Journal of Web Engineering, Vol. 6, No. 2 (2007) 143–164

ONTOLOGY DEVELOPMENT FOR THE SEMANTIC WEB: AN HTML FORM-BASED REVERSE ENGINEERING APPROACH

SIDI MOHAMED BENSLIMANE
University of Claude Bernard, Lyon, France
sidi-mohamed.benslimane@liris.cnrs.fr

DJAMAL BENSLIMANE
University of Claude Bernard, Lyon, France
Djamal.benslimane@liris.cnrs.fr

MIMOUN MALKI
University of Sidi Bel-Abbes, Sidi Bel-Abbes, Algeria
malki@univ-sba.dz

ZAKARIA MAAMAR
Zayed University, Dubai, U.A.E
zakaria.maamar@zu.ac.ae

PHILIPPE THIRAN
University of Namur, Namur, Belgium
philippe.thiran@fundp.ac.be

YOUSSEF AMGHAR
INSA, Lyon, France
youssef.amghar@liris.cnrs.fr

MOHAND-SAID HACID
University of Claude Bernard, Lyon, France
mohand-said.hacid@liris.cnrs.fr

Received April 18, 2006
Revised November 2, 2006

The rapid growth of the Internet makes information available anywhere and anytime. Most businesses run Web-based front-end databases upon which online services are offered to end-users. The next generation of the Web, the semantic Web, seeks to offer data in a usable form for automatic reasoning. To this purpose, it is necessary to make existing database content ready-to-use for semantic Web applications, which use ontologies to formally define the semantics of their data. As a result, a large number of initiatives focus on building ontologies through automatic or semi-automatic processes. In this paper we present a semi-automatic reverse engineering approach that uses a relational database's HTML forms and a set of transformation rules to produce to an OWL ontology.

Keywords: Ontology extraction, Relational database, HTML form, Reverse engineering.
Communicated by: G.-J. Houben & K. Turowski

1 Introduction

During the last few years, the ease-of-use of databases has considerably grown in all businesses. In addition, the simplicity and proliferation of the World Wide Web have made the

information available anywhere and anytime. As a result, most businesses run Web-based front-end databases upon which a variety of online services. The next generation of the Web, the semantic Web, seeks to offer data in a usable form for automatic processing. A key ingredient to this is to enrich databases with semantics. Lately, ontologies have become the focus of research in several areas including knowledge engineering and management, information retrieval and integration, agent systems, the semantic Web, etc. The availability and proliferation of ontologies are crucial to the success of the semantic Web. Nevertheless building ontologies is so costly that the progress of the semantic Web enhancement can get refrained. Manual construction of ontologies [1, 2] still remains tedious, time-consuming, and error-prone. Automatic building of ontologies from existing information sources [3] is relevant and fully automated tools are still at the very early stage of development. Therefore, the use of a semi-automatic technique for ontology building is deemed appropriate.

Relational databases continue to be the most popular means for storing, retrieving, and manipulating data. Similarities do exist between relational and ontological models with respect to abstracting and modeling the domain of discourse. But, their purposes are different. In general, the incompatibilities between relational and ontological models can be summarized as follows:

- ***Design***: an ontology is by default application independent domain. Its concepts can be reused by different kinds of applications. A relational model represents the structure and integrity of the data elements of a specific enterprise application. Therefore, the conceptualization and vocabulary of a relational schema are not a priori intended to be used by independent users.
- ***Persistence***: the semantics of a relational schema is often an arrangement between what users want and what developers can do. In addition, a relational schema would be updated on the fly as particular new functional requirements emerge. Changes in the domain and in conceptualization may also require evolution in the content of ontologies. Nevertheless, changes occur more frequently in database schema than in ontologies.
- ***Usage***: ontology supports browsing concepts so reasoning task could be performed [4]. So, ontology is used to reason about concepts while relational schema is used to retrieve a collection of instance data.
- ***Model elements***: the differences are as follows:
  - There are no basic types; everything is a concept;
  - There are no attributes or relationships; everything is a property. Only binary relationships can be represented through properties. Ternary or higher degree relationships get their own concepts;
  - Concepts and properties can be organized into inheritance hierarchies.

Reverse engineering technique appears to be interesting solution for building ontologies to be fed with information contained in relational databases. This technique is a process that analyzes a "legacy" system so its components and their relationships are revealed [5]. However, building ontology on the basis of the information that is extracted from a relational schema faces the following obstacles:

- To improve performance, many database designers tend to break the best practices of database design by optimizing and de-normalizing the relational schema.

- Many organizations typically assume that a relational schema is in third normal form, which is not always the case.

- The complete information about the relational database such as functional and inclusion dependencies is usually not available [6].

- Since a relational schema does not support specific constructs that are in a conceptual schema like inheritance, these constructs are lost during the mapping of the conceptual schema onto the relational schema. This results in a "semantic degradation" of the relational schema that becomes simple, less complete, less understandable, and less expressive [7].

- Relation and attribute names in a relational schema are in general selected based on designers understanding and background. Moreover, these names are often shortened *(e.g., CUST_NB, StuName, S_125_AZE, etc.)*, which makes deducting the meaning of data a big challenge [8].

To address some of the aforementioned obstacles of reversing relational databases into ontologies, we propose a semi-automatic OWL-based approach. The proposed approach analyzes the content of a database by using its HTML forms and HTML tables[a]in order to restructure and enrich the relational schema. This paper is organized as follows. In Section 2 summarizes the approaches that deal with relational databases reverse engineering into ontologies. Section 3 explains our proposed approach. The Detection rules that use the structure of an HTML-form are shown in Section 4. Section 5 describes the process of semantics discovery and enrichment of a relational schema. Section 6 details the rules for building an OWL ontology from the enriched relational schema. Section 7 presents a portal prototype implementation of the ontology constructor. Finally, Section 8 contains concluding remarks and suggests some future works.

2 Related works

Several efforts have been put into relational databases reverse engineering by focusing on how to define semantics in a database schema [9], how to extract semantics out of a database schema [5], and how to transform a relational model into an object-oriented model [10, 11, 12]. However, these efforts' results do not meet the requirements of constructing ontology such as property inheritance and the use of negation constructor. Although the object-oriented model is close to the ontology theory, there are still some differences. For example, properties in an object-oriented model cannot be represented in a hierarchy. In the following we list some approaches that have drawn our attention with regard to the objective of our research. The approaches are classified in three categories that are the result of a survey of the literature addressing the techniques that consider ontologies as the target for reverse engineering of relational databases.

[a]In what follows HTML-form designates both HTML-form and HTML-table.

- *User query-based approaches*: in [13], Kashyap's approach builds and refines an ontology by analyzing the user requires that are run over a relational schema. However, this approach does not develop axioms, which are part of ontology. Moreover, the ontology refinement quality is not ensured since this depends on user's queries, which are random.

- *Relational schema-based approaches*: in [14], Stojanovic et al.'s approach provides a set of rules to map a relational database's constructs onto an RDF ontology constructs. These rules analyze relations, keys, and inclusion dependencies (which are not often available). In [15], Rubin et al.'s approach proposes to automate the process of filling the instances and their attributes values of an ontology using the data that are extracted from relational sources.

