Transforming Current Web Sources for Semantic Web Usage*

Martin Rimnac

rimnacm@cs.cas.cz
Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodarenskou vezi 2, 182 07 Prague 8,
Czech Republic

Abstract. The paper proposes a data structure modelling method, which aim is to estimate a structure model from a given input data set. The model can be seen as an estimate of data semantics – the obtained relations can be transformed into an RDF or OWL semantic web format documents to be included into the semantic web portfolio. The proposed method makes a connection between current web sources and the semantic web vision to be realized. Finally, the method usage and conversion rules are illustrated on an example.

The information retrieval "on the web" [1] approach continually develops: There are new methods for web pages mapping, for a similarity evaluation between documents or new methods for page ranks. Many of them interpret a web page as a simple list of words, nothing else. Alternatively, information system web interfaces or content managers presenting a view over databases via a XHTML page or the plain text (CSV format) are important because of enabling an inverse task - an information extraction from a web page followed by advanced data processing.

Alternatively, the semantic web vision [2] requires the presentation of information also in machine-friendly formats primary describing information semantics. The description can be made by the RDF format [3] using a predicate formalism organising a knowledge base as binary predicates $feature(object, subject)$, or by the OWL format [4], describing reality by item-set assignment.

The aim of the presented work combining these technologies is to enable current web sources to be processed by semantic web tools with the best to be known about these sources - their content estimated semantics.

The paper presents a data structure estimation process, gives a description of the estimated model properties and finally shows how to convert the model into the relational database schema or, as in [5], into semantic web RDF or OWL formats, and discuses feasibility and usefullness of the format choice.

* The work was supported by the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realisation" and partly by the Institutional Research Plan AV0Z10300504 "Computer Science for the Information Society: Models, Algorithms, Applications".

1 Data Structure Estimation Method

The `data structure estimation` is a machine learning method estimating a relational data model from a given input data set represented as a table (or in any format transformable into a table). The model is described by an "extensional functional dependency system", then data are stored in a repository implemented via a universal relation [6, 7] according to the estimated model, which moreover leads to an effective data storage. The model satisfying the third normal form (3NF) is on-line updated upon a new table row insertion into a repository. The method can be designed also as off-line one (processing all rows in one time), but in such a case no incremental knowledge base building is possible.

As in the area of other machine learning methods, some inductive logic programming tasks [8] can be transformed into the data structure estimation. Main similarities are connected with decision tree generating algorithms [9] (the type of knowledge organisation), but using different criteria: local (as an entropy) for decision trees versus a global one for the structure estimation.

The data structure estimation method is partly based on a database model decomposition. In the past, methods leading to relational data model modifications have been published. The input of these methods is usually a set of functional dependencies describing the current model and requirements for a new one (normal forms, multi-level relations, etc.). These methods use the theory of sets or graph/hypergraph theory [10–12] for relation and subrelation interpretation. An integral part of these methods is often a functional dependency set kernel construction from a given dependency set [13] or the step by step decomposition from one relation into its subrelations and removing redundant functional dependencies [14]. The similar issue is about the relational data model conversion into a multi-level secure one [15]. All of these methods fulfil requirements such as minimal redundancy, representativeness or separativeness [16].

The previously cited approaches describe methods being designed for an intensional functional dependency system (1). This paper topic belongs to the area of the `database dependency discovery` [17–19] working with an (data driven) `extensional "functional dependency system"` (2). The proposed method is more focused on the problem of NP–complexity [20] reduction keeping all instances in an input set to be processed instead of a random selection of several ones and consideration of "degree of true" measures for functional dependencies [19]. The functional dependency system potentionally has a cyclic structure being similar with the dependency network approach [21] enabling probabilistic acyclic Bayesian networks to work with structures containing cycles.

Several ideas used in the proposal are inspired by the vertical [22] and horizontal [23] partitioning or the XML document integration [24, 25].

The main goal of the data structure estimation is to face up the self-describing feature of data sets. It may be used not only for an effective storage (a schema design) of data extracted from web sources followed by an sofistifycated structure consideration, but also in the artificial intelligence area (similar with the inductive logic programming [8]) or especially in this paper for repository instance publication in any semantic web format respecting the estimated structure.

1.1 Functional Dependency System

The data structure estimation method uses the relational database theory as a basic concept for structure description and the graph theory for functional dependency interpretation (there exists an oriented arc between attributes represented by nodes in the data structure model graph, if the second attribute functionally depends on the first one).

