPARQL allows querying for triples from an RDF database (or triple store). RDF doesn't use foreign and primary keys either. It uses URIs, the standard reference format for the World Wide Web. By using URIs, a triple store immediately has the potential to link to any other data in any triple store. That plays to the distributed strengths of the Web.

Because triple stores are large amorphous collections of triples, SPARQL queries by defining a template for matching triples, called a Graph Pattern. To get data out of the triple store using SPARQL, you need to define a pattern that matches the statements in the graph. Those will be questions like this: find me the subjects of all the statements that say 'plays guitar'. Example below shows a query over data defined using the ontology about music:

```
PREFIX : <http://aabs.purl.org/music#>
SELECT ?instrument
WHERE {:andrew :playsInstrument ?instrument }
```

The query says "find all the triples that have a subject of :andrew and a predicate of :playsInstrument, then get the objects of the matching triples and return them" [SPARQL, 2013].

Another example of a SELECT query follows.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
          ?x foaf:mbox ?mbox . }
```

The first line defines namespace prefix, the last two lines use the prefix to express a RDF graph to be matched. Identifiers beginning with question mark ? identify variables. In this query, we are looking for resource ?x participating in triples with predicates foaf:name and foaf:mbox and want the subjects of these triples. Syntactic shortcuts of TURTLE can be used in the matching part.

In addition to specifying graph to be matched, constraints can be added for values using FILTER construct. An example of string value restriction is FILTER regex(?mbox, "company") that specifies regular expression query. An example of number value restriction is FILTER (?price < 20) that specifies that ?price must be less than 20. A few special operators are defined for the FILTER construct. They include isIRI for testing whether variable is IRI/URI, isLiteral for testing whether variable is literal, bound to test whether variable was bound and others - see the specification.

The matching part of the query may include OPTIONAL triples. If the triple to be matched is optional, it is evaluated when it is present, but the matching does not fail when it is not present. Optional sections may be nested. It is possible to make UNION of multiple matching graphs - if any of the graphs matches, the match will be returned as a result. The FROM part of the query is optional and may specify the RDF dataset on which query is performed.

The sequence of result may be modified using the following keywords with the meaning similar to SQL:

- ORDER BY - ordering by variable value;
- DISTINCT - unique results only;
- OFFSET - offset from which to show results;
- LIMIT - the maximum number of results.

There are four query result forms. In addition to the possibility of getting the list of values found it is also possible to construct RDF graph or to confirm whether a match was found or not.

- SELECT - returns the list of values of variables bound in a query pattern;
- CONSTRUCT - returns an RDF graph constructed by substituting variables in the query pattern;
- DESCRIBE - returns an RDF graph describing the resources that were found;
- ASK - returns a Boolean value indicating whether the query pattern matches or not.

The CONSTRUCT form specifies a graph to be returned with variables to be substituted from the query pattern, such as in the following example that will return graph saying that Alice knows last two people when ordered by alphabet from the given URI (the result in the RDF graph is not ordered, it is a graph and so the order of triples is not important).

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT { <http://example.org/person#Alice> foaf:knows ?x }
FROM <http://example.org/foaf/people>
WHERE { ?x foaf:name ?name }
ORDER BY desc(?name)
LIMIT 2
```

The DESCRIBE form will return information about matched resources in a form of an RDF graph. The exact form of this information is not standardized yet, but usually a blank node closure like, for example, Concise Bounded Description (CBD) is expected. In short, all the triples that have the matched resource in the object are returned; when a blank node is in the subject, then the triples in which this node participates as object are recursively added as well.

The ASK form is intended for asking yes/no questions about matching - no information about matched variables is returned, the result is only indicating whether matching exists or not [Obitko, 2007a].

B3. Storage characteristics of analyzed RDF triple stores

➢ *B3.1. DBMS based approaches*

✓ *3store*

3Store is MySQL based triple store, currently holding over 30 million RDF triples used by a range of Knowledgeable Services developed within the Advanced Knowledge Technologies project (AKT) [AKT Project, 2013] led by Nigel Shadbolt from Southampton. It is concerned with the management of the knowledge life cycle.

3store is a core C library that uses MySQL to store its raw RDF data and caches. The library offers OKBC and RDQL query interfaces, over HTTP (via an Apache web server module), or directly though the C library.

The server software itself does not expose any interfaces directly to the user, but it can be queried by a number of services, including a column based view and a direct RDF browser.

3store is distributed under the Gnu General Public License, and is available from Source forge [3storeSF, 2013].

✓ *Jena*

Jena is a Java toolkit for manipulating RDF models which has been developed by Hewlett-Packard Labs [Jena, 2013]. It has excellent support for RDQL queries, but does not provide an OKBC interface (however, given the level of RDQL support provided, the addition of an OKBC compatibility layer would be straightforward for the portion of the OKBC API that was implemented in the previous version of 3store).

Jena is a java framework for building semantic web applications. Jena implements APIs for dealing with Semantic Web building blocks such as RDF and OWL. Jena's fundamental class for users is the Model, an API for dealing with a set of RDF triples. A Model can be created from the file system or from a remote file. Using JDBC, it can also be tied to an existing RDBMS such as MySQL or PostgreSQL.

SDB is a component of Jena [CTS, 2012]. It provides for scalable storage and query of RDF datasets using conventional SQL databases for use in standalone applications, J2EE, and other application frameworks. The storage, as mentioned, is provided by an SQL database and many databases are supported, both open source and proprietary. An SDB store can be accessed and managed with the provided command line scripts and via the Jena API. In addition on the fine grain access provided by the Jena API, SDB can be coupled with the web-server - ‘Joseki’ - which is SPARQL query server. This enables an SDB store to be queried over HTTP. Jena recently introduced a non-transactional native store called TDB [CTS, 2012].

The attempt to import one of hyphen's larger RDF files (comprising around 5% of the data) into Jena, using its default MySQL back-end, had not completed after 24 hours it the import (the preliminary indications are that it repeatedly refreshes its database indexes during the import). From this experience, conclusion is that Jena is *unsuitable* for storing large volume of data [Harris & Gibbins, 2003].

✓ *RDFSuite*

RDF Suite is a set of high-level and scalable services enabling the realization of the full potential of the Semantic Web.

The RDF Suite is used by academics and software developers to produce scalable applications that rely on validating RDF/S compatible data for the Semantic Web.

RDF Suite supports the storage, query and update of semantic conceptualizations and thus can be used in order to store and better track and understand changes and evolution in the way users create and understand knowledge [RDF Suite, 2013].

✓ *Sesame*

Sesame is a de-facto standard framework for processing RDF data. This includes parsing, storing, inference and querying of/over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions [Sesame, 2012].

Sesame can be deployed on top of a variety of storage systems (relational databases, in-memory, file systems, keyword indexers, etc.), and offers a large scale of tools to developers to leverage the power of RDF and related standards. Sesame fully supports the SPARQL query language for expressive querying and offers transparent access to remote RDF repositories using the exact same API as for local access. Finally, Sesame supports all main stream RDF file formats, including RDF/XML, Turtle, N-Triples, TriG and TriX.

Sesame is an open source framework for storage, inference and querying of RDF data. Sesame matches the features of Jena with the availability of a connection API, inference support, availability of a web server and SPARQL endpoint. Like Jena SDB it provides support for multiple backends like MySQL and Postsgre.

✓ *4store*

4store is an RDF database that was designed by Steve Harris and developed at “Garlik” Co. to underpin their Semantic Web applications. It has been used by Garlik as their primary RDF platform for three years, and has proved itself to be robust and secure. 4store makes use of the Raptor and Rasqal libraries that have been developed for Redland [4store, 2013].

4store is a database storage and query engine that holds RDF data. 4store's main strengths are its performance, scalability and stability. It does not provide many features over and above RDF storage and SPARQL queries, but if you are looking for a scalable, secure, fast and efficient RDF store, then 4store should be on your shortlist.

✓ *Oracle*

Oracle Spatial and Graph RDF Semantic Graph (Formerly Oracle Database Semantic Technologies) is an open, standards-based, scalable, secure, reliable, and performance RDF management platform. Based on a graph data model, RDF data (triples) are persisted, indexed and queried, like other object-relational data types [Oracle, 2013].

Application developers use the power of the Oracle Database to design and develop a wide range of semantic-enhanced business applications in areas that include intelligence, law enforcement, integrated bioinformatics and health care informatics, finance, web social network, and media, games, and content management.

Oracle provides the industry's leading spatial database management platform. Oracle Spatial and Graph option includes advanced features for spatial data and analysis as well as for physical, network and social graph applications. The geospatial data features support complex Geographic Information Systems (GIS) applications, enterprise applications and location-based services applications. The graph features include a network data model (NDM) graph to model and analyze link-node graphs to represent physical and logical networks used in industries such as transportation and utilities. In addition, Oracle Spatial and Graph includes support for RDF semantic graphs used in social networks and social interactions.

The RDF semantic graph feature of Oracles Spatial and Graph provides a robust and standards-based platform on which to build semantic solutions. Identifying the business requirements and benefits, project requirements, application functionality, and design considerations helps in planning and discussing the project with Oracle. This information can help Oracle provide recommendations and best practices to facilitate a successful project.

Storing RDF data in a relational database requires an appropriate table design. There are different approaches that can be classified in generic schemas, i.e. schemas that do not depend on the ontology, and ontology specific schemas.

Current Object-oriented databases (ORDBMS) provide the suitable facility which allows for a better modeling of the subclass and sub-property relationships [Broekstra, 2005; Alexaki et al, 2001].

DBMS "Oracle" offers another object-relational feature: an own data type to store RDF based on a graph data model [oracledb, 2012; OSTI, 2009]. RDF triples can be persisted, indexed and queried, similar to other object-relational data types.

Although the RDF model has several object-oriented characteristics and most RDF stores are internally working with an object model, approaches to store RDF data and schema information using object database management systems (ODBMS) are rarely known. (Object-) Relational databases are still predominant, when large amounts of data have to be persisted on a server and object databases did not and will most probably not replace them. However, new developments of ODBMS may show some advantages over RDBMS in certain applications, e.g. for embeddable persistence solutions in mobile devices [Hertel et al, 2009].

A special attention has to be paid to the Oracle "Berkeley DB" as a tool for storing RDF information [Berkeley DB, 2012]. Oracle Berkeley DB is the industry-leading open source, embeddable storage engine that provides developers a fast, reliable, local database with zero administration. Oracle Berkeley DB is a library that links directly into your application. Your application makes simple function calls, rather than sending messages to a remote server, eliminating the performance penalty of client-server architectures.

Berkeley DB has a number of key advantages over comparable systems. It is simple to use, supports concurrent access by multiple users, and provides industrial-strength transaction support, including surviving system and disk crashes.

Berkeley DB supports three access methods: B+tree, Extended Linear Hashing (Hash), and Fixed- or Variable- length Records (Recno). All three operate on records composed of a key and a data value. In the B+tree and Hash access methods, keys can have arbitrary structure. In the Recno access method, each record is assigned a record number, which serves as the key. In all the access methods, the value can have arbitrary structure. The programmer can supply comparison or hashing functions for keys, and Berkeley DB stores and retrieves values without interpreting them. All of the access methods use the host file system as a backing store [Olson et al, 1999] (Figure 111).

