Managing Data from Heterogeneous Data Sources Using Knowledge Layer

Krzysztof Goczyła, Teresa Zawadzka, Michał Zawadzki

Gdańsk University of Technology, Department of Software Engineering,
ul. Gabriela Narutowicza 11/12, 80-952 Gdańsk, Poland
{kris,tegra,michawa}@eti.pg.gda.pl

Abstract. In the process of data integration using ontologies it is important to manage data from external data sources in the same way as data stored in the *Knowledge Base*. In previous papers [1], [2] the way of inference from data stored in the Knowledge Base, using *Knowledge Cartography* idea has been presented. However, this solution requires loading all data to the Knowledge Base. The solution presented in this paper shows how the Knowledge Cartography can be used to infer from data stored in external data sources, without loading them to the Knowledge Base. The presented solution is to enrich each data source with an additional layer that allows managing data using signatures. The paper additionally describes the results of tests comparing times of inference when data are loaded to the Knowledge Base and when data are fetched on demand.

1 Introduction

Managing data from heterogeneous data sources using ontologies is a key problem that must be resolved to integrate data [3] [4] and to allow inferring from them. This problem has appeared even more important while the Internet grows larger and more popular. To achieve the aim of data integration, the Semantic Web [5] initiative has been proposed. Within this initiative the OWL [6] language has been standardized. These achievements have been a large step to data integration, however does not resolve all problems.

In the paper we propose that data coming from external data sources (e.g. from web sites) can be integrated with a Knowledge Base (KB) in such a way that logically the data sources become a integral part of the Knowledge Base. From the KB point of view, they comply with the ontology stored in the KB. In our approach, the ontology is formulated in terms of OWL-DL and managed using the Knowledge Cartography – a (presented elsewhere [1], [2]) set of algorithms for processing Description Logics ontologies.

The paper is organized as follows: Section 2 briefly recalls the idea of Knowledge Cartography and describes motivations behind our work. The rest of the paper presents the main contribution of this paper – the Knowledge Layer solution. The architecture of Knowledge Layer is described in Section 3. The next two sections: Section 4 and Section 5 describe specific solutions applied in Knowledge Layer.

In Section 6 the results of efficiency tests comparing times of answering to the queries when data are loaded to the Knowledge Base and when data are fetched on demand from external data sources. Section 7 summarizes the paper.

2 Motivations

The idea of Knowledge Layer appears as a response for the requirement of fetching data from external data sources on demand. The solution applied in KaSeA system [1] requires inserting all data from data sources to the Knowledge Base. We can say that in the KaSeA system there are two main components. The first component – the *Inference Engine* – is responsible for inferring knowledge on the basis of information stored in the *Knowledg Base*. Two kinds of information are loaded to the Knowledge Base: terminology and assertions about individuals.

Following the idea of Knowledge Cartography, each concept has a *signature*. A signature is an array of binary digits representing a region covered by the concept in the map. A map of concepts is basically a description of interrelationships between concepts in a terminology. The map is created in the course of Knowledge Base creation. A map of concepts can be graphically represented in a form similar to a Venn diagram (see Figure 1).

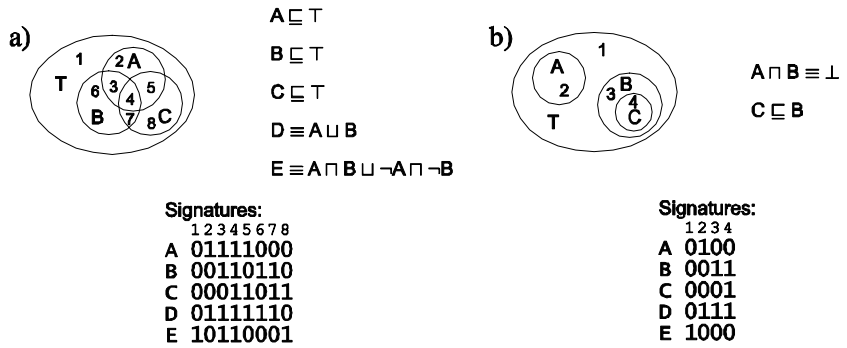

Fig. 1. A map of 5 concepts (a), with two terminological axioms added (b)