Let us denote by $A = \{A_i \mid i = 1 \dots n\}$ the set of all single attributes A_i in the relation R and by $\mathscr{D}(A_i)$ the domain of the attribute A_i. Further, let a model M describes valid relationships between attributes in A. The reality corresponding relationship between the (complex-)attributes $B_i \subset A$ and $B_j \subset A$ will be called the `(intensional) functional dependency` (notation $B_i \rightarrow B_j$) with the meaning that a value of the attribute B_j is uniquely defined by a value of the attribute B_i in all possible relations R. The consequence is the existence of the corresponding injection $\mathscr{P} : \mathscr{D}(B_i) \rightarrow \mathscr{D}(B_j)$.

$$(B_i \rightarrow B_j) \in M \Rightarrow \exists \mathscr{P} : \mathscr{D}(B_i) \rightarrow \mathscr{D}(B_j) \tag{1}$$

This notion of functional dependencies is used in relational database theory, but the area of the dependency discovering approaches [17–19] uses a different one. Let us denote by R' the subrelation of R with the same attribute set A and by $\mathscr{D}_\alpha^{R'}(A_i)$ the `attribute` A_i `active domain related to` R'. The injection $\mathscr{P}' : \mathscr{D}_\alpha^{R'}(B_i) \rightarrow \mathscr{D}_\alpha^{R'}(B_j)$ valid over R' will define the `extensional functional dependency` in R' (notation $B_i \rightarrow_{R'} B_j$; the index R' will be usually omitted in the rest of the paper). The dependency $B_i \rightarrow_{R'} B_j$ may not be satisfied by another R'' subrelation of R or may not respect a relationship given by a reality. The model will be called extensional (notation M^E), when it contains at least one extensional functional dependency.

$$(B_i \rightarrow_{R'} B_j) \in M^E \Leftrightarrow \exists \mathscr{P}' : \mathscr{D}_\alpha^{R'}(B_i) \rightarrow \mathscr{D}_\alpha^{R'}(B_j) \tag{2}$$

Note that the extensional functional dependency system model M^E allways principally depends on the given R', but the intensional dependency system does not (instances are depended on the logical – given by reality – relationships, but the logical relationships are independent of instances). So the extensional functional dependency system only aggregates the structural relationships as important characteristics of R'. These structural relationships may be changed by another R'' considering. Comparison of (1) and (2) leads to the consequence that all data driven reverse engineering methods can be seen `only as an estimation` due to the fact:

$$M \subseteq M^E \tag{3}$$

In the following, the shortcut name "functional dependency" will be used instead of "extensional functional dependency".

1.2 Properties of Functional Dependencies

If attributes depend mutually, they have the same size of their active domains.

$$A_i \rightarrow A_j \wedge A_j \rightarrow A_i \Rightarrow \|\mathscr{D}_\alpha(A_i)\| = \|\mathscr{D}_\alpha(A_j)\| \tag{4}$$

The set of functional dependencies has a transitive property:

$$A_i \rightarrow A_j \wedge A_j \rightarrow A_k \Rightarrow A_i \rightarrow A_k \tag{5}$$

The functional dependency may exist only in the case, when the size of the depending attribute active domain is not greater than the size of the active domain of the attribute on which it depends.

$$A_i \rightarrow A_j \Rightarrow \|\mathscr{D}_\alpha(A_i)\| \geq \|\mathscr{D}_\alpha(A_j)\| \tag{6}$$

Trivial functional dependencies in fact do not describe a model, they are valid independently on the concrete model. They are, in most cases, in the form:

$$\begin{aligned} A_i &\rightarrow A_i \\ A_i &\rightarrow \oslash \end{aligned} \tag{7}$$

If a value of the attribute B_j depends on the attribute B_i, the (complex-)attribute $B_j \cup A_j$ depends also on the (complex-)attribute $B_i \cup A_i$, where $A_i, A_j \subset A$. These dependencies are sometimes also called trivial, if $A_i \cup A_j \neq \oslash$.

$$B_i \rightarrow B_j \Rightarrow B_i \cup A_i \rightarrow B_j \cup A_j \quad \forall A_i, A_j \subset A \tag{8}$$

1.3 Model Skeleton

The method input is a table with n columns (coresponding to a relation with n attributes) and m rows; the method output is a minimal functional dependency set being satisfied by all input data. This set is called `the model skeleton` and represents so called `fundamental relationships` in the model. This means no trivial or derived via transitivity dependency occur in the model skeleton. Moreover all functional dependencies in the model can be derived from the skeleton. In the set theory notion, the model skeleton corresponds to a kernel of functional dependency set and all dependencies in the model to a transitive closure of the kernel. The related repository is self-organised as instances (associative rules) of functional dependencies contained in the model skeleton.