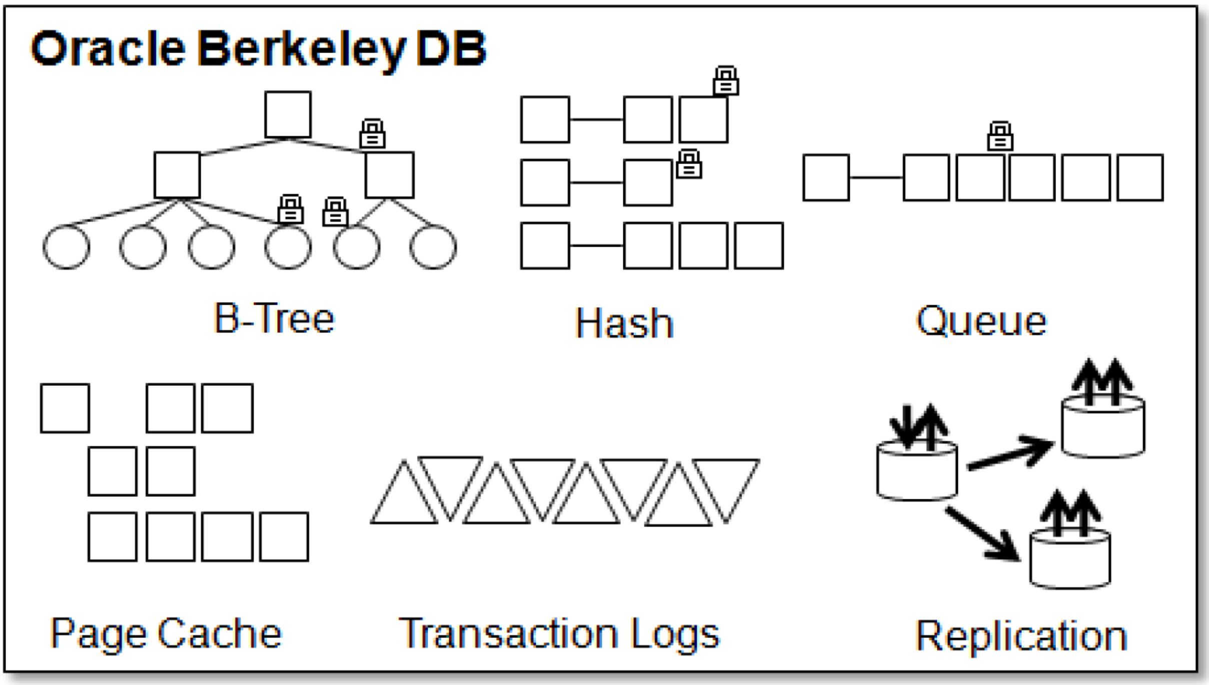


Figure 111. Main features of Oracle Berkeley DB

➢ *B3.2. Multiple indexing frameworks*

✓ *YARS*

YARS (Yet Another RDF Store) [YARS, 2013] is a data store for RDF in Java and allows for querying RDF based on a declarative query language, which offers a somewhat higher abstraction layer than the APIs of RDF toolkits such as Jena or Redland. YARS uses Notation3 as a way of encoding facts and queries.

The main requirement for YARS is to enable fast storage and retrieval of large amounts of RDF (in the order of millions of triples) while keeping a small footprint and a lightweight architecture approach.

There is a JDBC-like API for YARS available that can be used to issue calls either locally or via HTTP within Java programs. Active RDF is a library for accessing RDF data from within Ruby programs by addressing RDF resources, classes, properties, etc. programmatically, without queries. RDF2Go is an abstraction over triple (and quad) stores and has support for YARS as a backend store.

The YARS system combines methods from Information Retrieval and Databases to allow for better query answering performance over RDF data. It stores RDF data persistently by using six B+ tree indices. It not only stores the subject, the predicate and the object, but also the context information about the origin of the data. Each element of the corresponding quad (i.e., 4-uplet) is encoded in a dictionary storing mappings from literals and URIs to object IDs (OIDs-stored as number identifiers for compactness). To speed up keyword queries, the lexicon keeps an inverted index on string literals to allow fast full-text searches. In each B+ tree, the key is a concatenation of the subject, predicate, object and context. The six indices constructed cover all the possible access patterns of quads in the form (s, p, o, c) where c is the context of the triple (s, p, o). This representation allows fast retrieval of all triple access patterns. Thus, it is also oriented towards simple statement-based queries and has limitations for efficient processing of more complex queries. The proposal sacrifices space and insertion speed for query performance since, to retrieve any access pattern with a single index lookup, each triple is encoded in the dictionary six times, in different sorting order. Inference is not supported.

✓ *Kowari*

Kowari^{TM} is an Open Source, massively scalable, transaction-safe, purpose-built database for the storage and retrieval of metadata [Kowari, 2004].

Much like a relational database, one stores information in Kowari and retrieves it via a query language. Unlike a relational database, Kowari is optimized for the storage and retrieval of many short statements (in the form of subject-predicate-object, like "Kowari is fun" or "Kowari imports RDF"). Kowari is not based on a relational database due to the large numbers of table joins encountered by relational systems when dealing with metadata. Instead, Kowari is a completely new database optimized for metadata management.

Kowari is implemented in the Java programming language and is 100% Java. It depends on standard Java packages available from Sun Microsystems^{TM}. Kowari also includes Java code from other projects. Details may be found on the Legal page [Kowari, 2004].

The Kowari system uses an approach similar to YARS. Indeed, the RDF statements are also stored as quads in which the first three items form a standard RDF triple and the fourth describes in which model the statement appears. The approach also uses six different orderings of quad elements acting as a compound index, and independently contains all the statements of the RDF store. In this ordering, the four quad elements can be arranged such that any collection of one to four elements can be used to find any matching statement or group of statements. However, Kowari uses a hybrid of AVL and B trees instead of B+ trees for multiple indexing purposes. Kowari solution also envisions simple statement-based queries like YARS.

✓ *Virtuoso*

OpenLink Virtuoso is the first CROSS PLATFORM Universal Server to implement Web, File, and Database server functionality alongside Native XML Storage, and Universal Data Access Middleware, as a single server solution [Virtuoso, 2013]. Virtuoso is a native triple store available in both open source and commercial licenses. It provides command line loaders, a connection API, and support for SPARQL and web server to perform SPARQL queries and uploading of data over HTTP. A number of evaluations have tested virtuoso and found it to be scalable to the region of 1B+ triples.

The commercial system Virtuoso stores quads combining a graph to each triple (s, p, o). It, thus, conceptually stores the quads in a triples table expanded by one column. The columns are **g** for graph, **p** for predicate, **s** for subject and **o** for object. While technically rooted in an RDBMS, it closely follows the model of YARS but with fewer indices. The quads are stored in two covering indices, (g, s, p, o) and (o, g, p, s), where the URI's are dictionary encoded. Several further optimizations are added, including bitmap indexing. In this approach, the use of fewer indices tips the balance slightly towards insertion performance from query performance, but still favors query one.

✓ *RDF-3X*

RDF-3X (RDF Triple eXpress) is the experimental RDF storage and retrieval system [Neumann & Weikum, 2008]. RDF-3X can import N-Triples/Turtle RDF data. RDF-3X is an RDF storage system with advanced indexes and query optimization that eliminates the need of physical database design by the use of exhaustive indexes for all permutations of subject-property-object triples.

RDF-3X uses a potentially huge triples table, with own storage implementation underneath (as opposed to using an RDBMS). It overcomes the problem of expensive self-joins by creating a suitable set of indexes. All the triples are stored in a compressed clustered B+ tree. The triples are sorted lexicographically in the B+ tree. The triple store is compressed by replacing long string literals in the triples IDs using a mapping dictionary. The system supports both individual update operations and entire batches updates.

✓ *Hexastore*

Hexastore [Weiss et al, 2008] takes also a similar approach to YARS. The framework is based on the idea of main-memory indexing of RDF data in a multiple-index framework. The RDF data is indexed in six possible ways, one for each possible ordering of the three RDF elements by individual columns. The representation is based on any order of significance of RDF resources and properties and can be seen as a combination of vertical partitioning and multiple indexing approaches. Two vectors are associated with each RDF element, one for each of the others two RDF elements (e.g., [subject, property] and [subject, object]). Moreover, lists of the third RDF element are appended to the elements in these vectors. Hence, a sextuple indexing schema is created. As [Weiss et al, 2008] point out in, the values for O in PSO and SPO are the same. So in reality, even though six tables are created only *five copies* of the data are really computed, since the object columns are duplicated. To limit the amount of storage needed for the URIs, Hexastore uses the typical dictionary encoding of the URIs and the literals, i.e. every URI and literal is assigned *a unique numerical identifier*. Hexastore provides efficient single triple pattern lookups, and also allows fast merge-joins for any pair of two triple patterns. However, space requirement of Hexastore is five times the space required for storing statement in a triples table. Hexastore favors query performance over insertion time passing over applications that requires efficient statement insertion. Updates and insertions operations affect all six indices, hence can be slow. Hexastore does not provide inference support. [Weiss et al, 2008] proposed an on-disk index structure/storage layout so that Hexastore performance advantages can be preserved. Additionally to their experimental evaluations, they show empirically that, in the context of RDF storage, their *vector storage schema provides significantly lower data retrieval times compared to B trees*.

✓ *RDFCube*

The system RDFCube [Matono et al, 2007] is a three-dimensional hash index designed for RDFPeers [Cai & Frank, 2004] which is a distributed RDF repository that efficiently search RDF triples. Each triple is stored by specifying its subject, predicate, or object as a key. The RDFCube storage schema consists of set of cubes of the same size called cells. Each of these cells contains a bit called existence flag indicating the presence or absence of triples mapped into the cell. During the processing of a query, by checking the existence flags of cells into which candidate answer triples are mapped, it is possible to know the existence of the triples before actually accessing remote nodes where the candidate answer triples are stored. This information helps reducing the amount of data that is transferred among nodes when processing a join query since it is possible to narrow down the candidate triples by using AND operator between existence flags bits and transfer only the actual present candidate triples. However, using a DHT (Distributed Hash Table) for the indexation suffers from some problems such as freshness of data and security [Faye et al, 2012].

✓ *BitMat*

BitMat [Atre et al, 2009] is a main-memory based bit-matrix structure for representing a large set of RDF triples with the idea to make the representation compact. Each RDF triple is considered as a 3-dimensional entity which conceptually gives rise to a single universal table holding all RDF triples. This last can be horizontally partitioned into multiple fragments based on the usage requirements. BitMat can be viewed as a 3-dimensional bit-cube, in which each cell is a bit representing a unique triple and denoting the presence or absence of that triple. For representing the bit-cube in memory, it is flattened in a 2-dimensional bit matrix. There are six ways of flattening a bit-cube into a BitMat. Each structure contributes to more efficient particular set of single-join queries. To deal with the inherent sparsity of BitMat, this latter is maintained as an array of bit-rows, where each row is a collection of all the triples having the same subject. The underlying goal is to represent large RDF triple-sets with a compact in-memory representation and supporting a scalable multi-join query execution. These queries are processed using bitwise AND, OR operations on the BitMat rows and the resulting triples are returned as another BitMat. *BitMat is designed to be mainly a read-only RDF triple storing system.* Dynamic insertion or deletion of RDF triples is not supported at present.

✓ *Parliament*

Parliament [Kolas et al, 2009] is an open source triple store that is an improved version of DAMLDB. Parliament takes linked list style of approach. It uses BerkeleyDB for storing the URI values and then stores the triple of references in a linked list.

Parliament describes storage and indexing schema based on linked lists and memory-mapped files with a storage structure composed of three parts: the resource table, the statement table, and the resource dictionary.

The resource table is a single file of fixed-length records (sequentially numbered with numbers serving as ID of the corresponding resources), each of which representing a single resource or literal. This allows direct access to a record given its ID via simple array indexing. Each record has eight components:

- Three statement ID fields representing the first statements that contain this resource as a subject, predicate, and object, respectively;
- Three count fields containing the number of statements using this resource as a subject, predicate, and object, respectively;
- An offset used to retrieve the string representation of the resource;
- Bit-field flags encoding various attributes of the resource.

➢ *B3.3. Storage characteristics of outlined RDF triple stores*

Storage characteristics of outlined RDF triple stores are presented in Table 77.

Table 77. Storage characteristics of outlined RDF triple stores

Store	Triple Table	Property Table	Multi-indexing	DBMS	File	RAM	Update Support
3store [Harris & Gibbins, 2003]	√			√			
Jena [Jena2, 2012; Wilkinson et al, 2003]		√		√		√	√
RDFSuite [Alexaki et al, 2001]		√		√			√
Sesame [Sesame, 2012; Broekstra et al, 2002]		√		√	√	√	√
4store [Harris et al, 2009]		√		√			√
Oracle [Oracle, 2013]				√			
YARS [YARS, 2013]			√		√	√	√
Kowari [Wood et al, 2005]			√	√			
Virtuoso [Erling & Mikhailov, 2007]			√	√			√
RDF-3X [Neumann & Weikum, 2008]			√	√			√
Hexastore [Weiss et al, 2008]			√	√		√	
RDFCube [Matono et al, 2007]			√				
BitMat [Atre et al, 2009]			√			√	
Parliament [Kolas et al, 2009]			√		√		√

References

[3storeSF, 2013] 3store on Source forge, http://threestore.sourceforge.net/links.php (accessed: 23.03.2013).

