Toward the Semantic Web – An Approach to Reverse Engineering of Relational Databases to Ontologies

Irina Astrova

Tallinn University of Technology, Ehitajate tee 5,
19086 Tallinn, Estonia
irinaastrova@yahoo.com

Abstract. We propose a novel approach to reverse engineering of relational databases to ontologies. Our approach incorporates two main sources of semantics: HTML pages and a relational schema. This incorporation results in that: (1) only minimal information about a relational database is required to build an ontology; and (2) the ontology is no longer "impaired" by bad-database design, and by optimization and de-normalization of the relational schema. Our approach can be used for migrating HTML pages (especially those that are dynamically generated from a relational database) to the ontology-based Semantic Web. The main reason for this migration is to make the relational database information that is available on the Web machine-processable.

1 Introduction

One of the main driving forces for the Semantic Web has always been the expression, on the Web, of the vast amount of relational database information in a way that can be processed by machines [1]. Indeed, most information on the Web is not machine-processable, because it is often represented in HTML (Hypertext Markup Language) [2]. This language describes how the information looks like and not what it is. In order for machines to process the information, it must be represented in an ontology language – e.g. Frame Logic (F-Logic) [3] – and linked to ontologies. An ontology can be used for annotating HTML pages with semantics.

Manual or semi-automatic semantic annotation [4] is time-consuming, subjective and error-prone. It is even impossible on scale of the Web that contains billions of pages. Most pages even do not exist until they are dynamically generated from relational databases at the time of submitting HTML forms.

An alternative to the semantic annotation is automatic or semi-automatic reverse engineering of relational databases to ontologies [5]. However, because of the novelty of that area, there are few approaches that consider an ontology as the target for reverse engineering. A majority of the work has been done on extracting a conceptual schema such as an entity-relationship model from relational databases.

2 Common Problems of Reverse Engineering

At first glance, it seems easy to reverse engineer a relational database to an ontology: just map each relation to a class, each attribute in the relation to an attribute in the class, each tuple to an instance, and each constraint to an axiom [6]. This provides simple and fully automatic (i.e. without user interaction) reverse engineering. So, why would not we want to do this? The easy approach ignores common problems of reverse engineering; e.g.:

- *Optimization and de-normalization:* A relational schema is often optimized and de-normalized for performance reasons [7].
- *Unrealistic assumptions:* Many organizations believe in keeping all their data in third normal form. However, every database designer has war stories about finding the entire relational schema in first normal form instead of third normal form [7].
- *Bad database design:* A relational schema is often bad-designed, because it may be done by novice and untrained database designers who are not familiar with database theory and database methodology [8, 9].
- *Non-translated constructs:* Since a relational schema does not support all constructs of a conceptual schema, some of the semantics captured in the conceptual schema – e.g. inheritance – will necessarily be lost when translating the schema from conceptual to relational. Indeed, this translation usually results in "semantic degradation" of the schema that becomes simpler, less complete, less understandable, and less expressive [10].
- *Implicit semantics:* Semantics may be not in a relational schema, but rather in data or even in the heads of users who query a relational database [10].
- *Meaningless names:* Relations and attributes in a relational schema are often assigned names that are a maze of cryptic abbreviations; e.g. `YRTREBUT`, `B_423_SPD` or `FRED` [7]. However, it is difficult or even impossible to deduce the meaning (i.e. semantics) of data from those names.

3 Related Work

Existing approaches to reverse engineering of relational databases to ontologies fall roughly into one of the three categories:

- *Approaches based on an analysis of relational schema:* E.g. Stojanovic et al's approach [5] provides a set of rules for mapping constructs in the relational database (i.e. relations, attributes, tuples, and constraints) to semantically equivalent constructs in the ontology (i.e. classes, attributes, instances, and axioms). These rules are based on an analysis of relations, attributes, primary and foreign keys, and inclusion dependencies.
- *Approaches based on an analysis of data:* E.g. Astrova's approach [11] builds an ontology based on an analysis of relational schema. However, since a relational schema often captures little explicit semantics [12], this approach also analyzes data in the relational database.