Each atomic region (i.e. a region that does not contain any other region) represents a unique valid intersection of base concepts. By valid we mean an intersection that is satisfiable with respect to a given terminology. Intersections of concepts that are not allowed by terminological axioms are excluded from the map. A number of valid atomic regions n is calculated and each atomic region has assigned a subsequent integer number from the range [1, n]. Because any region in the map consists of some number of atomic regions it can be represented by an array of binary digits of length n with "1"s in positions mapped to contained atomic regions and "0"s elsewhere. In this way we achieve a signature and in terms of signatures we can describe any combination of complement, union and intersection of described concepts by simply mapping these operations to Boolean negation, disjunction and conjunction.

Analogically, during loading assertions about individuals, signatures for individuals are specified. The difference between the signature of a concept and the signature of an individual is the way how "1"s at specified positions of signatures are interpreted. In case of an individual, "1" means that the individual can belong to the corresponding atomic region. Also instances of roles are stored in the Knowledge Base. The process of inference is based on comparing signatures. For example, when we ask about all instances of the specified concept, the process of answering this query is reduced to the problem of finding these individuals whose which signatures are subsumed by the signature of the specified concept (signature s_1 is subsumed by the signature s_2 when each bit of signature s_1 is less than or equal to the corresponding bit in the signature s_2).

The solution applied in the KaSeA system has one indisputable advantage – all conclusions about data can be quickly retrieved. However, it has also many disadvantages:

- the process of loading data into Knowledge Base is time consuming,
- there are needed advanced techniques to update data loaded to the Knowledge Base of KaSeA system,
- KaSeA system must observe a data source to react for changes or a data source must notify KaSeA system that update must be carried out,
- the solution is not easily scalable: it is not possible to manage all data from external data sources just by loading them to the Knowledge Base.

In the case of systems which manage few data sources which are actually rarely updated the presented solution can be sufficient. However, for systems which manage numerous data sources which change very often the other way of retrieving data must be developed. The answer for this need is the Knowledge Layer.

3 Knowledge Layer Architecture

The main assumption of Knowledge Layer is the fact that the Knowledge Layer can be queried analogically as KaSeA system (e.g by DIGUT – Description Logic Interface by Gdańsk University of Technology [8]). And, what is even more important, the inference capabilities are the same as inference capabilities of KaSeA system. This is caused by the fact that both KaSeA and Knowledge Layer use Cartographic Representation of knowledge.

Within Knowledge Layer we can distinguish a set of components depicted in Figure 2. Each Knowledge Layer must logically cooperate with KaSeA system with a terminology loaded. During loading terminology all signatures for concepts are calculated. The terminology contains theses notions in terms of which the External Data Source will be queried. Another component is an XML file that contains mappings between the ontology and an external data source. We can distinguish three types of mappings: concept mappings, role mappings and attribute mappings.

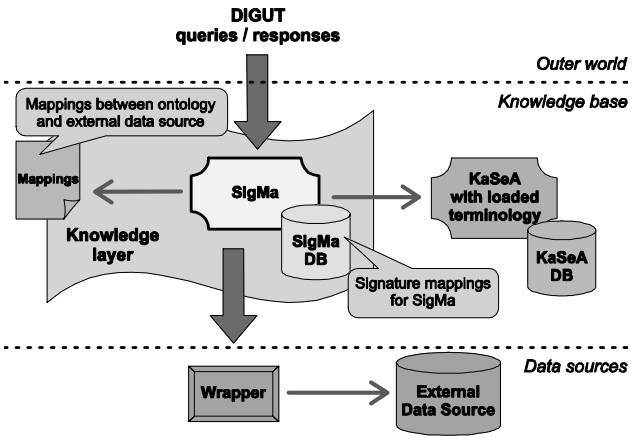

Fig. 2. The Knowledge Layer architecture

A concept mapping is a pair: concept and query allowing to retrieve all individuals belonging to this concept. By role mapping we mean a pair: role and query allowing to retrieve all pairs of individuals which are related to each other via the specified role. Analogically, by the attribute mapping we mean a pair: attribute and query allowing to retrieve values of the specified attribute for the individuals.