[4store, 2013] 4store, http://4store.org/ (accessed: 23.03.2013).

[Aasman, 2011] Jans Aasman, "*Will Triple Stores Replace Relational Databases?*", Information Management and SourceMedia, Inc. APR 18, 2011 http://www.information-management.com/newsletters/database_metadata_unstructured_data_triple_store-10020158-1.html?zkPrintable=true (accessed: 11.01.2013).

[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.

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

[AHD, 2009] The American Heritage® "*Dictionary of the English Language*" Fourth Edition copyright© 2000 by Houghton Mifflin Company, Updated in 2009; Published by Houghton Mifflin Company. All rights reserved.

[AKT Project, 2013] Advanced Knowledge Technologies project, http://www.aktors.org/technologies/3store/ (accessed: 23.03.2013).

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

[Alexaki et al, 2001] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, Karsten Tolle "*The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases*", 2nd International Workshop on the Semantic Web (SemWeb'01), Hongkong, 2001.

[Alexaki et al, 2001a] Alexaki S., V. Christophides, G. Karvounarakis, D. Plexousakis, "*On Storing Voluminous RDF Descriptions: The case of Web Portal Catalogs*", In Proceedings of the 4th International Workshop on the Web and Databases (WebDB'01) - In conjunction with ACM SIGMOD/PODS, Santa Barbara, CA. May 24-25, 2001.

[Amann & Scholl, 1992] Amann B. and Scholl, M. "*Gram: A Graph Data Model and Query Language*", In European Conference on Hypertext Technology (ECHT), ACM, 1992, pp. 201–211.

[Amardeilh, 2006] Florence Amardeilh, "*OntoPop or how to annotate documents and populate ontologies from texts*", In Proceedings of the Workshop on Mastering the Gap: From

Information Extraction to Semantic Representation (ESWC-06), Budva, Montenegro, 2006 http://hal.archives-ouvertes.fr/docs/00/11/52/55/PDF/amardeilh_ESWC06.pdf (accessed: 31.07.2013)

[Andries et al, 1992] Andries M., Gemis M., Paredaens J., Thyssens I., and den Bussche, J. V. "*Concepts for Graph-Oriented Object Manipulation*", In Proc. of the 3rd Int. Conf. on Extending Database Technology (EDBT) LNCS, vol. 580, Springer, 1992, pp. 21–38.

[Angelov, 2012] St. Angelov. SA Dictionary http://www.thediction.com/ (accessed: 11.01.2013)

[Angles & Gutierrez, 2005] Angles, R. and Gutierrez, C, "*Querying RDF Data from a Graph Database Perspective*", In Proc. 2nd European Semantic Web Conference (ESWC), Number 3532 in LNCS. 2005, pp. 346–360.

[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

[Apollo, 2012] http://apollo.open.ac.uk/index.html (accessed: 25.08.2012)

[Arge, 2002] Arge, L., "*External memory data structures*", In: Handbook of Massive Datasets, Part 4, ch. 9. Kluwer Academic Publishers, 2002. pp. 313-357.

[Arpírez et al, 2001] Arpirez J., O. Corcho, M. Fernandez-Lopez, A. Gomez-Perez, "*WebODE: a Scalable Workbench for Ontological Engineering*", First International Conference on Knowledge Capture (KCAP'01). ACM Press (1-58113-380-4), pp. 6-13. October 2001.

[Arrouse, 1999] Arrouye Y. *The RealNames System - an International Human-Friendly Web Navigation System* http://www.unicode.org/iuc/iuc16/a333.html (accessed: 16.11.2012).

[Artemieva & Reshtanenko, 2008] Artemieva L. I. and N. V. Reshtanenko, "*Intellectuallized system based on multi-layer chemical ontologies*", (Артемьева Л. И. Н. В. Рештаненко, Интеллектуальная система, основанная на многоуровневой онтологии химии/Программные продукты и системы, 2008. № 1, pp. 84-87), http://www.swsys.ru/print/article_print.php?id=113, (accessed: 17.07.2013), (in Russian)

[Atre et al, 2009] Medha Atre, Jagannathan Srinivasan, James A. Hendler, "*BitMat: A Main Memory RDF Triple Store*", Technical Report, Tetherless World Constellation, Rensselaer Polytechnic Institute, Troy NY, USA, 2009.

[Auge, 1909] Claude Auge (ed.) „*Petit Larouse Illustré*", Librarie Larouse, Paris, 1909.

[Bachimon et al, 2002] Bachimont B., Isaac A. and Troncy R., "*Semantic Commitment for Designing Ontologies: A Proposal*", In Asuncion Gomez-Pérez and V. Richard Benjamins, editors, 13th International Conference on Knowledge Engineering and Knowledge Management, EKAW'2002, volume LNAI 2473, pp. 114-121, Sigüenza, Spain, October, 1-4, 2002. Springer Verlag. Paper, Slides

[Bachimont, 2000] Bachimont B., "*Engagement sémantique et engagement ontologique: conception et réalisation d'ontologies en ingénierie des connaissances*"; In "Ingénierie des connaissances Evolutions récentes et nouveaux défis", Jean Charlet, Manuel Zacklad, Gilles Kassel, Didier Bourigault; Eyrolles 2000, ISBN 2-212-09110-9

[Baidu, 2013] http://hi.baidu.com/huyangtree/item/5993ece1c094e1bc2f140b86 (accessed: 16.12.2013)

[Baker et al, 1998] Baker F. C., C. J. Fillmore, J. B. Lowe, "*The Berkeley FrameNet Project*", COLING–ACL, Montreal, Canada, 1998, pp. 86-90, http://acl.ldc.upenn.edu/C/C98/C98-1013.pdf (accessed: 21.07.2012)

[Bashmakov, 2005] Bashmakov A.I., "*Intellectual Information Technologies*" (Башмаков А. И. Интеллектуальные информационные технологии: Учеб. Пособие. М.: Изд.-во МГТУ им. Н. Э. Баумана, 2005, с.304) (in Russian)

[Bayer, 1971] Rudolf Bayer. „*Binary B-Trees for Virtual Memory*", ACM-SIGFIDET Workshop 1971, San Diego, California, Session 5B, pp. 219 - 235.

[Beale et al, 1996] Beale S., S. Nirenburg and K. Mahesh, „*Semantic Analysis in the Mikrokosmos Machine Translation Project*", 1996, pp. 1-11, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.9053 (accessed: 21.07.2012)

[Bechhofer et al, 2001] Bechhofer, S., Horrocks, I., Goble, C., Stevens, R., "*OILEd: a reasonable ontology editor for the semantic web*", In: KI2001, Joint German/Austrian conference on Artificial Intelligence, volume LNAI Vol. 2174, 2001, pp. 396-408, Vienna

[Becker, 2008] Christian Becker, "*RDF Store Benchmarks with Dbpedia*", Freie Universität Berlin, 2008, http://wifo5-03.informatik.uni-mannheim.de/benchmarks-200801/ (accessed: 05.04.2013)

[Beckett, 2001] Beckett David, "*The design and implementation of the Redland RDF Application Framework*", WWW10, 2001, Hong Kong, ACM 1-58113-348-0/01/0005, Redland - URL: http://www.redland.opensource.ac.uk/ (accessed: 15.10.2012).

[Beeri, 1988] Beeri, C., "*Data models and languages for databases*", In Proceedings of the 2nd International Conference on Database Theory (ICDT), LNCS, vol. 326, Springer, 1988, pp. 19–40.

[Belazzougui et al, 2009] Djamal Belazzougui, Fabiano C. Botelho, Martin Dietzfelbinger, "*Hash, Displace, and Compress*", In: Algorithms - ESA 2009 - 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009, Proceedings. Lecture Notes in Computer Science Volume 5757, Springer, 2009, pp 682-693. DOI 10.1007/978-3-642-04128-0_61 Print ISBN: 978-3-642-04127-3 Online ISBN: 978-3-642-04128-0. http://link.springer.com/chapter/10.1007%2F978-3-642-04128-0_61 (accessed: 20.07.2013).

[Berkeley DB, 2012] ORACLE Berkeley DB Products http://www.oracle.com/technetwork/products/berkeleydb/learnmore/berkeley-db-family-datasheet-132751.pdf;
http://www.oracle.com/technetwork/products/berkeleydb/overview/index.html (accessed: 15.10.2012)

[Bhadkamkar et al, 2009] Medha Bhadkamkar, Fernando Farfan, Vagelis Hristidis, and Raju Rangaswami, "*Storing Semi-structured Data on Disk Drives*", ACM Transactions on Storage, Vol. 5, No. 2, Article 6, Publication date: June 2009, pp. 6.1–6.35, ACM New York, NY, USA ISSN: 1553-3077 EISSN: 1553-3093 doi>10.1145/1534912.1534915 (accessed: 20.07.2013)

[BIG DATA INITIATIVE, 2012] Obama Administration univels "BIG DATA" INITIATIVE: Announces $200 Million in New R&D Investments, Office of Science and Technology

Policy | Executive Office of the President. March 29, 2012. http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_press_release_final_2.pdf (accessed 09.04.13)

[Big data, 2012] Fact Sheet: Big Data across the Federal Government, March 29, 2012, Office of Science and Technology Policy | Executive Office of the President, http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_fact_sheet_final.pdf (accessed 09.04.13)

[Bizer & Schultz, 2008] Christian Bizer, Andreas Schultz, "*Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints*", In: Proc. of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems (SSWS2008), http://www4.wiwiss.fu-berlin.de/bizer/pub/BizerSchulz-BerlinSPARQLBenchmark.pdf (accessed: 31.07.2013)

[Bizer & Schultz, 2009] Christian Bizer, Andreas Schultz, "*The Berlin SPARQL Benchmark*", In: International Journal on Semantic Web & Information Systems, Vol. 5, Issue 2, Pages 1-24, 2009, http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/Bizer-Schultz-Berlin-SPARQL-Benchmark-IJSWIS.pdf;
see also http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/ (accessed: 31.07.2013)

[Borrie, 2004] H. Borrie, "*The Firebird Book: a Reference for Database Developers*", Apress, 2004, ISBN: 1-59059-279-4

[Bourbaki, 1960] Bourbaki, N., "*Theorie des Ensembles*", Hermann, Paris, 1960, English version: Bourbaki, N. Theory of Sets, Volume package: Elements of Mathematics. Springer, 1st ed. 1968, 2nd printing 2004, ISBN 978-3-540-22525-6. 414 p.

[Bray et al, 1998] Bray, T., Paoli, J., and Sperberg-Mcqueen, C. M., "*Extensible Markup Language (XML) 1.0*", W3C Recommendation 10, (February), 1998. http://www.w3.org/TR/1998/REC-xml-19980210 (accessed: 20.07.2013).

[Briggs, 2012] Mario Briggs, "*DB2 NoSQL Graph Store*", What, Why & Overview, A presentation, Information Management software IBM, 2012, https://www.ibm.com/developerworks/mydeveloperworks/blogs/nlp/resource/DB2_NoSQL GraphStore.pdf?lang=en (accessed: 01.12.2012)

[Broekstra et al, 2002] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen, "*Sesame: A Generic Architecture for Storing and Querying RDF and RDF*", 2002.

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

[Brookshear, 2012] J. Glenn Brookshear, "*Computer science – an overview (11-th edition)*", Copyright© 2012, 2009, 2007, 2005, 2003, Pearson Education, Inc., publishing as Addison-Wesley, 2012 ISBN 10: 0-13-256903-5; ISBN 13: 978-0-13-256903-3. pp. 19-72

[Brusa et al, 2006] Graciela Brusa, Ma. Laura Caliusco, Omar Chiotti, "*A Process for Building a Domain Ontology: an Experience in Developing a Government Budgetary Ontology*", In: M. A. Orgun and T. Meyer, Eds. Proceedings of the second Australasian Workshop on Advances in ontologies (AOW 2006), Hobart, Australia; Conferences in Research and Practice in Information Technology, Vol. 72, pages 7-15; Australian Computer Society, Inc.