– *Approaches based on an analysis of user queries:* E.g. Kashyap's approach [13] builds an ontology based on an analysis of relational schema; the ontology is then refined by user queries. However, this approach does not create axioms that are part of the ontology.

Not all of the common problems of reverse engineering can be solved using the existing approaches. In particular, the existing approaches can be limited in terms of requiring more input information than it is possible to provide in practice and making unrealistic assumptions about the input. E.g. they typically assume that a relational schema is in third normal form.

The search for a solution leads us to a novel approach where HTML pages are analyzed. So far, this analysis has been focused on generation of *wrappers*; see e.g. [14, 15, 16, 17]. A wrapper is a program that extracts the relational database information from HTML pages. There are wrappers that are based on ontologies; see e.g. [18].

Wrappers have the main advantage of reconstructing a (part of) relational database "hidden" behind HTML forms, when a relational schema is unknown. The backside of this advantage is that any changes to structures of HTML pages – e.g. adding or deleting fields in the pages – can break the wrappers and thus, the ontologies they are based on. HTML pages are volatile by nature, meaning that they are often redesigned [19] – typically more than twice a year [20].

The biggest problem of wrappers is that they rely on structures of HTML pages to extract semantics, thus throwing away all the advantages of analyzing a relational schema; e.g.:

– An analysis of HTML pages often leads to a "brittle" ontology. Since a relational schema is more stable than HTML pages, its analysis guarantees that the ontology need not be rebuilt every time the pages change their structures.
– HTML pages represent views of the relational database; i.e. different ways of viewing the relational database information on the Web. Thus, some semantics may be not in the pages, but rather in the relational schema.

Apart from these, the relational schema is a formal explicit agreement between database designers and users about data and its meaning in an organization. Thus, the relational schema provides an important source of semantics to be extracted into an ontology [25].

4 Our Approach

As an attempt to solve the common problems of reverse engineering, we propose a novel approach. Our approach is *based on an analysis of HTML pages*. There are two main reasons for this analysis. One is that a relational schema often captures little explicit semantics [12], while a conceptual schema is usually unavailable or out-of-date [10]. Another reason for analyzing HTML pages is to benefit from their user-friendliness. This user-friendliness results in that:

– HTML pages partially represent a logical structure of the relational database, rather than its physical structure (i.e. a relational schema). Indeed, they often provide a user-friendly interface to the relational database. Behind this interface, a relational schema can be bad-designed, optimized, and de-normalized.
– Table and field names in HTML pages are often more explicit and more meaningful than the corresponding relation and attribute names in a relational schema.

