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.

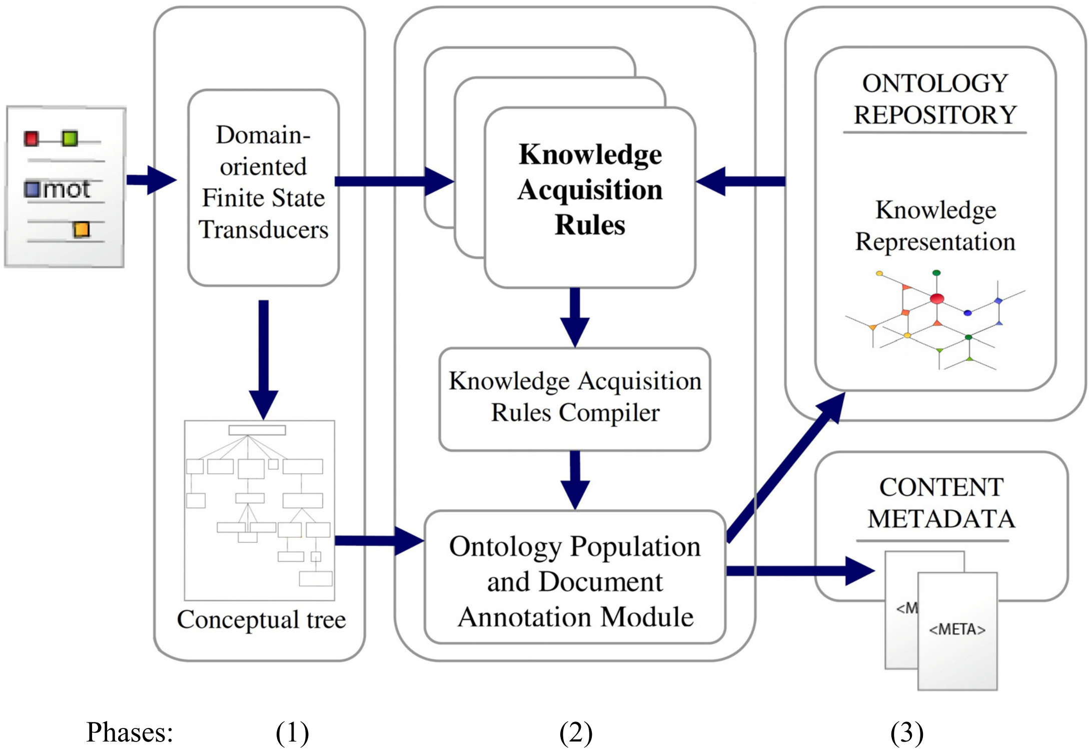

mot
Domain-oriented Finite State Transducers
Conceptual tree
Knowledge Acquisition Rules
Knowledge Acquisition Rules Compiler
Ontology Population and Document Annotation Module
ONTOLOGY REPOSITORY
Knowledge Representation
CONTENT METADATA
<META>
Phases: (1) (2) (3)

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.

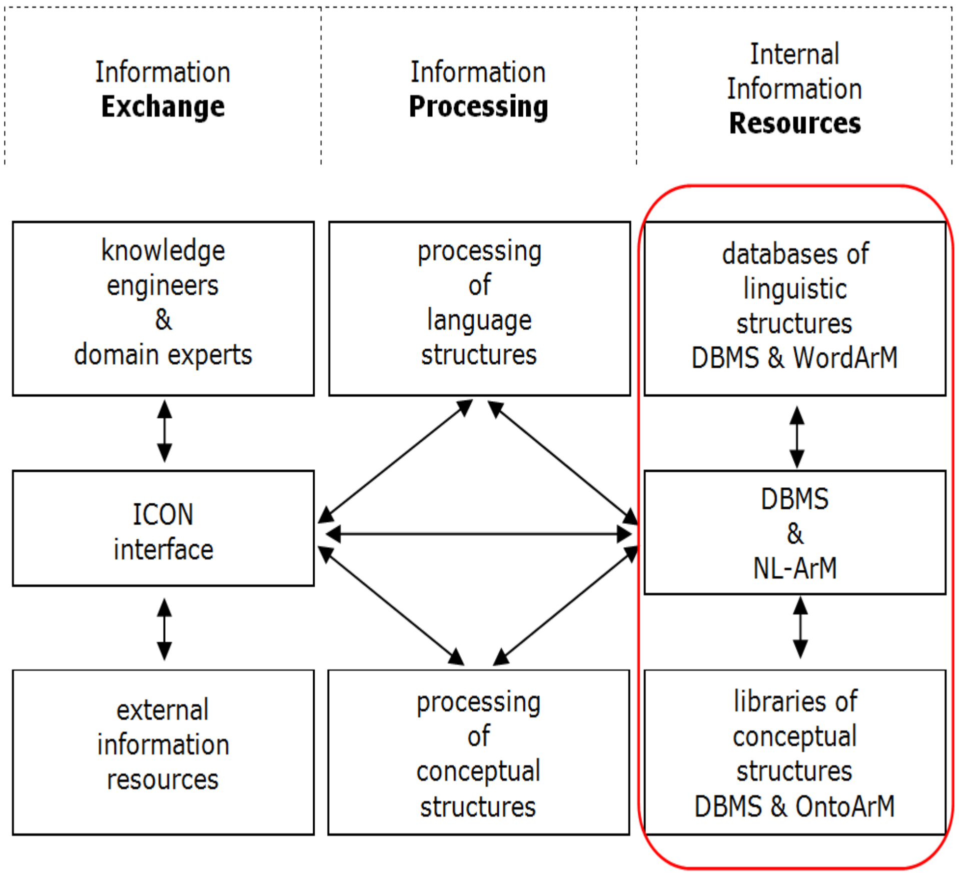

Information Exchange
Information Processing
Internal Information Resources
knowledge engineers & domain experts
processing of language structures
databases of linguistic structures DBMS & WordArM
ICON interface
DBMS & NL-ArM
external information resources
processing of conceptual structures
libraries 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].