- *Tuple-based approaches*: since the relational schema is poor in explicit semantics [4], Astrova's approach [16] analyzes a relational database to discover additional "hidden" semantics (e.g., inheritance). However, this approach is very time consuming when the number of tuples in a relational database is huge.

As an attempt to solve the common problems of reverse engineering, recent research projects suggest data semantics extraction using HTML pages to be linked to wrapper [17, 18, 19, 20]. The drawback here is that any change in an HTML page can break the wrappers and thus, the ontologies they are based on. HTML pages are often redesigned typically more than twice a year [21]. Recently, Astrova proposed in [22] an HTML form based approach to extract ontology and to create ontological instances from HTML-pages' data. However, the main drawback of this approach is that it does not offer any way to identify inheritance relationship.

3 The proposed ontology development approach

To overcome some drawbacks of the aforementioned works, our approach proceeds by extracting semantics from the HTML-forms that relational databases expose to the external world. Prior to building the ontology in a semi-automatic way, the extracted semantics is combined with the proper semantics that underpins these relational databases.

3.1 Motivations

The following arguments motivate why we adopted HTML-forms to start with the process of generating an ontology:

- HTML-forms are convenient interfaces to enter, change, and view data on Web pages. Therefore, studying and analyzing HTML-forms can reveal important information such as mandatory data and range of data.

- HTML-forms are a structured collection of fields that are used to communicate with a relational database. While data in a form are usually structured, a relational database's structure is often not available in advance [23].

- HTML-forms 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. In the back-end of this interface, a

relational schema is probably not well-designed, not optimized, and even not normalized [8].

- Field names in HTML-forms are usually more explicit and meaningful than the corresponding relations' and attributes' names in a relational schema.
- HTML-forms are normally associated with instructions that provide additional information on how data are structured and managed [24].

3.2 Overview of the approach

The proposed ontology building approach consists of three rules-based engines (Figure 1):

- ***Structure detection engine*** has two sets of rules. The first set analyzes the HTML pages in order to identify the constructs of a form. The second set uses these constructs to produce this form's XML-schema.
- ***Semantic discovery engine*** has two sets of rules. The first set acquires the domain semantics by extracting the relational schemas of the forms and their dependencies constraints. The second set allows the enrichment of the database relational schema with the discovered semantics.
- ***Ontology building engine*** has two sets of rules. The first set automatically translates the enriched relational schema into OWL-based ontological constructs. Rules are organized into four groups for constructing classes, properties, hierarchies, and axioms. The second set is responsible for creating instances of the ontology using the relational tuples.

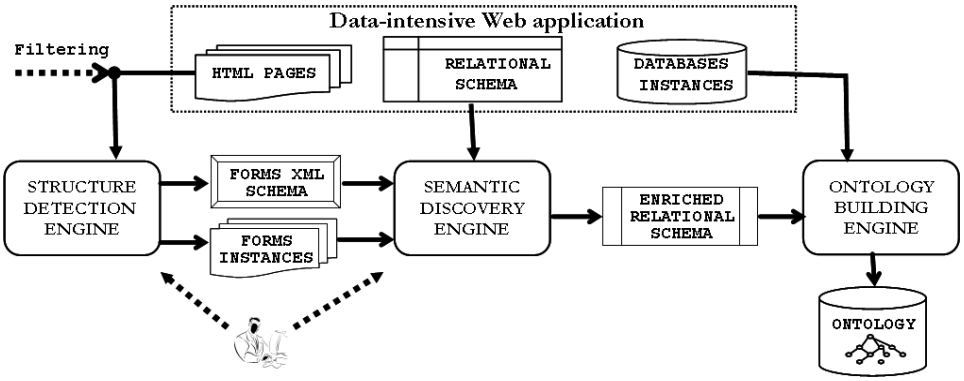

Fig. 1. Ontology building framework.

4 Form structure detection

For illustration purposes, we use the Algerian airline company Web site[b]. Two HTML pages among several are shown in Figure 2: *Booking* form and *Flight Program* table. The underlying source of the Web site is a relational database whose schema is given in Table 1. Underlined and italic attributes represent primary and foreign keys, respectively.

[b] *http://www.airalgerie.dz*

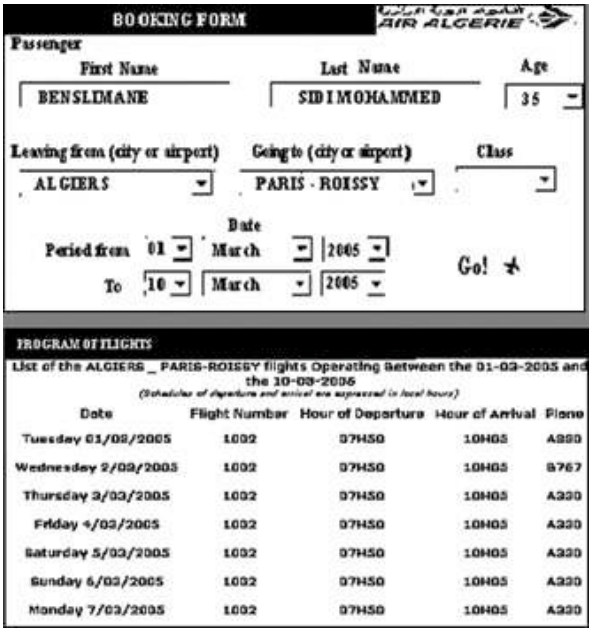

Fig. 2. HTML pages along with HTML-Form and HTML-Table.

Table 1. A relational database schema.

N	Relational schema
1	Passenger (PassengerID, FN, LN, Age)
2	City (CityID)
3	Departure-City (CityID, DC-Name)
4	Arrival-City (CityID, AC-Name)
5	Date (DeparatueDate)
6	Departure-Hour (HourID, type)
7	Arrival-Hour (HourID, type)
8	Company (CompanyID, CompanyName)
9	Plane (PlaneID, *CompanyID*, Capacity)
10	Flight (*FlightID*, *Dep-HourID, Arr-HourID, PlaneID*)
11	Leaving-From (*FlightID, Dep-CityID*)
12	Going-To (*FlightID, Arr-CityID*)
13	Book (*PassengerID, FlightID, DepartureDate*, Price, Class)

4.1 Analysis of HTML pages structure

This analysis phase uses an HTML-form to clarify its structure, identify its components and relationships, and finally extract its E/R schema.