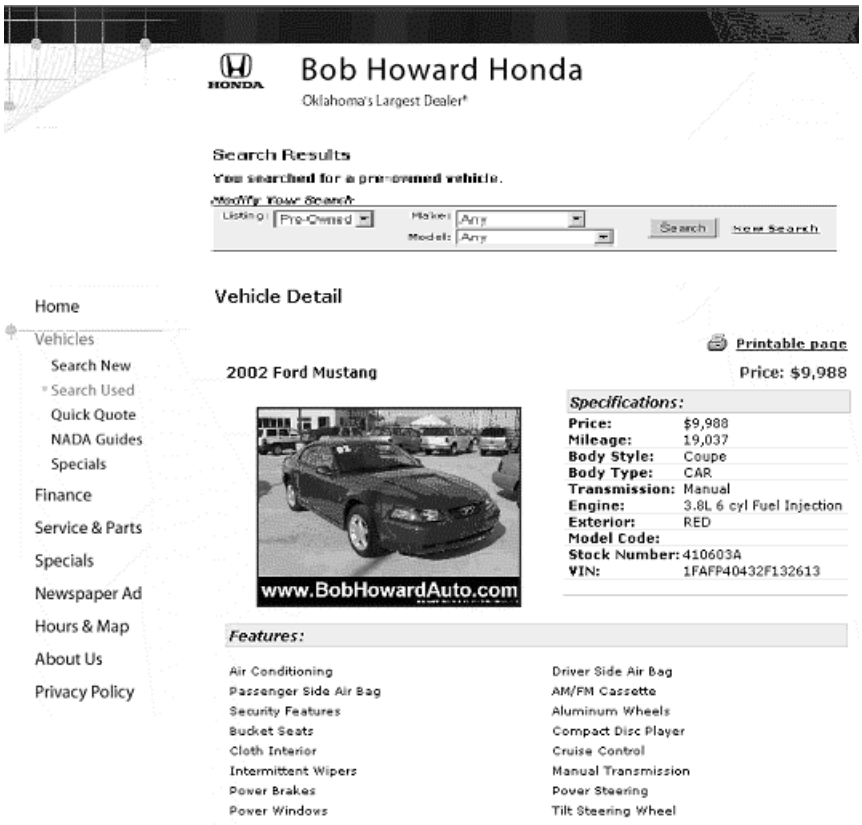

Figure 1. HTML page

Given the reasons for analyzing HTML pages, let's now consider more precisely what our approach is and then illustrate it by example. Suppose going to a website `http://www.bobhowardhonda.com` and searching for information about a used vehicle; e.g. Ford Mustang. Since such information is stored in a relational database, we fill out an HTML form in Figure 1 (located in the upper frame of the page) and submit it. After submitting the form, search results will be returned in an HTML page in Figure 1. This page is dynamically generated from a relational database and contains specifications of Ford Mustang and its features.

4.1 Extracting Form Model Schema

The first step of our approach is extracting a form model schema [10]. This schema was originally proposed to extract an entity-relationship model from database forms. Basically, the form model schema contains:

- *Form field:* This is an aggregation of *name* and *entry* associated to it[1]. A name is pre-displayed and serves as a clue to what will be entered by users or displayed by HTML pages. An entry is the actual data; it roughly corresponds to an attribute in the relational schema. We use the term of *linked attribute* for such an entry to distinguish it from other entries that are computed or simply unlinked with the relational schema.
- *Structural unit:* This is a logical group of closely related form fields. It roughly corresponds to a relation in the relational schema.
- *Relationship:* This is a connection between structural units that relates one structural unit to another (or back to itself). There are two kinds of relationship: *association* and *inheritance*.
- *Constraint:* This is a rule that defines what data is valid for a given linked attribute. A *cardinality constraint* specifies for an association relationship the number of instances that a structural unit can participate in.
- *Underlying source:* This is a physical structure of the relational database (i.e. a relational schema) that defines relations and attributes with their data types.
- *Form type:* This is a collection of empty form fields.
- *Form template:* This is a particular representation of form type. Each form template has a layout (i.e. its graphical representation) and a title that provides its general description.
- *Form instance:* This is an occurrence of form type, when its template is filled in with the actual data. E.g. Figure 1 is an instance of the form type.
- *Hierarchical tree:* This is a hierarchical structure of form instance. There are two kinds of hierarchical tree: *structured data tree* and *content tree* [21]. A structured data tree captures, for an HTML page, the hierarchy of HTML tags and data contents (i.e. the syntactic hierarchy). A content tree is the same, except that HTML tags are deleted. Thus, it captures only the hierarchy of data contents (i.e. the intended hierarchy).

4.1.1 Analysis of HTML Pages Structures and Relational Schema

Extracting a form model schema consists in an analysis of HTML pages (especially their structures) and a relational schema to identify constructs of the form model schema and to assign those constructs names using wrapper generation techniques [14, 15, 16, 17, 18].

4.1.1.1 Identifying Form Instances

[1] However, we can also identify a form field with no name for its entry; e.g. the photo, year, make, and model in Figure 1.

HTML pages typically contain advertisements and navigational menus that can be viewed as "noisy" data [17]. Thus, given an HTML page, the first task is to identify a data-rich section (i.e. a form instance). We can do this in three ways.

First, we can compare HTML pages for overlaps in structure [17]. The implication is that all pages from a given website will organize their content in a similar way, regarding the location of advertisements and navigational menus.

Second, we can examine HTML code for block tags such as `<frame>` [18]. A difficulty is that HTML pages often consist of multiple frames.

Third, we can search through all frames in the page to find the largest one [14]. This approach typically implies that a frame that takes up the largest display area will be the most interesting to users.