The kernel construction belongs in NP–complete tasks [16, 20] with not unique solution. Due to an incremental model building together with the fact that after the first row gathered each attribute depending on another one, the model skeleton can be created by one of two polynomially complex ways:

- The first skeleton type, called the `linear skeleton` – given on Fig. 1, is based on a random order of attributes and the connection of attributes in their neighbour by an oriented arc (a functional dependency). The number of possible order configurations is $n!$, the maximum cycle length is $2(n-1)$.
- The second skeleton type, the `star skeleton`, randomly chooses one from attributes and consideres functional dependencies between this attribute and all others, see Fig. 2. The number of possibilities is less, only n, and the cycle length is 2 for mutually dependencies or 4 for the others.

All model skeletons must be a combination of these two basic types, the properties of these skeletons are given by the properties of the basic types. The following process is still independent on the model skeleton configuration.

Fig. 1. Linear Skeleton

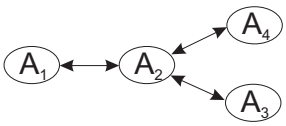

Fig. 2. Star Skeleton

1.4 Initialising and Updating Model Skeleton

The model skeleton is initialised by the first row of an input table via one of the previous ways. The model contains n^2 functional dependencies $A_j \rightarrow A_i$, but implicitly covers $n!$ of them (the $n!$ includes also all the trivial ones). These n^2 functional dependencies can be covered by only $2(n-1)$ ones using the model skeleton (see Fig. 1, 2).

Generally, from another point of view, the data structure estimation process can be seen as a state space scanning one; the state space is bounded by the functional dependency hierarchy given by (8) and the transitivity (5) together with the model skeleton configuration choice. It is clear this can reduce the effective problem complexity (bounds a state space of possible models to scan).

Let us assume the attribute order according to their active domain size given by criterion:

$$\|\mathscr{D}_\alpha(A_i)\| < \|\mathscr{D}_\alpha(A_j)\| \Rightarrow i < j \tag{9}$$

This attribute order enables to define `fundamentality` of the functional dependency $A_i \rightarrow A_j$ as $\delta(A_i, A_j)$ representing the distance between the given attributes in the order (9).

The active domain size change leading to the swapping of attributes in the order may cause the situation (10 or 11), where the model skeleton S includes less fundamental functional dependency (which was the fundamental one before the changes) than feasible, there exists more fundamental functional dependency in the skeleton transitive closure $\overline{S}$

$$(A_i \rightarrow A_k), (A_i \rightarrow A_j) \in S \wedge (A_j \rightarrow A_k) \in \overline{S} \wedge \delta(A_i, A_k) > \delta(A_j, A_k) \tag{10}$$

$$(A_i \rightarrow A_k), (A_j \rightarrow A_k) \in S \wedge (A_i \rightarrow A_j) \in \overline{S} \wedge \delta(A_i, A_k) > \delta(A_i, A_j) \tag{11}$$

In this case, the less fundamental functional dependency is removed out from the model skeleton and the more fundamental is given into. The situation (10) is illustrated in Fig. 3 and the skeleton after update in Fig. 4. The corresponding modifications (transforming into the new skeleton) are made in a repository.

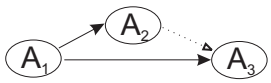

Fig. 3. $\delta(A_1, A_3) > \delta(A_2, A_3)$

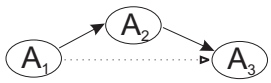

Fig. 4. Skeleton after update

1.5 Testing Functional Dependency Corruption

The next step after the model skeleton update is a testing to functional dependency corruption. The extensional functional dependency is corrupted $A_j \nrightarrow A_i$ (at m'-th row insertion) respecting (2), when

$$\exists k < m' \leq m \quad a_j^k = a_j^{m'} \wedge a_i^k \neq a_i^{m'} \tag{12}$$

The corrupted functional dependencies are removed from the model. Thanks to (5) and transitive closure properties, it is necessary to test only functional dependencies in the model skeleton and if any failure occurs, then also all functional dependencies with the same left or right side must be tested. Note that no corrupted functional dependency can be derived from the model skeleton.