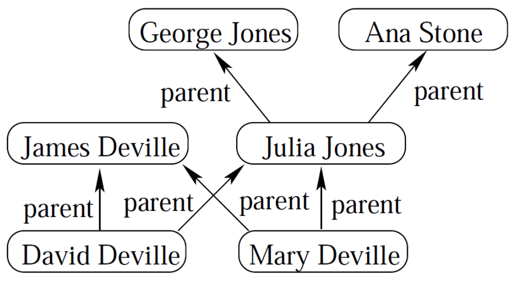
For large ontological structures the Protégé approach is not effective. As we have seen in Chapter 4 “Basic experiments”, the relational data bases are slower than post-relational ones based on NL-addressing, and take much resources for updating the information (especially for updating the indexes). 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 7 (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):

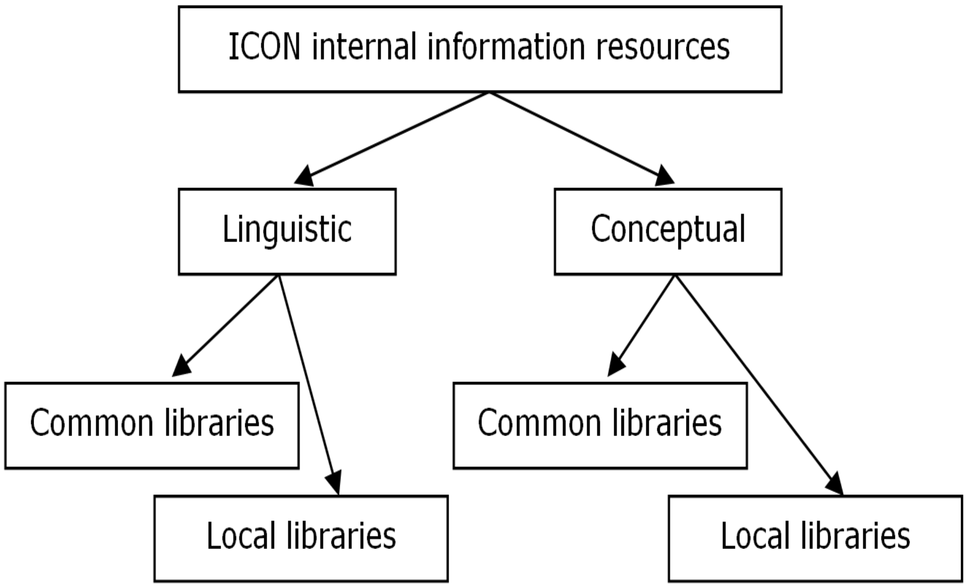


Figure 82. Taxonomy of ICON internal information resources

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 13) 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]).

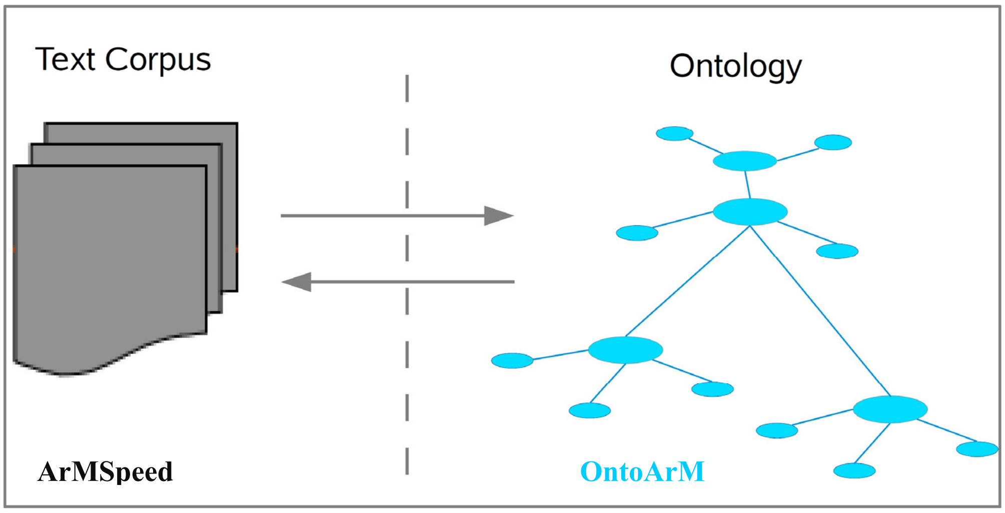


Figure 83. Using OntoArM for storing ontologies of text documents (following [Witte et al, 2010])

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.