E.g. from Figure 1 we would identify that `Vehicle Detail` represents all data that is the subject of interest to users.

4.1.1.2 Identifying Structural Units

We can take three basic approaches to this. First, we can examine HTML code for block tags such as `<table>`, `<ul>` and `<ol>` [17, 18]. The implication is that structural units will be represented by tables or lists in HTML pages. The biggest problem with this approach is that it relies on a physical structure of HTML pages. Thus, it fails if the pages change their structures frequently. There are many other situations, where the approach does not work either such as errors in the code and misuse of the block tags [14]. E.g. not only is `<table>` used for representing a relation in the relational schema, but it is also the primary method for grouping data in HTML pages [15]. The data is often grouped just for easier viewing it by users.

Second, we can use visual cues to determine a logical structure of HTML pages – that is, the real meaning of the pages as they are understood by users [14, 16]. E.g. the users might consider the year, make, model, price, mileage, …, and vin in Figure 1 as a whole group, just because they all are specifications.

Third, we can look for structural units in relations of the relational schema (i.e. the underlying source).

E.g. from Figure 1 we would identify two structural units: `Vehicle` and `Feature`. One contains specifications for a used vehicle (`Year`, `Make`, `Model`, `Price`, `Mileage`, …, and `VIN`); while another structural unit lists the vehicle features (`Air Conditioning`, `Passenger Side Air Bag`, …, and `Tilt Steering Wheel`).

4.1.1.3 Identifying Linked Attributes

We can take three basic approaches to this. First, we can examine HTML code for block tags such as `<thead>` and `<th>` [18]. Again, this approach works as long as the code is well designed, correct, and stable. Moreover, the approach is viable only if fields in HTML pages are separated with the block tags; it does not work for merged data. E.g. the year, make, and model in Figure 1 are all merged data, meaning that they are combined in a single text string: "`2002 Ford Mustang`".

Form model schema	Ontology
-- Structural units Structural-Units ::= { Feature(**-- Linked attributes** name : VARCHAR) Vehicle(**-- Linked attributes** year : INTEGER, make : VARCHAR, model : VARCHAR, price : FLOAT, mileage : FLOAT, … vin : VARCHAR)} **-- Relationships** Relationships ::= { Has(Vehicle, Feature)} **-- Constraints** NotNull(Vehicle, mileage) Cardinality (Vehicle, Feature, 1, n) **-- Underlying source** Underlying-Source ::= { **-- Relations** Detail(**-- Attributes** name : VARCHAR) Vehicle (**-- Attributes** year : INTEGER, make : VARCHAR, model : VARCHAR, price : FLOAT, mileage : FLOAT, … vin : VARCHAR)}	**// Classes** Feature::Object[**// Attributes** name =>> String]. Vehicle::Object[**// Attributes** year =>> Integer, make =>> String, model =>> String, price =>> Float, mileage =>> Float, … vin =>> String, **// Relationships** features =>> {Feature}]. **// Axioms** NotNull(Vehicle, mileage). Forall C,A NotNull(C, A) <- Forall IC Exists IA IC:C And IC[A ->> IA]. **// Instances** f1:Feature[**// Attributes** name ->> "Air Conditioning"]. f2:Feature[**// Attributes** name ->> "Passenger Side Air Bag"]. … fn:Feature[**// Attributes** name ->> "Tilt Steering Wheel"]. v:Vehicle[**// Attributes** year ->> 2002, make ->> "Ford", model ->> "Mustang", price ->> 9988, mileage ->> 19037, … **// Relationships** features ->> {f1, f2, …, fn}].

Figure 2. Summary of reverse engineering

Second, we can use visual cues [14, 17]. This approach typically implies that there will be some separators (e.g. blank areas) that help users split the merged data. E.g. the year, make, and model in Figure 1 are space-separated. Sometimes we can also use data formats as visual cues to understand the meaning of data. E.g. the price in Figure 1 is also indicated with a dollar sign (i.e. "$").

Third, we can look for linked attributes in attributes of the relational schema. This is because a given HTML page may contain only a part of the total attributes in the relational schema.