The key component of the Knowledge Layer is SigMa (*Signature Mapper*). SigMa is responsible for:

- transforming mappings defined in the file to the signature mappings,
- storing these mappings in the SigMa database and
- answering queries asked in terms defined in the ontology.

The first task requires obtaining information about signatures of concepts from the KaSeA system. Each pair: concept and query is transformed to another pair: signature and query. The query remains unchanged and the signature for the mapped concept (possibly complex) is calculated on the basis of signatures stored by the KaSeA. These mappings are stored in SigMa database. Answering queries asked in terms defined in the ontology requires finding the most suitable query understandable by External Data Source.

The next advantage of the Knowledge Layer is its unawareness of the variety of types of data sources (SQL, XML, CSV, XLS, MDB and so on). The only thing the Knowledge Layer must know are queries defined in the mapping file. However, the Knowledge Layer does not have to know their meanings. Another components, i.e. Wrappers, are responsible for understanding these languages. Such a situation is presented in Figure 3. In this way the process of finding by the SigMa system the most suitable query understandable by the External Data Source is independent on the type of the data source. However, such a solution puts one requirement: there must exist a query language for any, supported by the Knowledge Layer, type of data source. This language must fulfill some requirements (e.g. it must allow to formulate the union of queries returning the set being a union of results of corresponding simple

queries) and in this language the queries in mapping file must be expressed. Obviously, not all previously mentioned types of data sources have such languages.

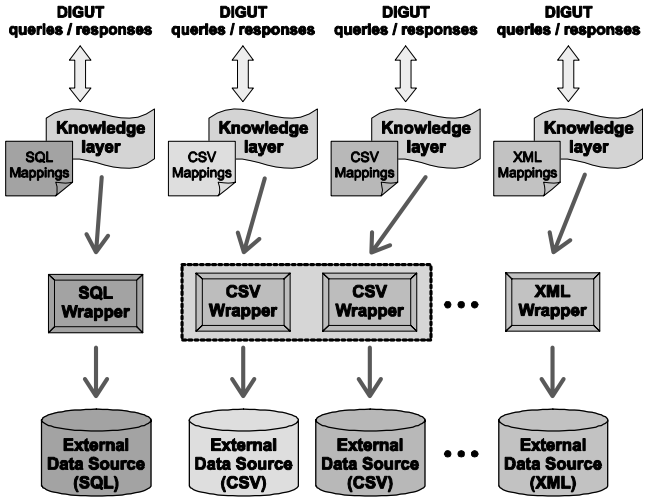

Fig. 3. Knowledge Layer for various types of External Data Sources

Thus, in the course of development of Knowledge Layer also such languages must be specified. The next part of the paper is focused on SQL Data Source because SQL allows for retrieving data and, what is even more important, most data sources are relational.

4 Maximum Coverage algorithm

The key problem in SigMa is to find the most suitable query retrieving individuals for the specified concept. The most suitable queries for mapped concepts are just the queries stored in the database. There is one very important assumption for such queries. The query corresponding to a signature always retrieves all individuals which belong to the concept with that signature. To formulate the most suitable query for concept which is not mapped, and is represented by the signature *s*, SigMa must create a new signature combined from some number of existing (mapped) ones that is subsumed by the signature *s* and which covers the maximum number of atomic regions represented by the signature *s*. It is important to notice that by taking the signature which covers the maximum number of atomic regions represented by the signature *s* we guarantee that all individuals stored in the External Data Source which certainly belong to the concept represented by the signature *s* will be retrieved.