Darlinghurst, Australia, 2006. ISBN: 1-920-68253-8 http://dl.acm.org/citation.cfm?id=1273661 (accessed: 31.07.2013)

[BSBM DG, 2013] Data Generator and Test Driver, In: Berlin SPARQL Benchmark (BSBM) - Benchmark Rules, http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/BenchmarkRules/index.html#datagenerator (accessed: 31.07.2013)

[BSBM, 2012] Berlin SPARQL Benchmark, http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/ (accessed 09.04.13).

[BSBMv1, 2008] Berlin SPARQL Benchmark Results, V1, 2008, http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/V1/results/index.html (accessed: 31.07.2013)

[BSBMv2, 2008] Berlin SPARQL Benchmark Results, V2 2008, http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V2/index.html (accessed: 31.07.2013)

[BSBMv3, 2009] Berlin SPARQL Benchmark Results, V3, 2009, http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V3/index.html (accessed: 31.07.2013)

[BSBMv5, 2009] BSBM Results (V5) for Virtuoso, Jena TDB, BigOWLIM, 2009, http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V5/index.html (accessed: 31.07.2013)

[BSBMv6, 2011] Berlin SPARQL Benchmark Results, V6, 2011, http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V6/index.html (accessed: 31.07.2013)

[BTC, 2012] Billion Triple Challenge 2012 Dataset http://km.aifb.kit.edu/projects/btc-2012/ (accessed: 16.03.2013)

[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.

[Buneman, 1997] Buneman, P, "*Semistructured data*", In Proceedings of the 16th Symposium on Principles of Database Systems (PODS), ACM Press, 1997, pp. 117-121.

[Buneman, 2001] Peter Buneman, "*Semistructured Data*", Department of Computer and Information Science, University of Pennsylvania http://homepages.inf.ed.ac.uk/opb/papers/PODS1997a.pdf (accessed: 20.07.2013)

[Burgin & Gladun, 1989] Mark Burgin, Victor Gladun, "*Mathematical Foundations of Semantic Networks Theory*", In: LNCS No.: 364, Springer, 1989. pp. 117-135.

[Burgin, 2010] Mark Burgin, "*Theory of Information - Fundamentality, Diversity and Unification*", World Scientific Publishing Co. Pte. Ltd. Singapore, 2010, ISBN-13 978-981-283-548-2, pp. 672.

[Cai & Frank, 2004] Min Cai & Martin Frank, "*RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network*", WWW '04: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, 2004.

[Calvanese et al, 2007] Calvanese D., Cuenca B. Grau, Franconi E., "*Software Tools for Ontology*", Design and Maintenance FP6-7603 – Thinking ONtologiES (TONES) 2007, pp. 1–57, http://www.sts.tu-harburg.de/tech-reports/2007/TonesD15.pdf (accessed: 21.07.2012)

[Cantu, 2012] C.H. Cantu. Get to know Firebird in 2 minutes. March/2006 http://www.firebirdnews.org/imgs/firebird_in_2_minutes.pdf (accessed: 11.01.2013)

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

[Čech, 2012] Pavel Čech, "*Multi-dimensional Data Model of Textual Information*", In: V. M. Marques, A. Dmitriev (Eds.): Advances in Data Networks, Communications, Computers and Materials. WSEAS Press, ISBN: 978-1-61804-118-0, 2012; pp.197–202, http://www.wseas.org/wseas/cms.action?id=2514 (accessed: 20.07.2013)

[Chakrabarti, 2001] Chakrabarti, K., "*Managing Large Multidimensional Datasets Inside a Database System*", Phd Thesis, University of Illinois at Urbana-Champaign. Urbana, Illinois, 2001.

[Chavez et al, 2001] Chavez, E., Navarro, G., Baeza-Yates, & R., Marroquin, J., "*Searching in metric spaces*", ACM Computing Surveys, 33/3, 2001, pp.273-321

[Chen, 1976] Chen, P. P. S, "*The entity-relationship model—toward a unified view of data*", ACM Trans. Database Syst., 1, 1, 1976, pp. 9–36

[Chimaera, 2012] http://www-ksl.stanford.edu/software/chimaera/ (accessed: 09.08.2012).

[Chong et al, 2005] Eugene Inseok Chong, Souripriya Das, George Eadon, Jagannathan Srinivasan, "*An efficient SQL-based RDF querying scheme*", VLDB '05: Proceedings of the 31stinternational conference on Very large data bases, Trondheim, Norway, 2005.

[CODASYL, 1971] Codasyl Systems Committee, "*Feature Analysis of Generalized Data Base Management Systems*", Technical Report, May, 1971.

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

[Codd, 1980] Codd, E. F., "*Data Models in Database Management*", In Proc. of the 1980 Workshop on Data abstraction, Databases and Conceptual Modeling. ACM Press, 1980, pp. 112–114.

[Collins, 2003] "*Collins English Dictionary – Complete and Unabridged*", HarperCollins Publishers, 1991, 1994, 1998, 2000, 2003

[Connolly & Begg, 2002] T.M. Connolly, C.E.Begg, "*Database Systems*", A Practical Approach to Design, Implementation, and Management, Third Edition, Addison-Wesley Longman, Inc. – Pearson Education Ltd., 1995, 2002

[Corcho et al, 2005] Oscar Corcho, Mariano Fernández-López, Asunción Gómez-Pérez, Angel López-Cima, "*Building Legal Ontologies with METHONTOLOGY and WebODE*", In: Law and the Semantic Web, Lecture Notes in Computer Science Volume 3369, 2005, pp. 142-157, http://link.springer.com/chapter/10.1007%2F978-3-540-32253-5_9 (accessed: 31.07.2013)

[Costello & Jacobs, 2003] Roger L. Costello, David B. Jacobs, "*XML Design*", (A Gentle Transition from XML to RDF), The MITRE Corporation, 2003, http://www.csee.umbc.edu/courses/771/current/presentations/rdf.ppt
(accessed: 16.12.2013)

[CTS, 2012] Comparison of Triple Stores http://www.bioontology.org/wiki/images/6/6a/Triple_Stores.pdf (accessed: 11.01.2013).

[Daintith, 2004] John Daintith, "*Storage Schema*", A Dictionary of Computing, and 2004, Retrieved November 18, 2012, from Encyclopedia.com: http://www.encyclopedia.com/doc/1O11-storageschema.html (accessed: 26.11.2012)

[datahub_data0, 2012] BTC data set from Datahub, http://km.aifb.kit.edu/projects/btc-2012/datahub/data-0.nq.gz (accessed: 16.03.2013).

[Date, 1977] Date C. J., "*An Introduction to Database Systems*", Addison-Wesley Inc., 1975.

[Date, 2004] Date C. J., "*An Introduction to Database Systems*", 8th Edition, Pearson Education, Inc, ISBN 0-324-18956-6, 2004.

[DBpedia, 2007a] DBpedia dataset "homepages.nt" dated 2007-08-30, http://wifo5-03.informatik.uni-mannheim.de/benchmarks-200801/homepages-fixed.nt.gz (accessed: 31.07.2013)

[DBpedia, 2007b] DBpedia dataset "geocoordinates.nt" dated 2007-08-30, http://wifo5-03.informatik.uni-mannheim.de/benchmarks-200801/geocoordinates-fixed.nt.gz (accessed: 31.07.2013)

[DBpedia, 2007c] DBpedia dataset "infoboxes.nt" dated 2007-08-30, http://wifo5-03.informatik.uni-mannheim.de/benchmarks-200801/infoboxes-fixed.nt.gz (accessed: 31.07.2013)

[Dean & Ghemawat, 2008] J. Dean and S. Ghemawat, "M*apReduce: Simplified data processing on large clusters*," Commun ACM, 51(1), 2008, pp. 107-113.

[Demsar, 2006] Demsar J, "*Statistical comparisons of classifiers over multiple data sets*", J. Mach. Learn. Res., 7, 2006, pp. 1-30

[Deray & Verheyden, 2003] Deray T., P. Verheyden, "*Towards a Semantic Integration of Medical Relational Databases by Using Ontologies: A Case Study*", OTM Workshops, 2003, pp. 137-150.

[Dietzfelbinger et al, 1994] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert E. Tarjan, "*Dynamic Perfect Hashing: Upper and Lower Bounds*", SIAM J. Comput, 23, 4, 1994, ISSN: 0097-5397, pp. 738-761, http://portal.acm.org/citation.cfm?id=182370# (accessed: 20.07.2013).

[Dobrov et al, 2009] Dobrov B.V., Ivanov V.V., Lukashevich N.V., Soloviev V.D., "*Ontologies and Tesauruses: models, instruments, applications*", (Добров Б. В., Иванов В. В., Лукашевич Н. В., Соловьев В. Д. Онтологии и тезаурусы: модели, инструменты, приложения. Интернет-университет информационных технологий – ИНТУИТ.ру, БИНОМ. Лаборатория знаний, 2009, с. 176), (in Russian).

[DOE, 2012] http://www.eurecom.fr/~troncy/DOE (accessed: 15.10.2012).

[Dujmovi'c, 1996] Jozo Dujmovi'c, "*A Method for Evaluation and Selection of Complex Hardware and Software Systems*", In: CMG 96 Proceedings, 1996, pp. 368-378 http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.4388 (accessed: 31.07.2013)

[Dumbill, 2000] Dumbill E., "*Putting RDF to Work*", Article on XML.com, 09.08.2000. (http://www.xml.com/pub/a/2000/08/09/rdfdb/); rdfDB URL: http://guha.com/rdfdb/ (accessed: 15.10.2012).

[ebxml, 2012] http://www.ebxml.org (accessed: 25.05.2012).

[Erling & Mikhailov, 2007] Orri Erling, Ivan Mikhailov, "*RDF Support in the Virtuoso DBMS*", Conference on Social Semantic Web, 2007.

[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)

[Farquhar et al, 1996] Farquhar, A., Fikes, R., Rice, J., “*The Ontolingua server: a tool for collaborative ontology construction*”, In: Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1996.

[Faye et al, 2012] David C. Faye, Olivier Cure, Guillaume Blin, “*A survey of RDF storage approaches*”, Received, December 12, 2011, Accepted, February 7, 2012, ARIMA Journal, vol. 15, 2012, pp. 11-35.

[Fellbaum et al, 1998] Fellbaum, Christiane, ed., “*WordNet: An Electronic Lexical Database*”, MIT Press, Cambridge, MA, 1998, pp. 422

[Fellbaum, 1998] Fellbaum Christiane (ed.) WordNet, “*An Electronic Lexical Database*”, ISBN: 978026206197, MA: MIT Press, 1998, pp. 422

[Fernández et al, 1997] Mariano Fernández, Asunción Gómez-Pérez, Natalia Juristo, “*METHONTOLOGY: From Ontological Art towards Ontological Engineering*”, Spring Symposium on Ontological Engineering of AAAI; Stanford University, California, AAAI TR SS-97-06, 1997, pp 33–40. http://oa.upm.es/5484/1/METHONTOLOGY_.pdf (accessed: 31.07.2013)

[Filatov et al, 2007] Filatov V.A., Shcherbak S.S, Hairova A.A., “*Development effective tools for creating and processing of ontological knowledge*”, (Филатов В. А., Щербак С. С., Хайрова А. А. Разработка высокоэффективных средств создания и обработки онтологических баз знаний/ Системи обробки інформації, випуск 8 (66), 2007, pp. 120-124), www.nbuv.gov.ua/portal/natural/soi/2007_8/Filatov.pdf (accessed:21.07.2012), (in Russian).

[Fillmore, 1976] Fillmore C. J., “*Frame semantics and the nature of language*”, Annals of the New York Academy of Sciences, Volume 280, 1976, pp. 20–32.

[Firebird, 2013] Firebird Project Firebird Foundation Incorporated. Copyright© 2000-2013, http://www.firebirdsql.org/en/about-firebird/ (accessed: 16.03.2013).

[Fisher, 1973] R. A. Fisher, “*Statistical methods and scientific inference*”, (3rd edition) Hafner Press, New York, 1973, ISBN 978-002-844740-7.