Looking at a form model schema in Figure 2, we can see that each structural unit is defined by a set of linked attributes. E.g. the structural unit `Vehicle` contains linked attributes definitions for `year`, `make`, `model`, `price`, `mileage`, ..., and `vin`; while the structural unit `Feature` has a linked attribute `name`.

4.1.1.4 Identifying Relationships

We can take two basic approaches to this. First, we can look for relationships (usually many-to-many) in relations of the relational schema, then look for relationships (one-to-one and one-to-many) in foreign keys. A difficulty is that there are always relations with unknown foreign keys [8].

Second, since the relational database information typically does not reside on a single HTML page, we can try to find relationships in hyperlinks.

E.g. from Figure 1 we would identify an association relationship between the structural units `Vehicle` and `Feature`: a used vehicle has features. The implication is that related structural units will appear at the same page.

4.1.1.5 Naming Structural Units

Structural units can be given names of the corresponding relations in the relational schema. But it is generally less confusing to users if the names are more meaningful. Looking back at the form model schema in Figure 2, notice the adaptation of the name `Feature` to the structural unit. This can better convey the meaning of data than the original relation name `Detail` would.

4.1.1.6 Naming Linked Attributes

There are three basic approaches to this. One is to give linked attributes names of the corresponding attributes in the relational schema.

Since field names in HTML pages are often more explicit and more meaningful than the corresponding attribute names in the relational schema, another approach is to give linked attributes the field names. A difficulty is that the field names are not always encoded in HTML pages. E.g. the photo, year, make, and model in Figure 1 are given no names at all. However, missing names can be found in HTML forms. Since the forms are often used for querying a relational database, they provide a sketch (of part) of a relational schema [17]. Assuming that a given website will do its best to return the most relevant data to users, search criteria submitted through an HTML form are likely to re-appear in the returned HTML pages. E.g. from an HTML form in Figure 1, we could enter "`Ford`" and "`Mustang`" for fields `Make` and `Model`, respectively. Search results for the form will be returned in an HTML page in Figure 1. This contains details of a used vehicle that matches the search criteria; i.e. Ford Mustang. Therefore, linked attributes corresponding to the fields, with "`Ford`" and "`Mustang`" re-appeared in their entries, could be named `make` and `model`, respectively.

Yet another approach is to give linked attributes data type names [17]. E.g. a linked attribute represented by the photo in Figure 1 might be named `image`.

4.1.1.7 Naming Relationships

Relationships can be given names that are either names of the corresponding relations (usually for many-to-many relationships) or foreign key names (for one-to-one or one-to-many relationships). Again, users can give more meaningful names to the relationships.

The end result for the first step is the form model schema in Figure 2.

4.1.2 Data Analysis

In addition to the structures of HTML pages, we also analyze data in the pages to identify constraints. A data analysis includes a strategy of learning by examples, borrowed from machine learning techniques [22, 23]. In particular, it is performed as a sequence of learning tasks from the relational database. Each task is defined by: (1) *task relevant data* (e.g. data contained in the pages), (2) *problem background knowledge* (e.g. application domain knowledge), and (3) *expected representation of results of learning tasks* (e.g. first order predicate logic). The results of learning tasks are related to a current state of the relational database. They will be generalized into knowledge about all states through an induction process [10]. This process combines the semantics extracted from the pages with the application domain knowledge that is provided by users (i.e. the user "head knowledge"). Such knowledge controls the learning tasks to come to the best inductive conclusion, the conclusion that will be consistent with all states of the relational database.

E.g. from Figure 1 we would identify a constraint `NotNull` on the linked attribute `mileage`. This contains non-null values for any used vehicle.

4.1.3 Integration

There are typically several HTML pages (of different structures) for any given website. Thus, their analysis will produce several form model schemata. These will be merged into a single one through an integration process [10]. This process performs as follows. First, the schemata are compared for overlaps in structure. This means looking for structural units and relationships with similar names, then looking for similar structures within structural units and relationships. Second, the schemata are compared for overlaps in meaning. This means looking for structural units that correspond to the same real-world objects but have different names. Third, naming conflicts (i.e. synonyms and homonyms) are resolved. Conflicts can also be in different constraints on the linked attributes and different cardinality constraints on the relationships. By performing these tasks, the integration process makes the schemata consistent with one another and brings them together into a single one that makes sense for all HTML pages from a given website.