However, it can be a situation when a query concerns the signature *s* = 001100 representing concept *People*, for example and in the External Data Source only a concept *Women* is mapped (in the EDS no information about men is included). Let us assume that the concept *Women* is represented by the signature 001000. In such

a situation SigMa creates a signature 001000 and this signature for this specific EDS covers the maximum number of atomic regions represented by the signature 001100.

The list of signatures that must be combined is deduced by the MC (*Maximum Coverage*) algorithm described below. The MC algorithm is based on the Apriori algorithm [9].

Algorithm 1. Maximum Coverage algorithm

Input: A signature s.

Output: A union of intersections of signatures that covers maximal number of atomic regions described by signature s.

1. If signature s exists:
2. Return the signature.
3. Else
4. Make lists l_0, l_1, l_2 empty.
5. Find all mapped signatures that are not disjoined with and not subsumed by s and append them to the list l_0.
6. Find all mapped signatures that are subsumed by s and append them to the list l_1.
7. If list l_0 is not empty:
8. For each pair (s_i, s_j) such that $s_i, s_j \in l_0$; $s_i \neq s_j$:
9. Calculate signature $s_t = s_i$ AND s_j (keep track of signatures used)
10. If $s_t = s$
11. Return a list of signatures whose intersection created signature s_t.
12. Else If s_t is subsumed by s append s_t to the list l_1.
13. Else append s_t to the list l_2.
14. End
15. While list l_2 is not empty:
16. Copy list l_2 to l_3 and clear list l_2.
17. For each pair (s_i, s_j) such that $s_i \in l_3$, $s_j \in l_0$:
18. Calculate signature $s_t = s_i$ AND s_j (keep track of signatures used)
19. If $s_t = s$
20. Return a list of signatures whose intersection created signature s_t.
21. Else If s_t is subsumed by s append s_t to the list l_1.
22. Else append s_t to the list l_2.
23. End
24. End
25. End
26. For each $s_t \in l_1$ delete those s_t which are subsumed by any other s_t in l_1.
27. List l_1 contains list of lists of signatures. Return this list as a union of intersections of signatures from list l_1.
28. End

5 SigMa system design

Having defined the MC algorithm it is possible to describe the solutions applied in SigMa to provide services responsible for creating signature mappings, storing these mappings in the SigMa database and answering queries asked in terms defined in the ontology. The next three subsections describe how these aims have been achieved.

5.1 Creating signature mappings

Creating signature mappings from concept, role and attribute mappings defined in the mapping file is the first task of SigMa component. Firstly, signatures for mapped concepts are calculated with the use of KaSeA system with loaded terminology. In this way it is possible to specify list of pairs consisting of a signature and a query corresponding to that signature. The pair: a signature and a query is further referred to as a signature mapping. Such a list of signature mappings is stored in the database. Secondly, signature mappings for $\neg C$ concepts, where C is a mapped concept, are created. To create a query for the signature of $\neg C$ concept, the MC algorithm is used. The last step of creating signature mappings is creating signature mappings for Top concept (using the MC algorithm) and for concepts $\exists R.C$. Having defined the query for the role R and the query for concept C (if C concept is not a mapped concept, then the query can be created using the MC algorithm) SigMa creates the query: "return all role subjects of the role R for which the role filler is an instance of the concept C". All these signature mappings are stored in the database.

5.2 SigMa Database

The schema of SigMa database is quite similar to that specified for the KaSeA system [10]. The main difference is the fact that in the SigMa database there is no information about individuals, about their names, relations between them and their signatures. However, there is an additional entity storing information about queries. The ERD diagram for the SigMa database is depicted in Figure 3.