[Fletcher & Beck, 2009] George H. L. Fletcher, Peter W. Beck, “*Scalable indexing of RDF graphs for efficient joins processing*”, CIKM ’09: Proceeding of the 18th ACM conference on Information and knowledge management, New York, NY, USA, 2009.

[FrameNet, 2012] FrameNet II FrameGrapher. http://framenet.icsi.berkeley.edu/FrameGrapher (accessed: 21.07.2012)

[Franz Inc., 2013] Semantic Web Technologies http://www.franz.com/ (accessed: 16.05.2013).

[Frege, 1980] Frege G., “*An extract from an undated letter*”, published in Frege's Philosophical and Mathematical Correspondence (ed.) Gottfried Gabriel, Hans Hermes. Friedrich Kanbartel. Christian Thiel, and Albert Veraart, Abridged for the English (edn.), by Brian MeGuinness, and Trans. Hans Kaal (Oxford: Blackwell. 1980), http://mind.ucsd.edu/syllabi/00-01/phil235/a_readings/frege_jourdain.html (accessed: 15.11.2012).

[Friedman, 1940] Friedman, M., "*A comparison of alternative tests of significance for the problem of m rankings*", Annals of Mathematical Statistics, Vol. 11, 1940, pp.86-92

[Gabel et al, 2004] Gabel T, Sure Y, Voelker J., "*KAON – An overview*", Insititute AIFB, University of Karlsruhe, 2004, http://www.aifb.kit.edu/web/KAON/en (accessed: 11.08.2012).

[Gaede & Günther, 1998] Gaede V. and Günther O, "*Multidimensional access methods*", ACM Computing Surveys, 30(2), 1998

[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)

[Gandon, 2002] Gandon F., "*Ontology Engineering: a survey and a return on experience*", ACACIA Team, Thème 3: Interaction homme-machine, images données, connaissances, INRIA: Rapport de recherche n° 4396 - March 2002, pp. 181.

[Gavrilova, 2001] Gavrilova T.A., "*Knowledge bases of intellectual systems*", (Гаврилова Т. А. Базы знаний интеллектуальных систем/Т. А. Гаврилова, В. Ф. Хорошев-ский. СПб.: Питер, 2001. С. 384), (in Russian)

[Gemis & Paredaens, 1993] Gemis, M. and Paredaens, J., "*An Object-Oriented Pattern Matching Language*", In Proc. of the First JSSST Int. Symposium on Object Technologies for Advanced Software, Springer- Verlag, 1993, pp. 339–355.

[Giuglea & Moschitti, 2004] Giuglea A, A. Moschitti, "*Knowledge Discovering using FrameNet*", VerbNet and PropBank, 2004, pp. 6, http://olp.dfki.de/pkdd04/giuglea-final.pdf (accessed: 21.07.2012)

[Gladun, 2003] Gladun, V. P, "*Intelligent systems memory structuring*", International Journal Information Theories and Applications, 10(1), 2003, pp. 10–14.

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

[Graves & Gutierrez, 2006] Graves Alvaro and Caludio Gutierrez, "*Data representations for WordNet: A case for RDF*", In Petr Sojka, Key-Sun Choi, Christiane Fellbaum, and Piek Vossen, editors, GWC 2006 – Proceedings of the 3rd International WordNet Conference, South Jeju Island, Korea, January 22-26, 2006, pp. 165–169.

[Graves et al, 1994] Graves, M., Bergeman, E. R., and Lawrence, C. B., "*Querying a Genome Database using Graphs*", In In Proc. of the 3th Int. Conf. on Bioinformatics and Genome Research, 1994.

[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.

[Graves, 1993] Graves, M, "*Theories and Tools for Designing Application-Specific Knowledge Base Data Models*", PhD thesis - University of Michigan, 1993

[Greenwood, 2012] Eric Greenwood, "*Storage Models and their Most Glaring Vulnerabilities*", Tweak and Trick, http://www.tweakandtrick.com/2011/08/data-storage-model-risk.html (accessed: 26.11.2012)

[Grolinger et al, 2014] K. Grolinger, M. Hayes, W. Higashino, A. L'Heureux, D. S. Allison, M. A. M. Capretz, "*Challenges for MapReduce in Big Data*", Proc. of the IEEE 10th 2014 World Congress on Services (SERVICES 2014), Alaska, USA, June 27-July 2, 2014

[Gruber, 1993] Gruber R. T., "*A translation approach to portable ontologies*", Knowledge Acquisiton 5(2), 1993, pp. 199-220, http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html (accessed: 15.08.2012)

[Gruber, 1993a] Gruber R. T., "*Toward principles for the design of ontologies used for knowledge sharing*", Presented at the Padua workshop on Formal Ontology, March 1993, later published in International Journal of Human-Computer Studies, Vol. 43, Issues 4-5, November 1995, pp. 907-928, Available online.

[Guarino & Giaretta, 1995] Guarino N., Giaretta P., "*Ontologies and Knowledge Bases: Towards a Terminological Clarification*", In N. J. I. Mars (ed.), Towards Very Large Knowledge Bases, IOS Press 1995.

[Guarino, 1998] Guarino N., "*Formal Ontology and Information Systems*", N. Guarino In N. Guarino (ed.) Formal Ontology and Information Systems/FOIS'98, 6–8 June 1998, Trento, Italy: IOS Press, Amsterdam, 1998, pp. 3–15.

[Guha, 2013] R. V. Guha, "*rdfDB: An RDF Database*", http://www.guha.com/rdfdb/ (accessed: 16.03.2013).

[Guinn & Aasman, 2010] Guinn B., J. Aasman, "*Semantic Real Time Intelligent Decision Automation*", STIDS 2010 Proceedings, pp. 125-128. http://ceur-ws.org/Vol-713/STIDS_P1_GuinnAasman.pdf (accessed: 15.08.2012)

[Gunther, 1998] Gunther O., "*Environment Information Systems*", Springer, Berlin, New Work, 1998, pp. 244.

[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.

[Hadoop, 2014] Apache Hadoop, http://hadoop.apache.org . (accessed 22.12.14)

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

[Harris et al, 2009] Steve Harris, Nick Lamb, and Nigel Shadbolt, "*4store: The design and implementation of a clustered RDF store*", In SSWS2009: Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems, 2009.

[Hayes & Gutierrez, 2004] Hayes, J. and Gutierrez, C., "*Bipartite Graphs as Intermediate Model for RDF*", in: Proc. of the 3th Int. Semantic Web Conference (ISWC), Number 3298 in LNCS, Springer-Verlag, 2004, pp. 47–61.

[Hayes et al, 2005] Hayes, P., Eskridge, C. T., Reichherzer, T., Saavedra, R., Mehrotra, M., Bobrovnikoff, D., "*COE: Tools for Collaborative Ontology Development and Reuse*", In: Knowledge Capture Conference (K-CAP), 2005.

[Hayes, 2004] Hayes P., "*RDF Semantics*", W3C Recommendation, ed., 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: 28.08.2012)

[Heinz et al, 2002] Steffen Heinz, Justin Zobel, Hugh E. Williams, "*Burst Tries: A Fast, Efficient Data Structure for String Keys*", ACM Transactions on Information Systems (TOIS), Volume 20, Issue 2, April 2002, pp. 192 – 223, ACM New York, NY, USA, doi>10.1145/506309.506312, http://dl.acm.org/citation.cfm?id=506312 (accessed: 20.07.2013)

[Hertel et al, 2009] Hertel A., J. Broekstra, and H. 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.

[Hidders & Paredaens, 1993] Hidders, J. and Paredaens, J., "*GOAL A Graph-Based Object and Association Language*", Advances in Database Systems: Implementations and Applications, CISM, 1993, pp. 247–265.

[Hidders, 2001] Hidders, J., "*A Graph-based Update Language for Object-Oriented Data Models*", PhD thesis in Technische Universiteit, Eindhoven, 2001

[Hidders, 2002] Hidders, J., "*Typing Graph-Manipulation Operations*", In: Proc. of the 9th Int. Conf. on Database Theory (ICDT), Springer-Verlag, 2002, pp. 394–409.

[HORIZON 2020, 2013] HORIZON 2020 – WORK PROGRAMME 2014-2015. http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/main/h2020-wp1415-leit-ict_en.pdf (accessed: 29.12.2013)

[i7 950, 2009] Intel i7 950 @ 3.07GHz (quadcore); CPU Launched: 2009; http://www.cpubenchmark.net/cpu.php?cpu=Intel+Core+i7+950+%40+3.07GHz&id=837 (accessed: 31.07.2013)

[IBL, 2012] Internet Business Logic http://www.semanticweb.org/wiki/Internet_Business_Logic (accessed: 21.07.2012)

[IBM, 1965-68] IBM System/360 (1965-68): Disk Operating System, Data Management Concepts. IBM System Reference Library, IBM Corp. 1965, Major Revision, Feb.1968.

[ibphoenix, 2012] Latest Firebird and Interbase Related News and Information http://www.ibphoenix.com (accessed: 11.01.2013)

[ICOM, 2012] http://www.inf.unibz.it/~franconi/icom (accessed: 21.08.2012)

[Inseok et al, 2005] Eugene Inseok Chong, Souripriya Das, George Eadon, Jagannathan Srinivasan, "*An efficient SQL-based RDF querying scheme*", VLDB '05: Proceedings of the 31stinternational conference on Very large data bases, Trondheim, Norway, 2005.

[InterBase, 2012] Borland InterBase http://www.ibprovider.com/eng/documentation/interbase.html (accessed: 11.01.2013)

[ISI, 2012] http://www.isi.edu (accessed: 21.07.2012)

[ISP2.0, 2012] Intellidimension Inc., Semantics Platform 2.0. http://www.intellidimension.com/products/semantics-platform/ (accessed: 15.10.2012)

[Ivanova et al, 2012a] Krassimira Ivanova, Vitalii Velychko, Krassimir 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, 2012b] Krassimira Ivanova, Vitalii Velychko, Krassimir Markov, "*Storing RDF Graphs using NL-addressing*", In: G. Setlak, M. Alexandrov, K. Markov (ed.), Artificial Intelligence Methods and Techniques for Business and Engineering Applications. ITHEA® 2012, Rzeszow, Poland; Sofia, Bulgaria, ISBN: 978-954-16-0057-3 (printed), ISBN: 978-954-16-0058-0 (online), pp. 84 – 98.

[Ivanova et al, 2013a] Krassimira B. Ivanova, Koen Vanhoof, Krassimir Markov, Vitalii Velychko, "*Introduction to the Natural Language Addressing*", International Journal "Information Technologies & Knowledge" Vol.7, Number 2, 2013, ISSN 1313-0455 (printed), 1313-048X (online), pp. 139–146.

[Ivanova et al, 2013b] Krassimira B. Ivanova, Koen Vanhoof, Krassimir Markov, Vitalii Velychko, "*Introduction to Storing Graphs by NL-Addressing*", International Journal "Information Theories and Applications", Vol. 20, Number 3, 2013, ISSN 1310-0513 (printed), 1313-0463 (online), pp. 263 – 284.

[Ivanova et al, 2013c] Krassimira B. Ivanova, Koen Vanhoof, Krassimir Markov, Vitalii Velychko, "*Storing Dictionaries and Thesauruses Using NL-Addressing*", International Journal "Information Models and Analyses" Vol.2, Number 3, 2013, ISSN 1314-6416 (printed), 1314-6432(online), pp. 239 - 251.

[Ivanova et al, 2013d] Krassimira B. Ivanova, Koen Vanhoof, Krassimir Markov, Vitalii Velychko, "*The Natural Language Addressing Approach*", International Scientific Conference "Modern Informatics: Problems, Achievements, and Prospects of Development", devoted to the 90th anniversary of academician V. M. Glushkov. Kiev, Ukraine, 2013, ISBN 978-966-02-6928-6, pp. 214 - 215.

[Ivanova et al, 2013e] Krassimira B. Ivanova, Koen Vanhoof, Krassimir Markov, Vitalii Velychko, "*Storing Ontologies by NL-Addressing*", IVth All–Russian Conference "Knowledge-Ontology-Theory" (KONT-13), Novosibirsk, Russia, 2013, ISSN 0568-661X, pp. 175 - 184.