1.6 Complex Attribute Nontrivial Functional Dependencies

If the functional dependency is corrupted $A_j \nrightarrow A_i$ and there exists at least one (denote n') of corrupted functional dependencies $A_k \nrightarrow A_i$ (for $n' = 2$ see Fig. 5), the attribute A_i may depend on the combination of $n' + 1$ attributes, so it may exists functional dependency $\{A_j, A_k\} \rightarrow A_i$. If the test (12) passes, the complex attribute is added into the model and the model skeleton is extended by this new functional dependency (Fig. 6).

If $n' > 1$ and the test passes, it is necessary to make sure the dependency is not trivial before the new dependency insertion (A_i may depend on a subpart/subparts of the complex attribute (8), see Fig. 7). In this case, the complex attribute is decomposed and non-trivial functional dependency/dependencies are inserted into model skeleton by the same way as in the $n' = 1$ variant.

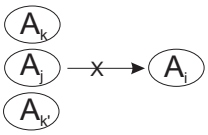

Fig. 5. $A_j \nrightarrow A_i$

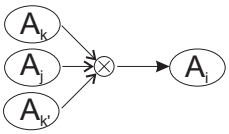

Fig. 6. Complex attribute

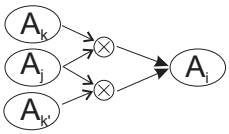

Fig. 7. Decomposition

It is easy to see the complex attribute decomposition is a binomial problem, the number of possible combinations k is:

$$k = \max_{\forall i < n'} \binom{n'}{i} = \binom{n'}{n'/2} = \frac{n'!}{(n'/2)!^2} \tag{13}$$

This process makes the data structure estimation to be a task with non-polynomial complexity.

1.7 Model Properties

The valid models during the estimation process can be ordered according to the number $|M|$ of functional dependencies they cover (skeleton, closure, trivial). Because the functional dependencies are only removed (or transformed into already covered ones), this number is monotonic non-increasing.

$$|M_0| = n! \tag{14}$$

$$M_i < M_j \Rightarrow |M_i| > |M_j| \tag{15}$$

$$\forall M_k : M_0 \leq M_k \leq M_m \leq M_\infty \tag{16}$$

The model M_m after all m rows are gathered may embody some differences to the logical model M_∞. They may be caused by:

- Data non-representativeness - a classical issue of machine learning methods.
- Dependency on sources - a structure or data may depend on a source.
- Granularity - a finite number of rows versus possibility of infinite domains.
- Extensional dependency system - the functional dependencies in the extensional model may not be the same in the intensional model (3).

The first two reasons may be inhibited by an integration process, the last two are fundamental limitations of the method. The problem can be expressed, for example, by a relationship between real number attributes.

The method assumes an error-less input data set. The row containing error may cause a bad dependency test (12) result, the functional dependency non-covered in M_∞ may be covered by M_m; this error type looks like data non-representativeness having no effect in the model. Alternatively, the error corrupting functional dependency covered by M_∞ causes

$$M_m > M_\infty \tag{17}$$

2 Interpreting Model Skeleton

The method described in previous section estimates the data structure and stores all input instances into a repository. This section presents rules interpreting this result to enable stored instances extended by structure metainformation to be transformed to established data formats, which could be used for processing these data by standard tools in given area. Especially, it enables to convert current web sources into semantic web formats.

The conversion from a repository into the relational data model schema, RDF or OWL format will be shown in the following example[1] (Fig. 8) given as a data structure estimation method result and complemented by one row from a repository.