4.1.1 Model of a form

A conceptual model of a form using an E/R schema was originally proposed in [25] and partially represented in Figure 3. We extended this schema by adding some constructs that

allow references between forms. In addition, we introduced the concepts of *field underlying source* and *linked-attribute* to keep a close link with the database that is under analysis. Basically, the extended E/R schema consists of:

- ***Form type***: is a structured collection of empty fields that are formatted in a way that permits communication with the database. A particular representation of a form type is called *form template* that suggests three basic components namely title, captions, and entries.
- ***Structural units (SUs)***: correspond to objects that closely group related fields in a form.
- ***Form instance***: corresponds to an occurrence of a form type. This is the extensional part that is obtained when a form template is filled in with data. Figure 2 shows two instances of *Booking* and *Program of flights* forms type.
- ***Form field***: consists of a caption and its associated entry. Each entry is generally linked to a table's name as per the table names in the underling database. The values that a form field displays/receives are provided by (or stored in) the linked-attribute. Some form fields are computed; others can be simply not linked to the relational database. We distinguish three types of fields: filling fields (e.g., *TEXT, CHECK-BOX, RADIO, TEXTAREA* attributes); selection fields (e.g., *SELECT* attribute); and link fields (*HREF* attribute).
- ***Underlying source***: corresponds to the structure of the relational database (i.e., a relational schema) in terms of relations and attributes along with their data types.
- ***Relationship***: is a connection between SUs. There are two kinds of relationship: Membership (belongs to) and Reference (refers to). *Membership* is one-to-many or one-to-one relationship between two SU types. One of the SUs (always the one-side) is called the parent SU, the other (many-side or sometimes also one-side) is called the child SU. An occurrence of a relationship consists of one SU occurrence of the parent and one or several occurrences of the child SU. *Reference* is a many-to-many relationship between SU types. A SU can refer to one (maybe itself) or to many other SUs.
- ***Constraint***: is a rule that defines which data validity for a given form field. For instance, a cardinality constraint specifies for an association relationship the number of instances that a SU can participate in.

4.1.2 Identification rules to obtain a form's E/R schema

The following rules summarize the mechanisms that permit identifying a form model's constructs using a relational schema as input. These rules populate the structure detection engine of Figure 1.

Rule *i1*: Form instance identification. In order to differentiate the different contents in an HTML document, Web pages are usually split into multiple areas. We refine these contents by removing stop words and useless tags like `<b>` and `<i>` and by preserving the following sections:

- The section between open and closing `<form>` tags that are used to access and update the relational database.

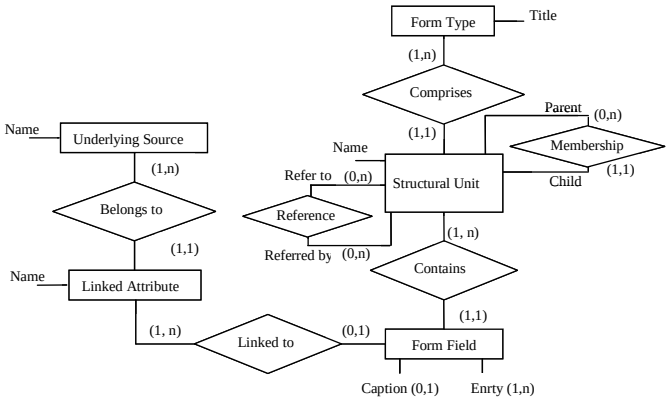

Fig. 3. Extended E/R schema of a form

- The section between open and closing (`<table>, <td>,<tr>,<li>,<ul>`) tags that are returned following user query execution. This represents a particular view of the relational database.

Rule *i2*: Linked attributes identification. Linked attributes are identified as follows:

- In an HTML-form, the value of attribute *name* in `<Input>, <Select>,` and `<Textarea>` tags is associated with the text segment that is located immediately ahead these tags. This attribute's name will be used in the enrichment process.
- In an HTML-table, the value of the structural tags `<thead>` and `<th>` [26].

If the linked attributes are not separated with the structural tags (merged data), we use visual cues [27, 18]. This approach typically means that there will be some separators (e.g., blank areas) that help users split the merged data.

Rule *i3*: Structural unit identification. To determine the logical structure of an HTML page (i.e., meaning of the page as understood by users), we use visual cues [27]. E.g., users might consider *FirstName, LastName*, and *Age* in Figure 2 as one entity (Passenger).

Rule *i4*: Relationship identification. Relationships can be established when two SUs are included in the same HTML page. Since a relational database's content does not reside in a single HTML page, extra relationships could be identified using hyperlinks. Hyperlinks are interpreted in many cases as semantic relations between SUs.

Rule *i5*: Constraint identification. In addition to an HTML page's constructs, data are analyzed to identify additional constraints. A data analysis includes a strategy of learning by example, borrowed from machine learning techniques [28]. For example, in Figure 2 we could identify a constraint *Not Null* on the linked attributes *Departure-City* and *Arrival-City*.

4.2 Form XML-schema generation

When the structure of the form type is extracted, the corresponding XML-schema can then be generated based on a set of translation rules.

Rule *g1***.** Each SU in the form type is translated into a complexType element in the corresponding XML schema.

Example: SU *Passenger* becomes as follows:
`<xsd:complexType name="passenger">...</xsd:complexType>`

Rule *g1* is recursively applied to all complex SU components.

Rule *g2***.** Each form field in an SU is translated into a sub-element of the corresponding complexType element. The primitive type of the element adopts the field type.

Example. Field *FirstName* is translated into a string type:
`<xsd:element name="firstname" type="xsd:string"/>`

Rule *g3***.** If an SU contains simple filling fields (e.g., *TEXT* tag), the corresponding ComplexType element takes `(minOccurs="1")` and `(maxOccurs="1")` as occurrence.

Rule *g4***.** If an SU contains multiple filling fields (e.g., *MULTIPLE* attribute), the corresponding ComplexType element takes `(maxOccurs="*")` as maximum occurrence.

Rules *g3* and *g4* are recursively applied to the form fields of each SU. While applying *g1*, *g2*, *g3* and *g4* rules to *Booking* form type structure, the obtained XML-schema is given in Table 2.

Table 2. XML schema of *Booking* Form.

≺ ?xml version=”1.0”? ≻
≺ xsd:schema=xmlns:xsd=”http://www.w3.org/2001/XMLSchema ≻
≺ xsd:complexType name=”formulaire-de-rservation” ≻
≺ xsd:attribute name=”class” type=”xsd:integer”/ ≻
≺ xsd:complexType name=”Passenger” type=”xsd:”PassengerID” maxOccurs=”1”/ ≻
≺ xsd:complexType name=”City” type=”xsd: ”CityID” maxOccurs=”1”/ ≻
≺ xsd:complexType name=”Date” type=”xsd: ”DateID” maxOccurs=”1”/ ≻
≺ xsd:complexType name=” PassengerID” ≻
≺ xsd:element name=”FirstName” type=”xsd:string”/ ≻
≺ xsd:element name=”LastName” type=”xsd:string”/ ≻
≺ xsd:element name=”Age” type=”xsd:integer”/ ≻
≺ xsd:complexType/ ≻
≺ xsd:complexType name=” CityID” ≻
≺ xsd:element name=”LeavingFrom” type=”xsd:string”/ ≻
≺ xsd:element name=”GoingTo” type=”xsd:string”/ ≻
≺ xsd:complexType/ ≻
...
≺ xsd:complexType/ ≻
≺ xsd:complexType/ ≻
≺ xsd:schema/ ≻

5 Semantic discovery engine

The role of the semantic discovery engine is to enrich the relational schema with semantics using the respective form's XML schema. The form's XML schema are used to identify form's relations to enable the extraction of the functional and inclusion dependencies.