[Ivanova, 2013] Krassimira Ivanova, "*Informational and Information models*", In Proceedings of 3rd International conference "Knowledge Management and Competitive Intelligence" in the frame of 17th International Forum of Young Scientists "Radio Electronics and Youth in the XXI Century", Kharkov National University of Radio Electronics (KNURE), Kharkov, Ukraine, Vol.9, 2013, pp 6-7.

[Janik & Kochut, 2005] Maciej Janik and Krys Kochut, "*BRAHMS: A WorkBench RDF Store and High Performance Memory System for Semantic Association Discovery*", In Fourth International Semantic Web Conference, 2005.

[Jena, 2013] Apache Jena, http://jena.apache.org/about_jena/about.html (accessed: 23.03.2013)

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

[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.

[Kalfoglou & Schorlemmer, 2003] Yannis Kalfoglou, Marco Schorlemmer, "*Ontology mapping: the state of the art*", The Knowledge Engineering Review, Vol. 18:1, pp. 1–31, Cambridge University Press, United Kingdom, USA, 2003. ISSN = 0269-8889, DOI: 10.1017/S0269888903000651 http://dl.acm.org/citation.cfm?id=975028 (accessed: 31.07.2013)

[Kalyanpur et al, 2005] Kalyanpur, A., Parsia, B., Hendler, J., "*A Tool for Working with Web Ontologies*", in: Proceedings of the International Journal on Semantic Web and Information Systems, Vol.1, No.1, Jan-Mar (2005)

[Kaon, 2012] http://www.aifb.kit.edu/web/KAON/en (accessed: 22.08.2012).

[Kerschberg et al, 1976] Kerschberg, L., Klug, A. C., and Tsichritzis, D, "*A Taxonomy of Data Models*", In: Proc. of Systems for Large Data Bases (VLDB), North Holland and IFIP, 1976, pp. 43–64.

[Kim, 1990] Kim, W, "*Object-oriented databases: definition and research directions*", IEEE Trans, Knowl. Data Eng. 2, 3, 1990, pp. 327–341.

[Kingsbury & Palmer, 2003] P. Kingsbury, M. Palmer, "*PropBank: the Next Level of the TreeBank*", University of Pennsylvania, Department of Computer and Information Science, 2003, pp. 12, http://w3.msi.vxu.se/~rics/TLT2003/doc/kingsbury_palmer.pdf (accessed: 21.07.2012)

[Klyne & Carroll, 2004] G. Klyne and J. 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: 22.08.2012)

[Knuth, 1997] Donald Knuth, "*The art of computer programming*", Vol. 1: Fundamental Algorithms, Third Edition, Addison-Wesley, 1997, ISBN 0-201-89683-4, Section 2.3, especially subsections 2.3.1–2.3.2, pp. 318–348.

[Knuth, 1998] Knuth, Donald E., "*The Art of Computer Programming*", Vol. 2: Seminumerical Algorithms (3rd edition ed.), Addison Wesley, ISBN 0-201-89684-2, 1998

[Kolas et al, 2009] Dave Kolas, Ian Emmons, Mike Dean, "*E_cient Linked-List RDF Indexing*", in Parliament. http://parliament.semwebcentral.org/ISWC2009ParliamentPaper.pdf. See also: http://parliament.semwebcentral.org/ (accessed: 23.03.2013)

[Kolosovskiy, 2009] Kolosovskiy M., "*Simple implementation of deletion from open-address hash table*", Cornell University Library, ArXiv e-prints, 2009, http://adsabs.harvard.edu/abs/2009arXiv0909.2547K (accessed: 20.07.2013)

[Kowari, 2004] Kowari Metastore, http://kowari.sourceforge.net/oldsite/1061.htm#o1068 (accessed: 23.03.2013)

[Kumar & Crowley, 2005] Sailesh Kumar, Patrick Crowley, "*Segmented Hash: An Efficient Hash Table Implementation for High Performance Networking Subsystems*", In: ANCS '05

Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems. ACM New York, NY, USA ©2005, ISBN:1-59593-082-5 doi: 10.1145/1095890.1095904, pp 91-103

[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.

[LDIF Benchmarks, 2013] LDIF - Benchmark Results, http://ldif.wbsg.de/benchmark.html (accessed: 31.07.2013)

[LDIF, 2013] LDIF – Linked Data Integration Framework, http://ldif.wbsg.de/ (accessed 09.04.13).

[Lee, 1999] Y. Tina Lee. Information Modeling: From Design to Implementation. Proceedings of the Second World Manufacturing Congress: Manifacturing Systems, Technology, Management. ICSC 1999, ISBN: 9783906454191, pp 315—321.

[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.

[Liang, 1983] Franklin Mark Liang, "*Word Hy-phen-a-tion by Com-put-er*", PhD thesis, Department of Computer Science, Stanford University, Stanford, California 94305, Report No STAN-CS-83-977, August 1983, http://www.tug.org/docs/liang/liang-thesis.pdf (accessed: 20.07.2013).

[Liebig & Noppens, 2003] Liebig, T., Noppens, O., "*OntoTrack: Fast Browsing and Easy Editing of Large Ontologies*", In: Proceedings of the 2nd International Workshop on Evaluation of Ontologybased Tools (EON-2003) Sanibel Island, Florida, USA (2003), pp. 47-56

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

[Lungen et al, 2007] Lungen Harald, Claudia Kunze, Lothar Lemnitzer, and Angelika Storrer, "*Towards an integrated OWL model for domain-specific and general language WordNets*", In Attila Tanacs, Dora Csendes, Veronika Vincze, Christiane Fellbaum, and Piek Vossen, editors, GWC 2008 – Proceedings of the 4th Global WordNet Conference, 2007, pp. 281-296, Szeged, Hungary, January 22-25, 2008.

[Macris, 2004] Macris A., "*CULTOS: Cultural Units of Learning Tools and Services*", 3rd Hellenic Conference on Artificial Intelligence, Samos, Greece, Proceedings, 5-8 May 2004, pp. 248-259

[Magkanaraki et al, 2002] Magkanaraki A., G. Karvounarakis, Ta Tuan Anh, V. Christophides, D. Plexousakis, "*Ontology Storage And Querying, Technical Report*", No 308, Foundation for Research and Technology, Hellas Institute of Computer Science, Information Systems Laboratory, April 2002. http://xml.coverpages.org/MagkanarakiOnt.pdf (accessed: 15.10.2012)

[Mainguenaud, 1992] Mainguenaud, M., "*Simatic XT: A Data Model to Deal with Multi-scaled Networks*", Computer, Environment and Urban Systems 16, 1992, pp. 281–288

[Mano, 1993] M. Morris Mano, "*Computer System Architecture*", Third edition. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, ISBN 0-13-175563-3, 1993, 530 p.

[Markov et al, 1990] K. Markov, T. Todorov, V. Nikolov, "*Multidomain Access Method for the IBM PC*", Research in Informatics, Vol. 3, Academie-Verlag Berlin, 1990, pp. 218-230.

[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 et al, 2013] Markov, Krassimir, Koen Vanhoof, Iliya Mitov, Benoit Depaire, Krassimira Ivanova, Vitalii Velychko and Victor Gladun, "*Intelligent Data Processing Based on Multi-Dimensional Numbered Memory Structures*", Diagnostic Test Approaches to Machine Learning and Commonsense Reasoning Systems, IGI Global, 2013, pp. 156-184, doi:10.4018/978-1-4666-1900-5.ch007, ISBN: 978 1-4666-1900-5, EISBN: 978-1-4666-1901-2

Reprinted in: Markov, Krassimir, Koen Vanhoof, Iliya Mitov, Benoit Depaire, Krassimira Ivanova, Vitalii Velychko and Victor Gladun, "*Intelligent Data Processing Based on Multi-Dimensional Numbered Memory Structures*", Data Mining: Concepts, Methodologies, Tools, and Applications, IGI Global, 2013, pp. 445-473, doi:10.4018/978-1-4666-2455-9.ch022, ISBN13: 978-1-4666-2455-9, EISBN13: 978-1-4666-2456-6

[Markov et al, 2014] Kr. Markov, Kr. Ivanova, K. Vanhoof, B. Depaire, V. Velychko, J. Castellanos, L. Aslanyan, St. Karastanev, "*Storing Big Data Using Natural Language Addressing*", In: N. Lyutov (ed.), Int. Sc. Conference "Informatics in the Scientific Knowledge", VFU, Varna, Bulgaria, 2014, ISSN: 1313-4345, pp. 147-164.

[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

[Markov, 2005] Markov, K., "*Building data warehouses using numbered multidimensional information spaces*", International Journal of Information Theories and Applications, 12(2), 2005, pp. 193–199.

[Markov, 2006] Kr. Markov, "*Multidimensional Context-free Access Method*", PhD Thesis, Intitute of Mathematics and Informatics, Sofia, Bulgaria. 2006. (in Bulgarian)

[Martin, 1975] J. Martin, „*Computer Data-Base Organization*", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[Masolo et al, 2003] Masolo C., Borgo S., Gangemi A., Guarino N., Oltramari A., "*WonderWeb Deliverable D18: Ontology Library (final)*", Laboratory for Applied Ontology – ISTC–CNR, 2003, pp. 349, http://www.loa-cnr.it/Papers/D18.pdf (accessed: 21.07.2012)

[Matono et al, 2007] Akiyoshi Matono, Said Mirza Pahlevi, Isao Kojima, "*RDFCube: A P2P-Based Three- Dimensional Index for Structural Joins on Distributed Triple Stores*", SpringerLink – Book Chapter Databases, Information Systems, and Peer-to-Peer Computing, 2007.

[McBride, 2001] McBride B., "*Jena: Implementing the RDF Model and Syntax Specification*", In: Steffen Staab et al (eds.), Proc. of the Second International Workshop on the Semantic Web-SemWeb2001, May 2001, http://ceur-ws.org/Vol-40/mcbride.pdf, Jena URL:http://www.hpl.hp.com/semweb/jena-top.html (accessed: 15.10.2012)

[McGlothlin & Khan, 2009] James P. McGlothlin, Latifur R. Khan "RDFJoin: A Scalable of Data Model for Persistence and Efficient Querying of RDF Datasets", UTDCS-08-09, 2009.

[McGlothlin & Khan, 2009a] James P. McGlothlin, Latifur R. Khan, "*RDFKB: efficient support for RDF inference queries and knowledge management*", IDEAS '09: Proceedings of the 2009 International Database Engineering, Applications Symposium, Cetraro - Calabria, Italy, 2009.

[Mell & Grance, 2011] Peter Mell, Timothy Grance, "*The NIST Definition of Cloud Computing*", NIST Special Publication 800-145, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, September 2011.

[Mendelzon et al, 2001] Alberto Mendelzon, Thomas Schwentick, Dan Suciu, "*Foundations of Semistructured Data*", 2001, http://www.dagstuhl.de/Reports/01/01361.pdf (accessed: 20.07.2013).

[Mikr, 2012] Mikrokosmos http://www.ilc.cnr.it/EAGLES96/rep2/node23.html (accessed: 21.07.2012).

[Miller, 1995] Miller G. A., "*WordNet: a lexical database for English*", G. A. Miller – Communications of the ACM 38: 11, 1995, pp. 39–41

[Minack, 2010] Enrico Minack, "RDF2RDF converter", http://www.l3s.de/~minack/rdf2rdf/ 2010, (accessed: 31.07.2013).

[Mitov, 2011] Iliya Mitov, "*Class Association RuleMining UsingMulti-Dimensional Numbered Information Spaces*", PhD Thesis, Hasselt University, Belgium, 2011.

[Moënne-Loccoz, 2005] Moënne-Loccoz, N., "*High-dimensional access methods for efficient similarity queries*", Technical Report N: 0505, University of Geneva, Computer Vision and Multimedia Laboratory, 2005.

[Mokbel et al, 2003] Mokbel, M., Ghanem, T., & Aref, "*Spatio-temporal access methods*", A Quarterly Bulletin of the Computer Society of the IEEE Technical Committee on Data Engineering, 26(2), 2003, pp. 40–49.

[Morin, 2005] Pat Morin, “*Hash tables*”, Chapter 9, of “Handbook of data structures and applications” /edited by Dinesh P. Mehta and Sartaj Sahni, Chapman & Hall/CRC computer & information science, 2005, 1321 pages, ISBN 1-58488-435-5.