[1] The rules have been verified by the experimental application on the Czech National Bank exchange rates available at `<http://wdb.cnb.cz/CNB_TXT/KURZY.K_CURRTXT>`

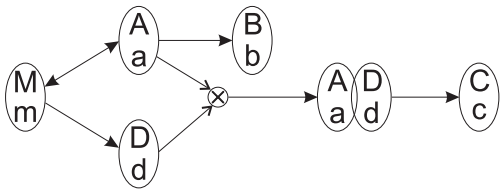

Fig. 8. Example - A row instance of the model skeleton

2.1 Relational Data Model Schema

The way, how to make a schema from a functional dependency set, is well-known from other algorithms [13, 14]. The functional dependencies with same left side are joined into one relation, the left side represents the primary key (signed ^{PK}).

$$(A_i \to A_j) \in S \wedge (A_i \to A_k) \in S \rightsquigarrow \begin{array}{l} (A_i \to A_j) \notin S' \wedge (A_i \to A_k) \notin S' \\ (A_i \to \{A_j, A_k\}) \in S' \end{array} \tag{18}$$

The converted relation schema of the given example in Fig. 8 using the rule (18) is (a relation structure is given on the left side, the example instance on the right side):

$$\begin{array}{lr|l} table_M & \{M^{PK} : A, D\} & \{m : a, d\} \\ table_A & \{A^{PK} : M, B\} & \{a : m, b\} \\ table_{AD} & \{A^{PK}, D^{PK} : C\} & \{a, d : c\} \end{array} \tag{19}$$

Note the output schema may contain instances of trivial functional dependencies given by the rule (18), but these dependencies can be automatically decomposed by the rule

$$(A_i \to \{A_j, A_k\}) \in S' \rightsquigarrow (A_i \to A_j) \in S \wedge (A_i \to A_k) \in S \tag{20}$$

This rule enables sources with a known relation schema to be transformed into the model given in previous section.

2.2 RDF Format Document

The same principle as the (18) rule can be used for an RDF transformation (see [5]). Let A^L be the set of attributes being a left side of any functional dependency and $\mathscr{T}(A_i)$ be a bijection, which gives an unique tag name to each attribute in A.

$$A^L = \{A_i | (A_i \to A_j) \in S\} \tag{21}$$

Let be $\mathscr{I}_{A_i \to A_j}$ a set of all instances of the functional dependency $A_i \to A_j$ and a_i, respective a_j a value of given attribute in the instance. Now, let $\mathscr{R}$ be a set of tags $t(A_i, a_i)$ for each $A_i \in A^L, a_i \in \mathscr{D}_\alpha(A_i)$. The tag $t(A_i, a_i)$ name is <$\mathscr{T}(A_i)$> and the tag has a attribute @*rdf* : *about* set to the value a_i. The other attributes @$\mathscr{T}(A_j)$ of the tag $t(A_i, a_i)$ corresponds to all attributes $A_j | (A_i \to A_j) \in S$ and has value a_j according to the corresponding instance in $\mathscr{I}_{A_i \to A_j}$.

If A_i is a complex attribute, the values of all simple attribute of the A_i are inserted by the same way as previous ones. Note, there is no other standard feature to describe a complex attribute in the RDF (the last row in the fragment).

The fragment of the RDF document corresponding to Fig. 8. example is

```
<dse:M   rdf:about="m"   dse:A="a" dse:D="d" />
<dse:A   rdf:about="a"   dse:B="b" dse:M="m" />
<dse:A-D rdf:about="a-d" dse:A="a" dse:D="d" dse:C="c" />
```

The possible improvement of this way is to specify a value of tag attribute indirectly by a reference, for example instead of $@\mathscr{T}(A_j)$ to create new tag $<\mathscr{T}(A_j)>$ with attribute $@rdf : resource$ set to a_j.

2.3 OWL Format Document

There are many ways a structure can be represented in the OWL format [2, 4]. The proposed one maps a functional dependency system to the relationships between sets. The set $A_i^{a_i}$ represents an attribute-value pair (A_i, a_i) – the fact that $a_i \in \mathscr{D}_\alpha(A_i)$. If there exits the functional dependency $A_i \to A_j$ with instances $\mathscr{I}_{A_i \to A_j}$, all the relationships $A_i^{a_i} \sqsubseteq A_j^{a_j}$ are inserted into the ontology.

$$(A_i \to A_j) \in S \wedge (a_i \to a_j) \in \mathscr{I}_{A_i \to A_j} \rightsquigarrow A_i^{a_i} \sqsubseteq A_j^{a_j} \tag{22}$$

The idea is a unique table row identifier (represented by $\langle \mathfrak{m} \rangle$) activates all pairs of attributes, which depends on this row identifier attribute. Thanks to these activations, next pairs can be activated according to the functional dependency system and attribute values of thw row represented by $\langle \mathfrak{m} \rangle$. The $\langle \mathfrak{m} \rangle$ item is a member of all attribute-value pairs in the given row. It enables to reconstruct the row and also, when the pairs (A_i, a_i) are "generalised" to $(A_i, *)$, the functional dependency system.