5.1 Form semantic extraction

Firstly, a form's relations and respective primary keys are identified using the nodes of the form's XML schema and the underlying database. Secondly, functional and inclusion dependencies between these relations are extracted from the form's instances.

5.1.1 Form relations extraction

The identification of a form's relations and respective primary keys consists of determining the equivalence and/or the similarity between the nodes of the XML-form hierarchical structure and the relations in the underlying database. This is the starting point from a reverse engineering perspective [25]. A node in an XML-form hierarchical structure may be either:

- Equivalent to a relation in the underlying database, i.e., node and relation have a same set of attributes;
- Similar to a relation, i.e., its set of attributes is a subset of the attributes of the relation;
- A set of relations, i.e., its set of attributes gathers several relations in the underlying database.

For dependent nodes, the primary keys are formed by concatenating the primary key of its non-null parent node with its requires the interaction with the analyst who can identify the objects that do not verify the equivalence and similarity properties defined above. While applying this process on both the hierarchical structure of *Booking* form and the relational schema of the underlying database in Table 1 the following relational sub-schema is extracted:

```
City(CityID)
Passenger(PassengerID, FirstName, LastName, Age)
Departure-City(CityID, DepartureCityName)
Arrival-City(CityID, ArrivalCityName)
Date(DepartureDate)
```

From the *Program of flights* form, the following relational sub-schema is extracted:

```
Departure-Hour(HourID, type)
Arrival-Hour(HourID, type)
Plane(PlaneID, capacity)
Flight(FlightNumber, DepartureCityID, ArrivalCityID,DepartureHourID, ArrivalHourID,
PlaneID)
```

From the relationships between the hierarchical structure of *Booking* and *Flight Program* forms, the following relational sub-schema is identified:
`Book(PassengerID, FlightNumber, DepartureDate, Class)`.

5.1.2 Functional dependencies extraction

The extraction of functional dependencies from a database extension has received a great deal of attention [29, 30, 31]. In our approach we use Malki et al.'s algorithm [25] that introduces

two ways to reduce the time for extracting functional dependencies. The first way is to replace database instances with a compact representation that is, the form instances. The second way is to use only the non-primary attributes in the left side of functional dependencies. While applying this algorithm on the sub-schema of *Program of flights* form and its instances, the following functional dependency is identified: `PlaneID` → `FlightNumber`, which means that a plane ensures only one flight.

5.1.3 Inclusion dependencies extraction

In our approach, we formulate possible inclusion dependencies between relation's key of relational sub-schemas of forms. The time complexity of this process is more optimized with regard to other approaches [31, 5], because the possible inclusion dependencies are verified by analyzing the form extensions which are more compact representation with regard to the database extension [25]. The heuristics employed by the extraction process for proposing possible inclusion dependencies are described below:

- Two relations R_i and R_j, having the same primary key X, may be linked by an inclusion dependency, noted: $((R_i, X), (R_j, X))$.
- When the primary key X of a relation R_i is a foreign key Y in an other relation R_j, then it is possible to have the inclusion dependency, noted: $((R_j, Y), (R_i, X))$.

While applying this algorithm on the sub-schema of *Booking* form, the following inclusion dependencies are obtained:

```
((Departure-City,CityID),(City,CityID))
((Arrival-City,CityID),(City,CityID))
```

5.1.4 Integration of forms sub-schemas

Forms sub-schemas are merged into one global schema by using integration schema techniques. We agree with [32], who claims that the schema integration process consists of two phases: comparison and conformity of schemas, and merging and restructuring of schemas. The comparison phase performs a dual comparison of relations (of the sub-schemas) and finds possible relation pairs. The merging and restructuring phases generate an integrated schema from two component schemas that have been compared. The result of integration schema is accumulated into a single schema, which evolves gradually toward the global schema of forms.

5.2 Semantic enrichment of the relational schema

The knowledge extracted throughout the steps discussed in the previous sub-sections is used to clarify, restructure, and augment the semantics of the relational schema by making use of the following enrichment rules:

Rule *e1***: Relational schema clarification.** Relation and attribute names of a form schema are usually more explicit than those of a relational schema. Thus, we retain names apparent in the global schema of forms instead those of the relational schema.

Example: Notice the adaptation of the name *DepartureCityName* to the attribute. This can better convey the meaning of data than the original attribute name *DC-Name* would. The same for *FirstName*, and *FlightNumber* instead *FN* and *FlightID*.

Rule *e2*: Relational schema restructuration. To satisfy the 3NF requirement we use the algorithm [33] that convert 1NF into 3NF through the recovered functional dependencies. During the conversion, all the semantics specified in the original relation definitions in database is preserved.

Example: Let be the following relations and recovered functional dependency:
`Flight(FlightNumber,DepartureHourID,ArrivalHourID,PlaneID);`
`Plane(PlaneID,companyID,Capacity).`
`PlaneID` → FlightNumber

The Relational schema restructuration will give us the following 3NF relations:
`Flight(FlightNumber,DepartureHourID,ArrivalHourID);`
`Plane(PlaneID,companyID,FlightNumber,Capacity).`

Rule *e3*: Constraint and dependency addition. To allow adding the extracted inclusion dependencies and constraints to the relational schema, we extend the usual formal definition of the relational model [34] with additional constructs typically found in SQL-DDLs, i.e., constructs which allow to state inclusion dependencies. In the next section, the added dependencies and constraints are used to detect respectively hierarchical and axioms construct in the target OWL ontology. The application of the enrichment rules on the relational schema described in Table 1, generates the enriched relational schema given in Table 3.

Table 3. Enriched relational database schema.

N	Schma relationnel enrichi
1	Passenger (PassengerID, FirstName, LastName, Age)
2	City (CityID)
3	Departure-City (CityID, DepartureCityName)
4	Arrival-City (CityID, ArrivalCityName)
5	Date (DeparatueDate)
6	Departure-Hour (HourID)
7	Arrival-Hour (HourID)
8	Company (CompanyID, CompanyName)
9	Plane (PlaneID, *CompanyID, FlightNumber*, Capacity)
10	Leaving-From (*FlightNumber, DepartureCityID*)
11	Going-To (*FlightNumber, ArrivalCityID*)
12	Flight (FlightNumber, *DepartureHourID, ArrivalHourID*)
13	Book (*PassengerID, FlightNumber, DepartureDate*, Price, Class)

Inclusion dependencies	**Constraints**
((Departure-City, CityID), (City, CityID))	DepartureCityName : NotNull
((Arrival-City, CityID), (City, CityID))	ArrivalCityName : NotNull
...	...

6 Ontology building

Before exposing our ontology development rules, the central features of a relational model are described and aligned with those provided by an ontological model.

6.1 Relational schema Vs Ontology

The underlying model of a source database is the relational model that is extended with additional constructs, while the target ontological model is the one defined by [35].