[Muys, 2007] Andrae Muys, “*Building an Enterprise Scale Database for RDF Data*”, Seminar, Netymon, 2007.

[Naur, 1963] Peter Naur (ed.), “*Revised Report on the Algorithmic Language Algol 60*”, Communications of the ACM, Vol. 6, Number 1, Jan. 1963.

[Navathe, 1992] Navathe, S. B., “*Evolution of Data Modeling for Databases*”, Communications of the ACM 35, 9, 1992, pp. 112–123.

[Neave & Worthington, 1992] Neave, H., Worthington, P., “*Distribution Free Tests*”, Routledge, 1992

[Nemenyi, 1963] Peter Nemenyi, “*Distribution-free multiple comparisons Unpublished*”, PhD thesis; Princeton University Princeton, NJ, 1963

[Neumann & Weikum, 2008] Thomas Neumann, Gerhard Weikum, “*RDF-3X: a RISC-style Engine for RDF*”, JDMR (formely Proc. VLDB) 2008, Auckland, New Zealand, http://www.mpi-inf.mpg.de/~neumann/rdf3x/, https://domino.mpi-inf.mpg.de/intranet/ag5/ag5publ.nsf/AuthorEditorIndividualView/ad3dbafa6fb90dd2c1257593002ff3df/$FILE/rdf3x.pdf?OpenElement (accessed: 23.03.2013).

[Nevzorova & Nevzorov, 2009] Nevzorova O., Nevzorov V., “*Ontological analysis of the domain: Automated methods of term extraction in “OntoIntegrator” system*”, in “Modelling methods” symposium, Kazan, 2009, pp.196-208, (in Russian)

[Nevzorova & Nevzorov, 2011] O. Nevzorova, V. Nevzorov, “*Terminological annotation of the document in a retrieval context on the basis of technologies of system “ONTOINTEGRATOR"*”, International Journal "Information Technologies & Knowledge" Vol. 5, Number 2, 2011. pp. 110-118

[Nevzorova et al, 2004] Nevzorova O.A., Nevzorov V.N., “*The Analysis of the Structural Features of the Ontology by the Development support system “OntoEditor”*”, (Система визуального проектирования онтологий "OntoEditor": функциональные возможности и применение //IX национальная конференция по искусственному интеллекту с международным участием) КII-2004. Т. 3. pp. 176-183, (In Russian)

[Nevzorova et al, 2007] Nevzorova O.A., Nevzorov V.N., Zinkina U.V., Pyatkin N.B., “*Integral Technology of Homonymy Disambiguation in the Text Mining System "LoTA"*”, Int. Conf. “Dialog 2007”, Moscow: ИПИ РАН, 2007, pp. 422–427, http://www.dialog-21.ru/digests/dialog2007/materials/html/64.htm, (accessed: 16.03.2013), (in Russian)

[Noy & Musen, 1999] Noy N, M. Musen, “*SMART: Automated Support for Ontology Merging and Alignment*”, Stanford Medical Informatics, Stanford Univ, 1999, pp. 24

[Noy & Musen, 2002] Noy, N. F., Musen, M. A., “*Evaluating ontology-mapping tools: Requirements and experience*”, In: Proceeding of OntoWeb-SIG3 Workshop, 2002, pp. 1-14

[N-Quads, 2013] N-Quads: Extending N-Triples with Context http://sw.deri.org/2008/07/n-quads/ (accessed: 16.03.2013).

[NRC, 2013] National Research Council, "*The Mathematical Sciences in 2025*", Washington, DC: The National Academies Press, USA, 2013. ISBN-13: 978-0-309-28457-8. http://www.nap.edu/catalog.php?record_id=15269 (accessed 09.04.13).

[Obitko, 2007] Obitko M., "*Ontologies and Semantic Web*", 2007 http://www.obitko.com/tutorials/ontologies-semantic-web/operations-on-ontologies.html (accessed: 09.08.2012)

[Obitko, 2007a] Obitko M., "*RDF Query Language SPARQL*", 2007 http://www.obitko.com/tutorials/ontologies-semantic-web/rdf-query-language-sparql.html (accessed: 06.04.2013).

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

[Olson et al, 1999] Michael A. Olson, Keith Bostic, and Margo Seltzer, "*Berkeley DB*", Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference; Monterey, California, USA. USENIX Association, 1999

[OntoLex, 2012] Alexa Melina, Bernd Kreissig, Martina Liepert, Klaus Reichenberger, Lothar Rostek, Karin Rautmann, Werner Scholze-Stubenrecht, Sabine Stoye, "*The Duden Ontology: An Integrated Representation of Lexical and Ontological Information*", http://www.bultreebank.org/OntoLex02/OntoLex02Paper01.pdf (accessed: 15.10.2012)

[Ontopia, 2012] "*The Ontopia Knowledge Suite: An introduction*", White Paper (V. 1.3), 2002 http://www.regnet.org/members/demo/ontopia/doc/misc/atlas-tech.html; URL: http://www.ontopia.net/solutions/products.html (accessed: 15.10.2012)

[ontoprise, 2012] http://www.ontoprise.de/products/index_html_en; http://help.semafora-systems.com/ (accessed: 15.10.2012)

[OntoTools, 2012] A List of Ontology Engineering Tools (Ontology Editors) - http://www.hozo.jp/OntoTools/; Mizoguchi Lab., The Institute of Scientific and Industrial Research, Osaka University: http://www.ei.sanken.osaka-u.ac.jp/ (accessed: 22.07.2012)

[Ooi et al, 1993] Ooi B., Sacks-Davis R., Han J, "*Indexing in spatial databases*", Technical Report, 1993

[OpenCyc, 2012] OpenCyc Documentation http://www.opencyc.org/doc (accessed: 21.07.2012)

[Oracle, 2013] ORACLE, http://www.oracle.com/technetwork/ (accessed: 23.03.2013).

[oracledb, 2012] http://www.oracle.com/technetwork/database/options/semantic-tech/index.html (accessed: 11.08.2012)

[OSTI, 2009] Oracle Semantic Technologies Inference Best Practices with RDFS/OWL 2009 http://download.oracle.com//otndocs/tech/semantic_web/pdf/semantic_infer_bestprac_wp.pdf (accessed: 11.08.2012)

[Ovdei & Proskudina, 2004] Ovdei M .O., Proskudina G.U., "*Survey of ontology engineering tools*", (Овдей М. О, Г. Ю. Проскудина "Обзор инструментов инженерии онтологий", Российский научный электронный журнал, «Электронные библиотеки», 2004, т. 7, Вып. 4, ISSN 1562-5419), http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2004/part4/op (accessed: 21.07.2012), (in Russian).

[Owens, 2009] Alisdair Owens, "*An Investigation into Improving RDF Store Performance an Investigation into Improving RDF Store Performance*", Ph.D. Thesis - University of Southampton, 2009.

[OWL, 2004] OWL Web Ontology Language Guide W3C, 2004. http://www.w3.org/TR/owl-guide/ (accessed: 21.07.2012)

[Palagin & Yakovlev, 2005] Palagin A.V., Yakovlev U.S., "*System integration of computer technique*", (Палагин А. В, Ю. С. Яковлев. Системная интеграция средств компьютерной техники/ А. В. Палагин. Винница: УНІВЕРСУМ, 2005, pp. 680, pp. 677-678, ISBN 966-641-140-7), (in Russian)

[Palagin et al, 2011] Palagin A.V., Krivii S.L., Petrenko N.G., "*Ontological methods and instruments for processing domain knowledge*", (А. В. Палагин, С. Л. Крывый, Н. Г. Петренко. Онтологические методы и средства обработки предметных знаний: монография/Луганск: изд-во ВНУ им. В. Даля, 2011. – 300 с.), (in Russian)

[Palagin, 2006] Palagin A.V., "*Architecture of ontologicaly controled computer systems*", (Палагин А. В. Архитектура онтолого-управляемых компьютерных систем /Кибернетика и системный анализ, 2006, №2, pp. 111 – 124), (in Russian).

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

[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.

[Papakonstantinou et al, 1995] Papakonstantinou, Y., Garcia-Molina, H., and Widom, J., "*Object Exchange across Heterogeneous Information Sources*", In Proc. of the 11th Int. Conf. on Data Engineering (ICDE). IEEE Computer Society, 1995, pp. 251–260.

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

[PC mag, 2013] PC Magazine Enciclopedia http://www.pcmag.com/encyclopedia_term/0,1237,t=indexing&i=44896,00.asp (accessed: 23.01.2013)

[Peckham & Maryanski, 1988] Peckham, J. and Maryanski, F. J, "*Semantic data models*", ACM Comput. Surv., 20, 3, 1988, pp. 153–189

[Pentium Dual, 2008] Intel Pentium Dual Core CPU @ 2.8 GHz; CPU Launched: 2008; http://www.cpubenchmark.net/cpu.php?cpu=Intel+Pentium+D+2.80GHz&id=1126 (accessed: 31.07.2013)

[Pfenning, 2012] Frank Pfenning, "*Lecture Notes on Tries*", Lecture 21, In 15-122: Principles of Imperative Computation November 8, 2012. http://www.cs.cmu.edu/~fp/courses/15122-f12/lectures/21-tries.pdf (accessed: 20.07.2013).

[Philpot et al, 2005] Philpot A., E. Hovy, P. Pantel, "*The Omega Ontology*", Information Sciences Institute of University of Southern California, 2005. pp. 8 http://omega.isi.edu/doc/ (accessed: 21.07.2012)

[Pidcock & Uschold, 2012] Woody Pidcock, Michael Uschold, "*What are the differences between a vocabulary, taxonomy, a thesaurus, ontology, and a meta-model?*", InfoGrid - the Web Graph Database http://infogrid.org/trac/wiki/Reference/PidcockArticle, Retrieved November 18, 2012 (accessed: 26.11.2012).

[Polikoff, 2003] Polikoff I., "*Ontology Tool Support*", In: TopQuadrant Technology Briefing, 2003.

[Poprat et al, 2008] Poprat Michael, Elena Beisswanger, Udo Hahn, "*Building a BioWordNet by Using WordNet's Data Formats and WordNet's Software Infrastructure — A Failure Story*", Software Engineering, Testing, and Quality Assurance for Natural Language Processing, Columbus, Ohio, USA, June 2008. Association for Computational Linguistics, 2008, pp. 31-39.

[Poulovassilis & Levene, 1994] Poulovassilis, A. and Levene, M., "*A Nested-Graph Model for the Representation and Manipulation of Complex Objects*", ACM Transactions on Information Systems (TOIS) 12, 1, 1994, pp. 35–68.

[Promt, 2012] http://protege.stanford.edu/plugins/prompt/prompt.html (accessed: 09.08.2012)

[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)

[Q9450, 2008] Intel Core 2 Quad Q9450 @ 2.66GHz, CPU Launched: 2008; http://www.cpubenchmark.net/cpu.php?cpu=Intel+Core2+Quad+Q9450+%40+2.66GHz&id=1046 (accessed: 31.07.2013)

[Ravenbrook, 2010] Ravenbrook, Software engineering consultancy, 2010 Retrieved from http://www.ravenbrook.com/ (accessed: 16.11.2012)

[RDF Suite, 2013] RDF Suite http://www.kp-lab.org/tools/rdfsuit (accessed: 23.03.2013).

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

[rdfedit, 2012] http://www.magnesiummedia.com/pcutilities/details15041.html (accessed: 21.02.2013)

[RDFStore, 2012] RDFStore URL: http://rdfstore.sourceforge.net/documentation/api.html (accessed: 15.10.2012)

[Sahni, 2005] Sartaj Sahni, "*Tries"*, Chapter 28, of "Handbook of data structures and applications" /edited by Dinesh P. Mehta and Sartaj Sahni, Chapman & Hall/CRC computer & information science, 2005, 1321 pages, ISBN 1-58488-435-5.