This way keeps all the data structure description, which allows the inverse transformation into the functional dependency system by:

$$A_i^{a_i} \sqsubseteq A_j^{a_j} \rightsquigarrow \begin{array}{l} (A_i \to A_j) \in S \\ a_i \in \mathscr{D}_\alpha(A_i),\ a_j \in \mathscr{D}_\alpha(A_j) \wedge (a_i \to a_j) \in \mathscr{I}_{A_i \to A_j} \end{array} \tag{23}$$

The generated ontology using the rule (22) from the example on Fig. 8 is:

$$\begin{array}{lll} A^a \sqsubseteq & M^m & M^m \sqsubseteq \ A^a \\ D^d \sqsubseteq & M^m & B^b \sqsubseteq \ A^a \\ C^c \sqsubseteq & A^a \sqcap D^d & M^m \sqsubseteq \{\langle \mathfrak{m} \rangle\} \end{array} \tag{24}$$

The drawback of this solution is an indirect searching in the generated ontology. It may be particularly solved by the index generation by the rule

$$A_i :\ a_i \in \mathscr{D}_\alpha(A_i) \rightsquigarrow A_i^{a_i} \sqsubseteq \langle \mathfrak{a}_\mathfrak{i} \rangle \tag{25}$$

These "index items" occur, that the inverse rule (23) can not be used, because there does not exist the set $A_i^{a_i}$ corresponding to the $\langle \mathfrak{a}_\mathfrak{i} \rangle$. It may be solved by the insertion of the special set V which all $\langle \mathfrak{a}_\mathfrak{i} \rangle$ are members and following inhibition of this set consideration during the reconstruction process.

3 Conclusion

The paper dealt with a data structure estimation method, its properties and the complexity reduction due to the model skeleton usage. The method has a non-polynomial complexity due to a binomial complexity part (13) determining complex attribute non-trivial functional dependencies. The used model skeleton notation may reduce average time for the input relation decomposition process. Finally, the features of the estimated model were given as well.

The second part introduced the method usage as a service for a semantic web transforming current web sources and several ways were detailly mentioned.

The `RDF format` transformation works on similar principle as methods generating a schema from known functional dependency set. The data stored in a repository can be transformed into a RDF document by given algorithm, but there is generally no support to reconstruct a functional dependency system from the document because of no way how directly express complex attributes and their values (it is not known, which attributes depend on the key given by the $@rdf : about$ value and which ones are decomposition of the complex attribute).

Alternatively, the `OWL format` uses the attribute-value pairs and relationships between pairs according to the estimated functional dependency system. The ontology generation rule (22) keeps all information needed to the direct functional dependency system reconstruction – the rule (23), where are considered generalised pairs for attributes. The improvement for a retrieval process is mentioned – the rule (25), but it disables a direct reconstruction in general case.

The OWL format thanks to the complementary rules (22/23) is preferred (keeping all structure information, the direct ontology transformation back, the extended version with (25) allowing a direct accessing values).

The presented work aim was to provide a connection tool between current web sources and semantic web tools. It enables these sources to be processed by the same tools as semantic web ones with the best metadata available – with the estimated data structure. These data can be stored in the knowledge base, be published in any semantic web format and may be used for exploration of other web sources by an incremental knowledge portal building. A continuous exploration may lead to the data structure estimation to be more precise.
The future work will be oriented towards combining sources using different ways to express attribute names or values and related primitive value matching issues.

References

1. A.A.Barfourosh, M.L. Anderson, H.R.M.Nezbad, D Perlis. "Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition". http://citeseer.ist.psu.edu/barfourosh02information.html [online]. 2002.
2. Grigoris Antoniou, Frank van Harmelen. "A Semantic Web Primer". MIT Press, 2004. ISBN: 0-262-01210-3.
3. Eric Miller, Ralph Swick, Dan Brickley. "Resource Description Framework". `<http://www.w3.org/RDF/>` [on-line]. 2004.