The main entity sets presented in the logical schema are: *ConceptDefs*, *AttributeDefs*, *RoleDefs*, *Signatures* and *Queries*. *ConceptDefs*, *AttributeDefs* and *RoleDefs* store information about mapped concepts, attributes and roles respectively. *AttributeDefs* and *RoleDefs* are related to the entity *Queries* in that way, that for each attribute there exists a relationship to the query that returns a list of pairs of individuals and a value of the attribute, and for each role there exists a relationship to the query that returns a list of role instances. Concepts are not directly related to the queries but through their signatures. There is one more entity *RoleConcepts*. In this entity all defined in the terminology concepts of the type $\exists R.C$ are stored. This entity is related to the *Signature* entity via the two relationships. The *existsHasSignature* says what the signature is for the concept of the form $\exists R.C$. And the second relationship *conceptHasSignature* says what the signature is for the concept C. The description of the specified entities are included in Table 1.

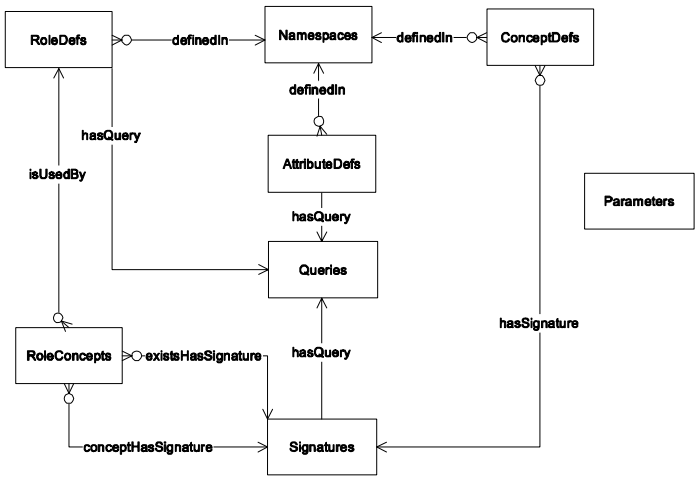

Fig. 4. ERD diagram for SigMa database

In the *Signatures* entity there are some additional attributes *hash* and *sec_i*, which have been introduced to make finding needed signatures more efficient. The application of these attributes has been described in detail in [10]. Entity *Parameters* stores the length of signatures and the entity *Namespaces* stores namespaces for the terms defined in the ontology.

Table 1. Short description of entities of SigMa database

Entities	Description	Entity Attributes
ConceptDefs	Stores information about mapped concepts	*id* – concept identifier *name* – a local name of the concept
RoleDefs	Stores information about mapped roles	*id* – role identifier *name* – a local name of the role
AttributeDefs	Stores information about mapped attributes	*id* – attribute identifier *name* – a local name of the attribute
Queries	Stores queries understandable by a wrapper of a specified type	*id* – query identifier *content* – content of the query
Signatures	Stores signatures used in signature mappings	*id* – signature identifier, *hash* – hash code for the signature, *sec_i* – no of ones in consequent *n* sections of the signature.
RoleConcepts	Stores concepts of the for $\exists R.C$	No attributes are defined
Namespaces	Stores namespaces	*id* – namespace identifier *uri* – uri of the namespace
Parameters	Stores parameters specific for the terminology	*name* – name of the parameter *value* – value of the parameter

5.3 Processing queries

Knowledge Layer allows processing queries about such inference problems as: instance retrieval problem, instance check problem, related individuals problem, role fillers problem and told values problems [8]. The processes of answering queries are presented below as generic algorithms which can be optimized for specific query languages and are dependent on wrappers capabilities.

Algorithm 2. Algorithm for instance retrieval problem

Input: A concept *C*.
Output: A set of individuals belonging to the concept *C*.

1. Find a signature for the concept *C*
2. Find a query for that signature
3. If the signature is not mapped to the query
4. Find the most suitable query (with maximal coverage) using the MC algorithm
5. Else
6. Fetch the query from the database
7. Execute the query
8. Return the result of the query

Algorithm 3. Algorithm for instance check problem

Input: A concept *C*, an individual *i*.
Output: *True* when *i* belongs to the concept *C*, *false* when *i* does not belong to the concept *C* and *maybe* otherwise.