Definition 1 *Relational model is a 8-tuple (R, A, T, attr, dom, U, I, notnull), where:*

1. *R is a finite set of relations.*
2. *A is a finite set of attributes.*
3. *T is a set of atomic data types.*
4. *$attr : R \rightarrow 2^A$ is a function that returns attributes contained in a specific relation $r_i \in R$.*
5. *$dom : A \rightarrow T$ is a function that returns type of an attribute.*
6. *U is a set of unique constraints of the form (r, A_r) where:*
   - *$r \in R$;*
   - *$A_r \subseteq att(r)$.*
7. *I is a set of inclusion dependencies where:*
   - *each element has the form: $((r_1, A_1), (r_2, A_2))$;*
   - *$r_1, r_2 \in R$;*
   - *$A_1 = \{a_{11}, .., a_{1n}\}$;*
   - *$A_2 = \{a_{21}, .., a_{2n}\}$;*
   - *$A_1 \subseteq attr(r_1)$ and $A_2 \subseteq attr(r_2)$;*
   - *$|A_1| = |A_2|$ and $dom(a_{1i}) = dom(a_{2i})$.*
8. *$notnull : R \rightarrow 2^A$ is a function which states the attributes of a relation that need to have a value.*

All these constitute a relational schema, which describes the structure of data. Tuples in relations reflect the values of a schema, and are the content of a database.

In our modelling, we consider that the set of unique constraints comprises those explicitly defined in the SQL-DDL by means of primary keys and those that are recovered by the semantic discovery engine (Section 5). Note that SQL enforces automatically that a relation has only one primary key. Note also that our approach assumes that a relation has at least one primary key.

In the same way, the set of inclusion dependencies comprises those explicitly defined in the SQL-DDL, by means of foreign keys, and those that are recovered by the semantic discovery engine. Note that SQL enforces that if $((r_1, A_1), (r_2, A_2))$ is a foreign key then (r_2, A_2) must be a primary key (and, hence $(r_2, A_2) \subseteq U$).

Definition 2 *Ontology is a six-tuple O=(C, A_C, R, A_R, H_C, X), where:*

1. *C is a finite set of concepts;*
2. *A_C is a collection of attribute sets about concepts;*

3. *R is a set of relations, each relation has a pair of concepts;*
4. *A_R is a collection of attribute sets about relations;*
5. *H_C is called concept hierarchy or taxonomy, which is a directed relation $H_C \subseteq C \times C$;*
6. *X is a set of axioms that describe additional constraints on the ontology.*

Based on the ontological structure, ontology comprises a set of instances, which could be seen as the concept extension.
As a result of the ongoing process of defining a standard ontology Web language, a number of intermediate versions have been defined (OIL, DAML, DAML+OIL, etc.). Unlike [36] that uses Frame Logic as the ontology description language, we adopt in this paper the latest standard recommended by W3C, namely, OWL (Ontology Web Language) [37].

6.2 Ontology development rules

Both relational, and ontology models are a kind of model for organizing knowledge. In the relational model, both entities and relationship are expressed by relations. Therefore one relation in the relational model may be corresponding to an ontological concept or relation. If two relations in database have inclusion dependency, then the two corresponding ontological concepts (or relations) will have a hierarchical relationship. Additionally, attribute constraints in a database may be converted into ontological axioms, and the tuples in database may constitute ontological instances. In what follows we present a set of ontology development rules. The rules are organized in five groups.

6.2.1 Rule for classes development

Rule *c1.* An OWL class c_i can be created based on the relation r_i, if one of the following conditions can be satisfied:

(i) $A_i \subseteq attr(r_i)$ and $(r_i, A_i) \in U$ and $|A_i| = 1$;

(ii) $A_i \subseteq attr(r_i)$ and $(r_i, A_i) \in U$ and $|A_i| > 1$; and there does not exist an A_1 such that $A_1 \subseteq A_i$ and $((r_i, A_1), (r_j, A_j)) \in I$.

Comment: Rule *c1* indicates that if the relation's primary key is atomic (i.e the relation describes an entity, or there is not a part of the relation's primary key that reflects a reference relationship to another relation, then the relation can be mapped into one ontological class.

6.2.2 Rules for properties development

OWL distinguishes two kinds of properties: object properties and datatype properties. By default a property is a binary relation between thing and thing. Properties can be functional, i.e., their range may contain at most one element. Their domain is always a class. Object properties may additionally be inverse functional, transitive, symmetric, or inverse to another property. Their range is a class, while the range of datatype properties is a datatype.

Rule *c2.* For relations r_i and r_j, if $A_i \subseteq attr(r_i)$, and $(r_i, A_i) \notin U$, and $((r_i, A_i), (r_j, A_j)) \in I$ are satisfied, then an object property P can be created based on A_i. Let c_i and c_j classes corresponding to r_i and r_j respectively, the domain and range of P are c_i and c_j.

Comment: Rule *c2* indicates that if a no-key attribute of one relation only reflects a reference relationship to another relation, then it will be used as an object property in ontology.

Rule *c3.* For relations r_i and r_j, two ontological objects property "has-part" and "is-part-of" will be created, if the two following conditions are satisfied:

(i) $A_i \subseteq attr(r_i)$, $(r_i, A_i) \in U$, and $|A_i| > 1$;

(ii) $((r_i, A_i), (r_j, A_j)) \in I$, $((r_j, A_j), (r_i, A_i)) \in I$, and $(r_j, A_j) \in U$.

Suppose that the classes corresponding to r_i and r_j are c_i and c_j respectively, the domain and range of "is-part-of" are c_i and c_j and the domain and range of "has-part" are c_j and c_i. Properties "has-part" and "is-part-of" are two inverse properties.

Comment: Rule *c3* indicates that if a part of a primary key of one relation reflects a reference relationship to another relation, then the entity described by second relation is a special kind of the entity described by the first relation. The classes corresponding to the two relations should have objects properties "is-part-of" and "has-part" respectively.

Rule *c4.* For relations r_i, r_j, and r_k, if $(r_i, A_i) \in U$, $(r_j, A_j) \in U$, and $((r_k, A_k), (r_i, A_i)) \in I$, $((r_k, A_k), (r_j, A_j)) \in I$, where $A_i \cup A_j = A_k$, and $A_i \cap A_j = \oslash$, then two object properties $P_j^{'}$ and $P_j^{''}$ can be created based on the semantics of r_k. Suppose that the classes corresponding to r_i and r_j are c_i and c_j respectively, the domain and range of $P_j^{'}$ are c_i and c_j and the domain and range of $P_j^{''}$ are c_j and c_i. $P_j^{'}$ and $P_j^{''}$ are two inverse properties.

Comment: Rule *c4* indicates that if a relation r is used to describe the relationship between two others relations, two inverse object properties can be acquired based on semantics of the relation r.

Rule *c5.* For relations r_1, r_2,...,r_i, and r_k, if $(r_t, A_t) \in U$, and $((r_k, A_k), (r_t, A_t)) \in I$, $\forall t = 1, i$, where $A_1 \cup A_2 ... \cup A_i = A_k$, and $A_1 \cap A_2 ... \cap A_i = \oslash$, then object properties $P_1, P_2, ..., P_i$ can be created. Their domain is the class c_k corresponding to r_k and their range are the classes $c_1, c_2, ..., c_i$ corresponding to the relations $r_1, r_2, ..., r_i$, respectively.