[sandsoft, 2012] http://www.sandsoft.com/products.html (accessed: 15.10.2012)

[Sesame, 2012] Sesame, OpenRDF, http://www.openrdf.org/index.jsp http://www.openrdf.org/doc/sesame2/2.3.2/users/userguide.html#chapter-sesame2-whats-new (accessed: 01.12.2012)

[Sharoff, 2001] Serge Sharoff, "*The Frequency Dictionary For Russian*", Russian Scientific Research Institute of Artificial Intellect (Russri AI), 2001, http://www.artint.ru/projects/frqlist/frqlist-en.php (С.А.Шаров. Частотный Словарь. РосНИИ ИИ, 2001. http://www.artint.ru/projects/frqlist.php) (accessed: 22.07.2013)

[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, pp. 72–79, Washington, D.C., 1978.

[Sigurd et al, 2004] Bengt Sigurd, Mats Eeg-Olofsson, Joost van de Weijer Word Length, “*Sentence Length and Frequency – Zipf Revisited*”, Studia Linguistica 58(1), Blackwell Publishing Ltd., Oxford, UK, 2004, pp. 37-52.

[Silberschatz et al, 1996] Silberschatz, A., Korth, H. F., and Sudarshan, S. “*Data Models*”, ACM Computing Surveys 28, 1, 1996, pp. 105–108.

[Sintek & Decker, 2001] Sintek M., S. Decker, “*TRIPLE-An RDF Query, Inference, and Transformation Language*”, In: Proceedings of the Deductive Databases and Knowledge Management Workshop (DDLP' 2001), Japan, October 2001, TRIPLE URL: http://triple.semanticweb.org/ (accessed: 15.10.2012)

[Sowa, 2000] Sowa John F., “*Ontology, Metadata, and Semiotics*”, Proceedings of ICCS'2000 in Darmstadt, Germany, on August 14, 2000. Published in: B. Ganter & G. W. Mineau, eds., Conceptual Structures: Logical, Linguistic, and Computational Issues, Lecture Notes in AI #1867, Springer-Verlag, Berlin, 2000, pp. 55-81. http://users.bestweb.net/~sowa/peirce/ontometa.htm (accessed: 10.10.2012)

[Sowa, 2000a] Sowa John F., “*Guided Tour of Ontology*”, http://www.jfsowa.com/ontology/guided.htm (accessed: 28.08.2012)

[SPARQL, 2013] “*SPARQL Query Language for RDF*”, W3C Recommendation, 2008, http://www.w3.org/TR/rdf-sparql-query/ (accessed: 23.03.2013).

[Stably, 1970] Stably D., “*Logical Programming with System 360*”, New York, 1970

[SUMO, 2012] Suggested Upper Merged Ontology (SUMO). http://www.ontologyportal.org/ (accessed: 23.07.2012)

[Sure et al, 2002] Sure Y., J. Angele, S. Staab, “*OntoEdit: Guiding Ontology Development by Methodology and Inferencing*”, CoopIS/DOA/ODBASE, 2002, pp 1205-1222.

[Sure et al, 2003] Sure Y., J. Angele, S. Staab “*OntoEdit: Multifaceted Inferencing for Ontology Engineering*”, J. Data Semantics I, 2003, pp 128-152.

[T9550, 2009] Intel® Core™2 Duo CPU T9550 @ 2.66GHz; CPU Launched: 2009, http://www.cpubenchmark.net/cpu.php?cpu=Intel+Core2+Duo+T9550+%40+2.66GHz&id=1011 (accessed: 31.07.2013)

[Taylor & Frank, 1976] Taylor, R. W. and Frank, R. L., “*CODASYL data-base management systems*”, ACM Comput. Surv., 8, 1, 1976, pp. 67–103

[TBC, 2012] Top Braid Composer http://www.topbraidcomposer.com (accessed: 21.07.2012)

[Tran et al, 2009] Thanh Tran, Gunter Ladwig, Sebastian Rudolph “*iStore: Efficient RDF Data Management Using Structure Indexes for General graph Structured Data*”, Institute AIFB, Karlsruhe Institute of Technology, 2009.

[Troncy & Isaac, 2002] Troncy R. and Isaac A., "*Semantic Commitment for Designing Ontologies: A Tool Proposal*", Poster Session at: 1st International Conference on the Semantic Web, ISWC'2002, Sardinia, Italia, June, pp. 9-12, 2002, Poster

[Tsichritzis & Lochovsky, 1976] Tsichritzis, D. C. and Lochovsky, F. H., "*Hierarchical data-base management: A survey*", ACMComput. Surv. 8, 1, 1976, pp. 105–123.

[TSRD, 2012] Triple Stores vs Relational Databases http://stackoverflow.com/questions/9159168/triple-stores-vs-relational-databases (accessed: 11.01.2013).

[Uschold & Gruninger, 1996] Uschold M. and Gruninger M., "*Ontologies: Principles, methods and applications*", Knowledge Engineering Review, Vol. 11:2, 93-136, 1996, Also available as AIAITR-191 from AIAI the University of Edinburgh.

[van Assem et al, 2006] van Assem Mark, Aldo Gangemi, and Guus Schreiber, "*Conversion of WordNet to a standard RDF/OWL representation*", In: LREC 2006 – Proceedings of the 5th International Conference on Language Resources and Evaluation. Genoa, Italy, May 22-28, 2006. Paris: European Language Resources Association (ELRA), available on CD.

[Velychko & Prihodnyuk, 2013] Velychko V.U., Prihodnyuk V.V., "*Technological tool for graphical design of computer ontologies*", (Величко В. Ю., Приходнюк В. В. Технологическое средство графического проектирования компьютерных онтологий.) In: Troitzsch K. G., Debicki R., Chernyshenko S. V., Romaniuk V.V., Kyrychenko K. I. (eds.) Conference Proceedings "Actual problems of training specialists in ICT", Part 2; Sumy State University, Sumy 2013, pp. 38-43 (in Russian).

[Virtuoso, 2013] OpenLink Virtuoso Universal Server: Documentation http://docs.openlinksw.com/pdf/virtdocs.pdf, http://virtuoso.openlinksw.com/ (accessed: 23.03.2013)

[Webonto, 2012] http://www.aktors.org/technologies/webonto (accessed: 02.09.2012)

[Webopedia, 2013] Webopedia QuinStreet, Inc. http://www.webopedia.com/TERM/I/index.html (accessed: 23.01.2013)

[Weibel et al, 1998] Weibel S., J. Kunze, C. Lagoze and M. Wolf, "*Dublin Core Metadata for Resource Discovery*", IETF #2413, The Internet Society, September 1998, http://dublincore.org/documents/1998/09/dces/ (accessed: 02.09.2012)

[Weiss et al, 2008] Weiss, C, Karras, P., Bernstein, A., "*Hexastore: Sextuple Indexing for Semantic Web Data Management*", In: 34th Intl Conf. on Very Large Data Bases (VLDB), Auckland, New Zealand, 28 August 2008, http://www.zora.uzh.ch/8938/2/hexastore.pdf (accessed: 23.03.2013).

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

[Wilkinson et al, 2003] Kevin Wilkinson, Craig Sayers, Harumi Kuno, Dave Reynolds, "*Efficient RDF Storage and Retrieval in Jena2*", SWDB, 2003.

[Wilkinson, 2006] Kevin Wilkinson, "*Jena Property Table Implementation*", HP Labs, 2006.

[Witte et al, 2010] René Witte, Ninus Khamis, Juergen Rilling, "*Flexible Ontology Population from Text: The OwlExporter*", International Conference on Language Resources and Evaluation

(LREC), Valletta, Malta: ELRA, pp. 3845--3850, 2010 http://www.lrec-conf.org/proceedings/lrec2010/pdf/932_Paper.pdf (accessed: 31.07.2013)

[Wood et al, 2005] David Wood, Paul Gearon, Tom Adams, "*Kowari: A Platform for Semantic Web Storage and Analysis*", WWW 2005, May 10--14, 2005, Chiba, Japan

[WordNet, 2012] Princeton University, "*About WordNet*", WordNet, Princeton University, 2010 http://WordNet.princeton.edu (accessed: 23.07.2012)

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

[YARS, 2013] Andreas Harth, Stefan Decker, "*Optimized Index Structures for Querying RDF from the Web*", Digital Enterprise Research Institute (DERI), National University of Galway, Ireland, http://sw.deri.org/2005/02/dexa/yars.pdf (accessed: 23.03.2013).

[Yongming et al, 2012] Yongming L., F. Picalausa, G.H.L. Fletcher, J. Hidders, Stijn Vansummeren, "*Chapter 2. Storing and Indexing Massive RDF Data Sets*", In: R. De Virgilio, F. Guerra, Y. Velegrakis (eds), "Semantic Search over the Web". ISBN 978-3-642-25007-1 ISBN 978-3-642-25008-8 (eBook), DOI 10.1007/978-3-642-25008-8. Springer Heidelberg New York Dordrecht London, 2012.

[Youn & McLeod, 2006] Seongwook Youn, Dennis McLeod, "*Ontology Development Tools for Ontology-Based Knowledge Management*", In Encyclopedia of E-Commerce, E-Government, and Mobile Commerce, ed. Mehdi Khosrow-Pour, ch138, pp. 858-864 (2006), http://www.igi-global.com/chapter/ontology-development-tools-ontology-based/12642 doi: 10.4018/978-1-59140-799-7 (accessed: 20.07.2013).

[YourDictionary, 2013] YourDictionary, "*LoveToKnow*", http://www.yourdictionary.com (accessed: 20.07.2013).

[Zikopoulos et al, 2012] Paul C. Zikopoulos, Chris Eaton, Dirk de Roos, Thomas Deutsch, George Lapis, "*Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*", Copyright© 2012 by The McGraw-Hill Companies, ISBN 978-0-07-179053-6, MHID 0-07-179053-5, 2012, 166 p.

[Zoho sheet, 2012] https://public.sheet.zoho.com/public.do?docurl=Natural+Language+Formulas&name=m7faALWlvQLgtPUoKu5%2FAA%3D%3D (accessed: 26.11.2012)

Authors' Informtion

Krassimir Markov - PhD, Professor associate at the Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences, Sofia, Bulgaria. His scientific interests are in the field of Intelligent Systems, Multidimensional Information Bases, Business informatics, Management Information Systems, Software Engineering, and General Information Theory. He is Editor in chief of four international journals (since 1993) and more than 50 scientific collections. He was Chairman of more than 110 international conferences and member of program committees of more than 60 international conferences all over the world. He is co-author of 5 monographs and has more than 120 peer-reviewed journal and conference papers.

e-mail: markov@foibg.com

Krassimira Ivanova – PhD, Professor assistant of mathematics at University of National and World Economy, Sofia, Bulgaria. Her research interests are in business informatics, association rules, data mining, multi-variant clustering and analysis high-dimensional data based on multi-dimensional pyramidal multi-layer structures in self-structured systems. She is co-author of two monographs and has 28 peer-reviewed journal and conference papers.

e-mail: krasy78@mail.bg

Vitalii Velychko - PhD, Professor associate at V.M.Glushkov Institute of Cybernetics of National Academy of Sciences of Ukraine. He is a specialist in the field of the development of systems of artificial intelligence. His research interests are in the area of machine learning; knowledge discovery; natural language processing; knowledge base reasoning; inductive and analogical inference. He is co-author of 2 monographs and he has more than 45 international peer-reviewed journal and conference papers.

e-mail: Velychko.Vitalii@gmail.com

Koen Vanhoof - PhD, Professor (Business Informatics) at Hasselt University, Belgium. His research interests focus on business intelligence, business process modeling, e-business strategy, ERP-systems, knowledge discovery management. He is co-author of 7 monographs, and he has more than 160 peer-reviewed journal and conference papers. He has been appointed as a guest professor at the University of Antwerp (Antwerp, Belgium), University of Maastricht (Maastricht, Netherland) and Erasmus University (Rotterdam, Netherlands). Currently he is vice-dean research at the Faculty of Applied Economics and project leader of the Business Informatics research group at Hasselt University.

e-mail: koen.vanhoof@uhasselt.be

Juan Castellanos - PhD, Professor associate at the Universidad Politécnica de Madrid, Facultad de Informática; Campus de Montegancedo s.n., 28660 Boadilla del Monte, Madrid, Spain; Head of Natural Computing Group.

e-mail: jcastellanos@fi.upm.es

ISBN: 978-954-16-0070-2 (printed)

ISBN: 978-954-16-0071-9 (online)