1. Find a signature for the concept *C*
2. Find a query for that signature
3. If the signature is not mapped to the query
4. Find the most suitable query (with maximal coverage) using MC algorithm
5. Else
6. Fetch the query from the database
7. Execute the query
8. Check if the individual *i* belongs to the result of the query
9. If the individual *i* belongs to the result of the query
10. Return *true*
11. Else
12. Find a signature for the concept $\neg C$
13. Find a query for that signature
14. If the signature is not mapped to the query
15. Find the most suitable query (with maximal coverage) using MC algorithm
16. Else

```
17.         Fetch the query from the database
18.     Execute the query
19.     Check if the individual i belongs to the result of the query
20.     If the individual i belongs to the result of the query
21.         Return false
22.     Else
23.         Return maybe
```

It is worth noticing that despite the fact that External Data Sources are modeled in CWA (*Closed World Assumption*) [11], Knowledge Layer opens this world and provides answers according to OWA (*Open World Assumption*). In case of instance retrieval problem the algorithm returns only these individuals for which it is certain that they belong to the specified concept. In case of instance check problem the algorithm returns *true* when it is certain that the specified individual belongs to the specified concept, *false* when it is certain that the specified individual does not belong to the specified concept and *maybe* when it cannot be unambiguously stated whether the individual belongs to the specified concept or not.

6 Efficiency tests

The KaSeA system provides inference capabilities of various types. Firstly it allows inferring implicit knowledge both from terminology and assertions about individuals. The Knowledge Layer only allows inferring from assertions about individuals. Table 2 compares times of responses for the query about individuals belonging to the specified concept (i.e. instance retrieval problem). The test is carried out for the Drug ontology and Farmadati data source. Drug ontology has been developed by the University of Liverpool within the PIPS project. The ontology contains information about drug manufacturers, active ingredients of drugs, interactions about them and also about ATC (*Anatomical Therapeutic Chemical*) code. Farmadati data source is a relational database managed by Oracle 9i that contains information about drugs, stored in 13 tables. These tables contain 250496 rows. In a presented architecture we can treat SQL Server as a Wrapper. In case of the Knowledge Layer times of answering queries strongly depends on the efficiency of various wrappers. The tests were performed on a PC with Pentium 4, 3GHz and 1GB RAM.

Table 1 presents times of answering queries for instance retrieval problem for different concepts. The results are average times of same tests repeated 100 times.

Table 2. Times of answering queries for instance retrieval problem

	KaSeA	**Knowledge Layer**
Top	time to long to count	22687 ms
Drug	27407 ms	4656 ms
Drug $\int$ *DrugContainer*	35094 ms	11375 ms
$S \int U \int V$	27305 ms	18969 ms

The two first concepts are mapped explicitly; it means that the appropriate query for the signature of *Top* concept and *Drug* concept are stored in the SigMa database. For the two next concepts (*S*, *U* and *V* concepts are subconcepts of the *ATCCode* concept) the query must be formulated on the basis of the MC algorithm. The Knowledge Layer is able to cache queries previously issued. It means that when a query about the instances of concept which signature is not mapped to the appropriate query in the SigMa database is issued, then an appropriate mapping is added to the database. Then when the same query is issued again, the query is directly fetched from the database and does not have to be created anew. This feature decreases the time of executing third and fourth query for about 3000 ms.

7 Related work

The authors of this paper personally experienced the need of managing data from external data sources within PIPS project (*Personalised Information Platform for life and health Services*) [7]. PIPS is a 6th European Union Framework Programme project whose main goal is to create a Web infrastructure to support health and promote healthy life style among European communities. One of its main aims is to develop knowledge management tools covering different information sources.

In the course of the project we have developed the KaSeA (acronym for *Knowledge Signature Analyzer*) [1], [2], system, which allows efficient reasoning from data about large numbers of individuals. The KaSeA system has been developed using *Knowledge Cartography* idea. However, the KaSeA system requires all data to be loaded into the Knowledge Base. It appears that this requirement cannot be always fulfilled. Thus in the PIPS project the other solution enriching data source with *Knowledge Layer*, which allows treating data from the external data source in the same way as data stored in the Knowledge Base, has been developed.