Comment: Rule *c5* indicates that if one relation is used to describe the relationship among multi-relations, it should be decomposed to multiple object properties in ontology.

Rule *c6.* For an ontological class c_i and the datatype properties set of c_i denoted as $DP(c_i)$, if c_i corresponds to relations $r_1, r_2, ..., r_i$ in database, then for every attribute in $r_1, r_2, ..., r_i$, if it cannot be used to create object property by using Rule *c2* or Rule *c5*, then it can be used to create datatype property of C_i. The domain and range of each property P_i are C_i and $dom(A_i)$ respectively, where $P_i \in DP(C_i)$ and $A_i \subseteq attr(r_i)$.

Comment: Rule *c6* indicates that for each attribute in relations, if it cannot be converted into ontological object property, it can be converted into ontological datatype property.

6.2.3 Rule for Inheritance development

In OWL, the classes and properties can be organized in a hierarchy. In our approach this hierarchy can be discovered through inclusion dependencies.

Rule *c7.* For relation r_i and r_j, if $A_i \subseteq attr(r_i)$, $(r_i, A_i) \in U$, $A_j \subseteq attr(r_j)$, $(r_j, A_j) \in U$, and $((r_i, A_i), (r_j, A_j)) \in I$ are satisfied, then the class corresponding to r_i is a subclass of the class corresponding to r_j.

Comment: Rule *c7* indicates that if two relations in database have an inclusion dependency based on their primary keys, then the two corresponding ontological class or property can be organized in a hierarchy.

6.2.4 Rules for axioms development

OWL defines property cardinality to further specify properties. The property cardinality can be created from the constraint on attributes in relations. Each relational attribute has by default a maximum cardinality of one and a minimum cardinality of zero.
If the attribute is not null then the minimum cardinality is altered to 1. For foreign keys the maximum cardinality is unlimited, unless the foreign key is part of the primary key of the relation.
In OWL, a property when applied to a class can be constrained by cardinality restrictions on the domain giving the minimum (minCardinality) and maximum (maxCardinality) number of instances, which can participate in the relation. In addition, an OWL property can be globally declared as functional (functionalProperty) or inverse functional (inverseFunctional).

Rule *c8.* For relation r_i and $A_i \subseteq attr(r_i)$, if $(r_i, A_i) \in U$, then the minCardinality and maxCardinality of the property P_i corresponding to A_i is 1.

Rule *c9.* For relation r_i and $A_i \subseteq attr(r_i)$, if A_i is declared as *NOT NULL*, the minCardinality of the property P_i corresponding to A_i is 1.

Comment: Rules *c8, c9* indicate that some constraints of attribute in relation may be converted to the property cardinality in ontology.

According to Rule *c1* to Rule *c9*, an OWL ontological structure is acquired from the enriched relational schema.

6.2.5 Rules for instance migration

Once the ontology is created, the process of data migration can start. The objective of this task is the creation of ontological instances (that form a knowledge base) based on the tuples of the relational database. For an ontological class c_i, its instances consist of tuples in relation(s) corresponding to c_i and relations between instances are established using the information contained in the foreign keys in the database tuples. The data migration process has to be performed in two phases based on the following rules:

Rule *c10.* First, the instances are created. To each instance is assigned a unique identifier. This translates all attributes, except for foreign-key attributes, which are not needed in the metadata.

Rule *c11.* Second, relations between instances are established using the information contained in the foreign keys in the database tuples. This is accomplished using a mapping function that maps keys to ontological identifiers.

Comment: Rules *c10* and *c11* show that for one ontological class, its instances consist of tuples in relations corresponding to the class and relations between instances are established using the information contained in the foreign keys in the database tuples.

7 Implementation

In this section, we present some experiments we performed to assess the effectiveness of the proposed approach to semi-automatically build an OWL ontology from a relational database using the related HTML-forms. The main purpose of the experiments is to evaluate the effectiveness of the ontology development rules presented in the previous sections, and to verify that the proposed approach can contribute to help users build ontologies.

7.1 OWL ontology generation algorithm

Since the construction of an OWL ontology from an enriched relational schema is characterized by the specific rules, the generation of this ontology can be automated. We developed an algorithm (Figure 4) that describes the OWL ontology construction process.

```
Algorithm Relational_TO_Ontology(RDB_Schema)
Input: Enriched schema of relational database
Output: OWL ontology
Begin
  for r_i ∈ RDB_Schema do
    if A_i ⊆ attr(r_i), (r_i, A_i) ∉ U, and((r_i, A_i), (r_j, A_j)) ∈ I then
      ▷ create object property A_{i_{r0}} having class C_x as domain and class C_y as range
      ▷ create A_{i_{r-1}} as inverse property of A_{i_{r0}}
    else create ontological class C_i
      for A_i ⊆ attr(r_i) ∧ ¬∃((r_i, A_i), (r_j, A_j)) ∈ I do
        ▷ create datatype property A_i having class C_i as domain and corresponding xsd datatype as datatype
        if A_i_is_NOT_NULL then
          ▷ declare minCardinality of property A_i as 1
        end if
      end for
      if A_i ⊆ attr(r_i), (r_i, A_i) ∈ U ∧ ∃((r_i, A_i), (r_j, A_j)) ∈ I then
        ▷ declare axiom C_i 'is subClassOf' C_j
      else create object property A_i having class C_i as domain and class C_j as range
        if (r_i, A_i) ∈ U, |A_i| > 1, ((r_i, A_i), (r_j, A_j)) ∈ I, ((r_j, A_j), (r_i, A_i)) ∈ I, and(r_j, A_j) ∈ U then
          ▷ define someValuesFrom restriction on property haspart for class C_i to C_j
        end if
        if A_i_is_NOT_NULL then
          ▷ declare minCardinality of property A_i as 1
        end if
      end if
    end if
  end for
End
```

Fig. 4. Algorithm for translating an enriched relational schema into OWL ontology.