4.2 Schema Transformation

The second step of our approach is transforming the form model schema into an ontology (i.e. "schema transformation"). Basically, this means replacing constructs of the form model schema to constructs of the ontology using mapping rules [24].

The ontology is formulated in F-Logic. This language has an object-oriented syntax. It provides support for classes, attributes with domain and range definitions, inheritance hierarchies of classes and attributes, and axioms that can be used for further characterizing relationships between instances.

Continuing the example, consider again a form model schema in Figure 2. Here schema transformation is straightforward. First, we create a class for each structural unit in the form model schema. E.g. we create two classes: `Vehicle` and `Feature`. Within each class, we create an attribute for each linked attribute in the structural unit. E.g. for the class `Vehicle`, we add attributes `year`, `make`, `model`, `price`, `mileage`, ..., and `vin`. We also add an attribute `features`. This associates the two classes. Finally, we create an axiom for each constraint (except cardinalities) in the form model schema. E.g. we add an axiom `NotNull` to the ontology.

The end result for the second step is the ontology in Figure 2. The ontology is nearing completion. But there are still instances to create. These instances will populate a knowledge base, whose schema is defined by the ontology [12].

4.3 Data Migration

The third step of our approach is creating instances from data contained in HTML pages (i.e. "data migration"). Basically, this means assigning values to the attributes in the ontology using wrapper generation techniques [17, 18].

Continuing the example, consider again an HTML page in Figure 1. Here data migration is easy for the attributes `year`, `make`, `model`, `price`, `mileage`, ..., and `vin` in the class `Vehicle`. However, we meet with a difficulty when trying to find a value for the attribute `features` that corresponds to the list of features in Figure 1. We overcome this difficulty by creating an instance for each feature in Figure 1 and assigning it to the attribute `features`.

The end result for the third step is the ontology in Figure 2.

5 Conclusion

We have proposed a novel approach to reverse engineering of relational databases to ontologies. Our approach is based on the idea that semantics of data in a relational database can be extracted by analyzing HTML pages. These semantics are supplemented with those captured in the relational schema to build an ontology.

There are three important advantages of our approach:

- *It requires minimal information about a relational database.* This is important because the complete knowledge of the relational database is usually unavailable [9]. E.g. there are always relations with unknown primary keys [8].
- *It makes no assumptions about a relational schema that can be bad-designed, optimized, and de-normalized.* This is important because even database experts may occasionally break the rules of good database design [8]. And many database

designers improve performance by optimizing and de-normalizing the relational schema [7].
- *It appeals to users who likely understand HTML pages better than a relational schema.* This is important because reverse engineering cannot be completely automated [5]. There are always situations where user interaction is necessary.

These advantages come in large part from an analysis of HTML pages. But this analysis has costs. One is the difficulty in automation [18]. This is because HTML pages are designed for (human) users use only. E.g. data in the pages can be embedded in natural language text or hidden within graphical presentation primitives [19].

6 Future Work

In the future, our approach can be used for migrating HTML pages (especially those that are dynamically generated from a relational database) to the ontology-based Semantic Web. The main reason for this migration is to make the relational database information that is available on the Web machine-processable [5].

Acknowledgement

This research is partly sponsored by ESF (Estonian Science Foundation) under the grant nr. 5766.