Other known practical solutions are data integration systems like Information Manifold [13], SIMS [14] or PICSEL [15]. However, most of these systems have been developed with the use of the global view approach when the OWL language was not a W3C standard. The systems were focused on particular field of application and therefore also the type of data sources. Despite the fact they were focused on performance – they cannot fulfill the requirements put by Internet which the Knowledge Layer has been developed for.

8 Summary

The presented solution of Knowledge Layer has many advantages:

– it is independent on the format of data source – for structured data sources like XML sources, relational sources or any other sources which have a query language and the processor for this language the only thing that is needed is to define appropriate mappings in the XML file;

- it can be applied for data sources which do not have any query language defined (e.g. XLS files) – in such a situation there must be defined the way of querying a data source and must be developed a kind of wrapper;
- all changes in data source that do not affect the structure of that data source are always visible in Knowledge Layer without any updates;
- changes in the structure of a data source require changing the mappings defined in the XML file – the process of creation new signature mappings is much shorter than the process of loading data into KaSeA system, e.g. for Farmadati data source the time of loading data into KaSeA system is about 48 hours and the time of creation signature mappings is about 3 minutes;
- no advanced techniques for update of knowledge base are required.

References

1. Goczyła K., Grabowska T., Waloszek W., Zawadzki M.: *The Cartographer Algorithm for Processing and Querying Description Logics Ontologies*. LNAI 3528: Advances in Web Intelligence, Third International Atlantic Web Intelligence Conference, Springer 2005. pp. 163-169.
2. Goczyła K., Grabowska T., Waloszek W., Zawadzki M.: The Knowledge Cartography – A new approach to reasoning over Description Logics ontologies. SOFSEM 2006: Theory and Practice of Computer Science, LNCS 3831, pp. 293-302.
3. Calvanese D., Giacomo D. G., Lenzerini M.: Ontology of integration and integration of ontologies. Proceedings of the International Workshop on Description Logics, 2001.
4. Calvanese D., De Giacomo G., Lenzerini M: A Framework for Ontology Integration. Proceedings of the First Semantic Web Working Symposium, 2001, 303-316.
5. Semantic Web Initiatives, http://www.semantic-web.org/.
6. OWL Web Ontology Language Guide, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-guide/
7. Goczyła K., Grabowska T., Waloszek W., Zawadzki M.: *Inference Mechanisms for Knowledge Management System in E-health Environment*, In: „Software Engineering: Evolution and Emerging Technologies", Eds. K. Zieliński, and T. Szmuc, IOS Press, Series: „Frontiers in Artificial Intelligence and Applications", 2005, pp. 418-423.
8. *DIGUT Interface Version 1.3*. KMG@GUT Technical Report, 2005, available at http://km.pg.gda.pl/km/digut/1.3/DIGUT_Interface_1.3.pdf.
10. Wittem I. H., Frank E.: *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publisher 2000.
11. Waloszek W.: *Cartographic Method of Knowledge Representation in KaSeA*, Technologie Przetwarzania Danych, Wydawnictwo Politechniki Poznańskiej, 2005, pp. 14-25 (in Polish).
12. Baader F. A., McGuiness D. L., Nardi D., Patel-Schneider P. F.: The Description Logic Handbook: Theory, implementation, and applications, Cambridge University Press, 2003.
13. Levy A. Y.: *The Information Manifold Approach to Data Integration*, IEEE Intelligent Systems, numer 13, 1998.
14. Arens Y., Knoblock C. A., Shen W.: *Query Reformulation for Dynamic Information Integration*, Journal of Intelligent Information Systems, 1996.
15. Lattes V., Rousset M.-C.: *The use of CARIN language and algorithms for Information Integration: the PICSEL project*, W: Proceedings of the ECAI-98 Workshop on Intelligent Information Integration, 1998.