We describe here some parts of an OWL ontology generated by applying the ontology construction rules to our running example (Table 3). For example, class *Plane* is generated from relation *Plane* by using Rule *c1* and the non-foreign key attributes *PlaneID* and *Capacity*, which generate two datatype properties[c] by using Rule *c6*. Since *PlaneID* is a *NOT NULL* attribute, the minimum cardinality restriction on that property must be declared by using Rule *c9*. An object property is created from foreign key attribute *CompanyID* by using Rule *c2*. It has *Plane* as domain class and *Company* as range. The corresponding OWL description is as follows:

[c] Note that a property is named after the attribute name prefixed by the relation name in order to avoid possible name conflicts because different relation may contain same attributes.

```
<owl:Class rdf:about="\#Plane" />
<owl:DatatypePropty rdf:about="\#plane-PlaneID">
 <rdfs:domain rdf:resource="\#Plane"/>
 <rdfs:Datatype rdf:resource="\&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="\#plane-Capacity">
 <rdfs:domain rdf:resource="\#Plane"/>
 <rdfs:Datatype rdf:resource="\&xsd;integer"/>
</owl:DatatypeProperty>
<owl:Class rdf:about="\#Plane">
 <rdfs:subClassOf>
    <owl:Restriction>
        <owl:minCardinality>1</owl:minCardinality>
        <owl:onProperty>
            <owl:DatatypeProperty rdf:about="\#plane-PlaneID"/>
        </owl:onProperty>
    </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Concerning the part-whole relation detected by the Rule *c3*, since OWL does not provide any built-in primitives for that, object properties partOf and hasPart must be rigorously defined. The part-of relation between two classes is expressed by the someValuesFrom restriction on the property partOf which means that all individuals of the part class must be a part of an individual of a class that it is part of.
For data migration process, Table 5, illustrates an example result from the application of the Rules *c10* and *c11* on the relation tuples of Table 4.

Table 4. Relational database instances

Plane			Company	
PlaneID	CompanyID	Capacity	CompanyID	CompanyName
A330	1	150	1	Air Algeria
B767	2	200	2	Air France

7.2 Prototype

A prototype is developed using Java (j2sdk 1.4.2) and Jena 2.1, and the Java API for ontology development and processing. The prototype has been implemented in order to experiment and verify that the proposed approach is doable. Our tool has a user-friendly GUI to perform the ontology development process, and to produce an ontology stored in an OWL file. The Web site URL, the relational schema, and other parameters such as information for the database connection (e.g. JDBC driver, database URL), base URI and ontology URI of the output OWL ontology are given in an input configuration file.

First, the HTML pages are parsed to detect the form's structure and instances. It is up to the user to validate the result. Next all the semantic behind the forms structure and instances are extracted and used to enrich the relational schema. The ontology building engine (Figure 1) processes the enriched schema and generates the corresponding OWL ontology based on the algorithm previously described. The output ontology can be formalized in the following standard formats: OWL, RDF/XML, RDF/XML-ABBREV, N3 and N-Triples.

Table 5. Ontology instances.