4. Eric Miller, Jim Hendler. "Web Ontology Language". <http://www.w3.org/2004/OWL/> [on-line]. 2005.
5. T.B. Lee "Relational Databases on the Semantic Web". <http://www.w3.org/DesignIssues/RDB-RDF.html> [on-line]. 1998.
6. S.M.Kuck, Y. Sagiv. "A Universal Relation Database System Implemented Via the Network Model" In *Symp. on Principles of Database Systems*, pp.147-157. 1982.
7. D.Bednarek, D.Obdrzalek, J.Yaghob, F. Zavoral. "Access Rights Definition and Management in an Information System based on a DataPile Structure". In *ITAT 2004, Workshop on Information Technologies, Application and Theory*, 2004.
8. M. Rimnac. "Odhadovani struktury dat a induktivni logicke programovani". In *ITAT 2005, Workshop on Information Technologies, Appl. and Theory*, 2005.
9. V. Marik, O. Stepankova. "Umela Inteligence 1". Academia. ISBN 80-200-0496-3.
10. G. Ausiello, A. D'Atri, M. Moscarini. "Chordality Properties on Graphs and Minimal Conceptual Connections in Semantic Data Models". In *Symposium on Principles of Database Systems*, pp. 164–170. 1985.
11. B.T. Messmer, H.Bunke. "Efficient Subgraph Isomorphism Detection: A Decomposition Approach". In *IEEE Transactions on Knowledge and Data Engineering.* pp. 307-323. 2000.
12. G. Ausiello, A.D'Atri, D.Secca "Graph Algorithms for Functional Dependency Manipulation". In *Journal of ACM.* Volume 30. Issue 4. pp. 752-766. 1983. ISSN: 0004-5411.
13. J. Biskup, U. Dayal, P.A. Bernstein. "Synthesising Independent Database Schemas". In *SigMod*, pp. 143–150, 1979.
14. P.A. Bernstein, J.R. Swenson, D.C. Tsichristzis. "A Unified Approach to Functional Dependencies and Relations". In *SigMod*, pp. 237–245, 1975.
15. F. Cuppens, K. Yazdanian. "A Natural Decomposition of Multi-level Relations". In *IEEE Symposium on Security and Privacy*, pp. 273-284. 1992.
16. G. Grahme, K. Räihä. "Database Decomposition into Fourth Normal Form". In *Conference on Very Large Databases*, pp. 186–196, 1983.
17. P.A.Flach, I.Savnik. "Database Dependency Discovery: A Machine Learnig Approach". In *AI Communications*, Volume 12/3, pp. 139–160. 1999.
18. H. Mannila, K.J. Räihä "Dependency Inference". In *Proc. of VLDB.* pp. 155–158. ISBN: 0-934613-46-X. 1987.
19. J. Kivinen, H. Mannila "Approximate Inference of Functional Depentencies from Relations". In *Proc. of 4. int. conf. on Database Theory*, Berlin, Germany. pp. 129–149. ISSN: 0304-3975. 1995.
20. C. Beeri, P.A.Bernstein. "Computational Problems Related to the Design of Normal Form Relation Schemes". In *ACM Trans. on Db. Sys..* 4,1. pp 30-59. 1979.
21. D. Heckerman, D.M. Chickering, Ch. Meek, R. Rounthwaite, C. Kadie. "Dependency Networks for Inference, Collaborative Filtering and Data Visualization". In *Journal of Machine Learning Research*, Volume 1, pp. 49–75. 2001. ISSN 1533-7928.
22. B.N. Shamkant, R. Minyoung. "Vertical Partitioning for Database Design – a Graphical Algorithm". In *SigMod*, pp. 440–450, 1989.
23. L. Bellatreche, K. Karlapalem, A. Simonet. "Algorithms and Support for Horizontal Class Partitioning in Object–Oriented Databases". In *Distributed and Parallel Databases, 8*, Kluwer Academic Publisher. pp. 115–179, 2000.
24. D. Rosaci, G. Terracina, D. Ursino. "A Framework for Abstracting Data Sources Having Heterogennous Representation Formats". In *Data & Knowledge Engineering*, vol. 48, pp. 1–38. 2004.
25. M. Golfarelli, S. Rizzi, B. Vrdoljak "Data Warehouse Design from XML Sources". In *Proc of Data Warehousing and OLAP*, pp. 40–47. ISBN: 1-58113-437-1. 2001.