References

1. Berners-Lee, T.: Relational Databases on the Semantic Web (2002) http://www.w3.org/DesignIssues/RDB-RDF.html (2002)
2. Raggett, D.: HTML 4.01 Specification, http://www.w3.org/TR/html401/ (1999)
3. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-oriented and Frame-based Languages, Journal ACM, No. 42 (1995) 741–843
4. Erdmann, M., Maedche, A., Schnurr, H., Staab, S.: From Manual to Semi-automatic Semantic Annotation: About Ontology-based Text Annotation Tools, Linköping Electronic Articles in Computer and Information Science Journal (ETAI), Vol. 6, No. 2 (2001)
5. Stojanovic, L., Stojanovic, N., Volz, R.: Migrating Data-intensive Web Sites into the Semantic Web, In: Proceedings of the 17th ACM Symposium on Applied Computing (SAC) (2002) 1100–1107
6. Dogan, G., Islamaj, R.: Importing Relational Databases into the Semantic Web (2002) http://www.mindswap.org/webai/2002/fall/Importing_20Relational_20Databases_20into_20the_20Semantic_20Web.html
7. Muller, R.: Database Design for Smarties: Using UML for Data Modeling, Morgan Kaufmann (1999)
8. Premerlani, W., Blaha, M.: An Approach for Reverse Engineering of Relational Databases, Communications of the ACM, Vol. 37, No. 5 (1994) 42–49

9. Hainaut, J., Henrard, J., Hick, J., Roland, D., Englebert, V.: Database Design Recovery, In: Proceedings of the 8th Conference on Advanced Information Systems Engineering (CAiSE), LNCS, Vol. 1080 (1996) 272–300
10. Mfourga, N.: Extracting Entity-Relationship Schemas from Relational Databases: A Form-driven Approach, In: Proceedings of the 4th Working Conference on Reverse Engineering (WCRE) (1997) 184–193
11. Astrova, I.: Reverse Engineering of Relational Databases to Ontologies, In: Proceedings of the 1st European Semantic Web Symposium (ESWS), LNCS Vol. 3053 (2004) 327–341
12. Noy, N., Klein, M.: Ontology Evolution: Not the Same as Schema Evolution, Knowledge and Information Systems, Vol. 6, No. 5 (2003)
13. Kashyap, V.: Design and Creation of Ontologies for Environmental Information Retrieval, In: Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW) (1999)
14. Yang, Y., Zhang, H.: HTML Page Analysis Based on Visual Cues, In: Proceedings of the 6th International Conference on Document Analysis and Recognition (ICDAR) (2001) 859–864
15. Wang, J., Hu, J.: A Machine Learning Based Approach for Table Detection on the Web, In: Proceedings of the 11th International Conference on World Wide Web (WWW) (2002) 242–250
16. Gu, X.-D., Chen, J., Ma, W-Y., Chen. G.-L.: Visual Based Content Understanding towards Web Adaptation, In: Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH) (2002) 29–31
17. Wang, J., Lochovsky, F.: Data Extraction and Label Assignment for Web Databases, In: Proceedings of 12th International Conference on World Wide Web (WWW) (2003) 187–196
18. Embley, D.: Toward Semantic Understanding – An Approach Based on Information Extraction, In: Proceedings of the 15th Australasian Database Conference (ADC) (2004) 3–12
19. Florescu, D., Levy, A., Mendelzon, A.: Database Techniques for the World Wide Web: A Survey, ACM SIGMOD Record, Vol. 27, No. 3 (1998) 59–74
20. Knoblock, C., Kambhampati, S.: Information Integration on the Web (2002) http://rakaposhi.eas.asu.edu/aaai-i3-tut-all.pdf
21. Lim, S., Ng, Y.: Extracting Structures of HTML Documents, In: Proceedings of the 12th International Conference on Information Networking (ICOIN) (1998) 420–426
22. Paredis, J.: Learning the Behavior of Dynamical Systems from Examples, In: Proceedings of the 6th International Workshop on Machine Learning (ICML) (1989) 137–140
23. Michalski, R.: A Theory and Methodology of Inductive Learning, Machine Learning: An Intelligence Approach, Vol. 1 (1983) 83–134
24. Astrova, I., Stantic, B.: Reverse Engineering of Relational Databases to Ontologies: An Approach Based on an Analysis of HTML Forms, In: Proceedings of the Workshop W6 on Knowledge Discovery and Ontologies (KDO), 15th European Conference on Machine Learning (ECML), 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), eds. Buitelaar, P. et al. (2004) 73–78
25. Meersman, R.: Ontologies and Databases: More than a Fleeting Resemblance. In: Proceedings of the International Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations (OES/SEO), eds. d'Atri, A., Missikoff, M. (2001)