```
≺ ?xml version="1.0"? ≻
≺ rdf:RDF
≺ owl:Ontology rdf:about=""/ ≻
≺ owl:Class rdf:ID="Company"/ ≻
≺ owl:Class rdf:ID="Plane"/ ≻
...
  ≺ Company rdf:ID="Company1" ≻
   ≺ CompanyId rdf:datatype="http://www.w3.org/2001/XMLSchema#int"≻ 1≺ /CompanyId ≻
   ≺ CompanyName rdf:datatype="http://www.w3.org/2001/XMLSchema#string" ≻
      Air Algeria ≺ /CompanyName ≻
  ≺ /Company ≻
  ≺ Plane rdf:ID="Plane1" ≻
   ≺ capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#int"150≺ /capacity ≻
   ≺ PlaneId rdf:datatype="http://www.w3.org/2001/XMLSchema#string" ≻ A330≺ /PlaneId ≻
   ≺ Possede rdf:resource="#Company1"/ ≻
  ≺ /Plane ≻
  ≺ Company rdf:ID="Company2" ≻
   ≺ CompanyName rdf:datatype="http://www.w3.org/2001/XMLSchema#string" ≻
      Air France≺ /CompanyName ≻
   ≺ CompanyId rdf:datatype="http://www.w3.org/2001/XMLSchema#int"≻ 2 ≺ /CompanyId ≻
  ≺ /Company ≻
   ...
≺ /rdf:RDF ≻
```

7.3 Experimental evaluation

In order to evaluate our approach, we performed two experiments on tourism domain. In the first experiment, we analyzed an airlines company Web site.[d] The constructs of the obtained OWL ontology are presented in Table 6.

Table 6. Results from the ontology development process using an airline company Web site.

OWL Ontology constructs	Constructs in the tutorial ontology (M)	Constructs extracted correctly (C)	Constructs extracted incorrectly (I)	Recall Ratio (C/M)	Precision Ratio C/(C+I)
Classes	30	15	1	0.50	0.94
Objects properties	16	09	1	0.56	0.90
Datatype properties	77	34	3	0.44	0.92

The results are compared to the tutorial ontology for a Semantic Web of tourism.[e] To evaluate the quality of the ontology development process, we compare the OWL ontology's constructs (correctly extracted: C, and incorrectly extracted: I) returned by the automatic extraction process with manually determined constructs (M) in the tutorial ontology for a Semantic Web of tourism. Based on the cardinalities of these sets, quality measures such as precision and recall are computed.

[d] *http://www.britishairways.com.*
[e] *http://protege.stanford.edu/plugins/owl/owl-library/travel.owl.*

$Precision = C/(C + I)$, is the faction of the automatic discovered constructs which are correct.
$Recall = (C/M)$, is the fraction of the correct constructs (the set M) which has been discovered by the ontology development process.

The low recall ratio is not so much a consequence of bad ontology development approach, but much more due to the restricted domain knowledge covered by the Web site itself. In the second experiment, we conducted experiments on three Web site related respectively to flights,[f] hotel,[g] and leisure[h]tourism activities. The ontology development process was rather successful, with average recall and precision ratios of 94% and 92% respectively (Table 7). The results obtained with the use of the second experiment could be much better if more Web sites covering a large part of the tourism activities were used as input.

Table 7. Results from the ontology development process using three Web site related respectively to flights, hotel and leisure tourism activities.

OWL Ontology constructs	Constructs in the tutorial ontology (M)	Constructs extracted correctly (C)	Constructs extracted incorrectly (I)	Recall Ratio (C/M)	Precision Ratio C/(C+I)
Classes	30	28	2	0.93	0.93
Objects properties	16	15	2	0.94	0.88
Datatype properties	77	73	5	0.95	0.94

8 Conclusion and future Work

With the proliferation of ontologies, the development of automated techniques for ontology building is crucial to the success of the semantic Web. The major difficulties in building ontology reside in the manual work that could be prone to error. Therefore, the use of a semi-automatic ontology extraction techniques is attractive. In this paper we focused on the problem of automating the generation of domain ontologies, at least partially, by applying reverse engineering technique. We presented the complete details of the process of semi-automatically creating an OWL ontology that corresponds to the content of a relational database based on the analysis of its related HTML-forms. 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.

However, in the most circumstances, the obtained ontological structure is coarse. In addition, some semantics of the obtained information needs to be validated. Existing repositories of lexical knowledge usually include authoritative knowledge about some domains. We suggest as future work refining the obtained ontology according to them, especially machine-readable dictionaries and thesauri (e.g. WordNet). In addition, we plan to add other more complex constraints specified in SQL in order to obtain all possible information of the relational databases, such as more complex CREATE DOMAIN sentences or CHECK constraints, or even triggers. Finally, it is our intention to integrate other learning techniques into our reverse engineering approach to obtain better result.

[f] *http://www.britishairways.com.*
[g] *http://www.hm-usa.com.*
[h] *http://www.travelandleisure.com.*

References

1. M. Erdmann, A. Maedche, H. Schnurr, S. Staab, From manual to semi-automatic semantic annotation: About ontology-based text annotation tools, in: Proceedings of the COLING 2000Workshop on Semantic Annotation and Intelligent Content, Luxembourg, 2000.
2. R. Volz, S. Handschuh, S. Staab, L. Stojanovic, N. Stojanovic, Unveiling the hidden bride: deep annotation for mapping and migrating legacy data to the semantic web., Journal of Web Semantic 1 (2) (2004) 187–206.
3. S. Haustein, J. Pleumann, Easing participation in the semantic web., in: Proceedings of the WWW2002 International Workshop on the Semantic Web, Hawaii, 2002.
4. N. F. Noy, M. C. A. Klein, Ontology evolution: Not the same as schema evolution., Knowl. Inf. Syst. 6 (4) (2004) 428–440.
5. R. H. L. Chiang, T. M. Barron, V. C. Storey, Reverse engineering of relational databases: Extraction of an eer model from a relational database., Data Knowl. Eng. 12 (2) (1994) 107–142.
6. W. J. Premerlani, M. R. Blaha, An approach for reverse engineering of relational databases., Commun. ACM 37 (5) (1994) 42–49, 134.
7. E. F. Codd, A relational model of data for large shared data banks., Commun. ACM 13 (6) (1970) 377–387.
8. R. J. Muller, Database Design for Smarties: Using UML for Data Modeling., M. Kaufmann, 1999.
9. J. Biskup, Achievements of relational database schema design theory revisited., in: Semantics in Databases, Vol. 1358 of LNCS, Springer, 1998, pp. 29–54.
10. M. W. W. Vermeer, P. M. G. Apers, Object-oriented views of relational databases incorporating behaviour., 1995, pp. 26–35.
11. A. Behm, A. Geppert, K. Dittrich, On the Migration of Relational Schemas and Data to Object-Oriented Database Systems, in: the 5th Int. Conference on Re-Technologies for Information Systems, Klagenfurt, 1997, pp. 13–33.
12. J.-L. Hainaut, J. Henrard, J.-M. Hick, D. Roland, V. Englebert, Database design recovery ., in: Advances Information System Engineering, 8th International Conference, (CAiSE'96), Vol. 1080 of LNCS, Springer, 1996, pp. 272–300.
13. V. Kashyap, Design and creation of ontologies for environmental information retrieval., in: Twelfth Workshop on Knowledge Acquisition, Modeling and Management, Alberta, Canada, 1999.
14. L. Stojanovic, N. Stojanovic, R. Volz, Migrating data-intensive web sites into the semantic web., in: Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'2002), ACM, Madrid, Spain, 2002, pp. 1100–1107.
15. D. L. Rubin, M. Hewett, D. E. Oliver, T. E. Klein, R. B. Altman, Automating data acquisition into ontologies from pharmacogenetics relational data sources using declarative object definitions and xml., in: Pacific Symposium on Biocomputing, 2002, pp. 88–99.
16. I. Astrova, Reverse Engineering of Relational Databases to Ontologies, in: the 1st European Semantic Web Symposium (ESWS), Heraklion, Crete, Greece, 2004, pp. 327–341.
17. A. Sahuguet, F. Azavant, Building intelligent web applications using lightweight wrappers., Data Knowl. Eng. 36 (3) (2001) 283–316.
18. J. Wang, F. H. Lochovsky, Data extraction and label assignment for web databases., in: Proceedings of the 12th international conference on World Wide Web (WWW'03), 2003, pp. 187–196.
19. D. W. Embley, Towards semantic understanding - an approach based on information extraction ontologies., in: Proceedings of the 25th Australasian Database Conference, New Zealand, 2004.
20. S. Suwanmanee, D. Benslimane, P.-A. Champin, P. Thiran, Wrapping and integrating heterogeneous relational data with owl., in: ICEIS (1), 2005, pp. 11–18.
21. D. Florescu, A. Y. Levy, A. O. Mendelzon, Database techniques for the world-wide web: A survey., SIGMOD Record 27 (3) (1998) 59–74.
22. I. Astrova, B. Stantic, An HTML Forms driven Approach to Reverse Engineering of Relational Databases to Ontologies, in: eds. M. H. Hamza (Ed.), the 23rd IASTED International Conference on Databases and Applications (DBA), Innsbruck, Austria, 2005, pp. 246– 251.

23. J. Choobineh, M. V. Mannino, V. P. Tseng, A form-based approach for database analysis and design, Commun. ACM 35 (2) (1992) 108–120.
24. M. V. Mannino, J. Choobineh, J. J. Hwang, Acquisition and use of contextual knowledge in a form-driven database design methodology., in: Proceedings of the Fifth International Conference on Entity-Relationship Approach, 1986, pp. 361–377.
25. M. Malki, A. Flory, M. K. Rahmouni, Extraction of object-oriented schemas from existing relational databases: a form-driven approach., Informatica, Lith. Acad. Sci. 13 (1) (2002) 47–72.
26. Y. A. Tijerino, D. W. Embley, D. W. Lonsdale, Y. Ding, G. Nagy, Towards ontology generation from tables., World Wide Web 8 (3) (2005) 261–285.
27. Y. Yang, H. Zhang, Html page analysis based on visual cues., in: ICDAR, IEEE Computer Society, Seattle, WA, USA, 2001, pp. 859–864.
28. R. S. Michalski, A theory and methodology of inductive learning., Artif. Intell. 20 (2) (1983) 111–161.
29. I. Budak Arpinar, B. Aleman-Meza, R. Zhang, A. Maduko, Extracting entity relationship schema from a relational database through reverse engineering, in: proc. of ER'94, San-Diego, USA, 2004.
30. H. Mannila, K.-J. Räihä, Design of Relational Databases Addison., Addison-Wesley Publishing Company, 1992.
31. J.-M. Petit, F. Toumani, J.-F. Boulicaut, J. Kouloumdjian, Towards the reverse engineering of denormalized relational databases., in: Proceedings of the Twelfth International Conference on Data Engineering, 1996, pp. 218–227.
32. C. Batini, M. Lenzerini, S. Navathe, A Comparative Analysis of Methodologies for Databases Schema Integration, ACM Computing Surveys 18 (1986) 323–364.
33. B. Salzberg, Third normal form made easy., SIGMOD Record 15 (4) (1986) 2–18.
34. S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.
35. A. Maedche, S. Staab, Ontology learning for the semantic web, IEEE Intelligent Systems 16 (2), special Issue on Semantic Web.
36. S. M. Benslimane, M. Malki, D. Amar Bensaber, Automated Migration of Data-Intensive Web Pages into Ontology-Based Semantic Web: A Reverse Engineering Approach, in: Proceedings of International Conference on Ontologies,Databases and Applications of SEmantics (ODBASE'2005), Vol. 3761 of LNCS, Springer, Cyprus, 2005, pp. 1640 – 1649.
37. M. K. Smith, C. Welty, D. McGuinness, Owl web ontology language guide. W3C Recommendation. http://www.w3.org/TR/owl-guide/ (2004).