INFORMATICA, 2007, Vol. 18, No. 4, 511–534
© 2007 *Institute of Mathematics and Informatics, Vilnius*

Extracting Personalised Ontology from Data-Intensive Web Application: an HTML Forms-Based Reverse Engineering Approach

Sidi Mohamed BENSLIMANE, Mimoun MALKI
Computer Science Department, University of Sidi Bel Abbes
B.P 89 Sidi Bel Abbes, 22000 Algeria
e-mail: benslimane@univ-sba.dz, malki@univ-sba.dz

Mustapha Kamal RAHMOUNI
University of Oran
BP 1524 El Mnaouer, 31000, ORAN, Algeria
e-mail: rahmouni@mail.univ-oran.dz

Djamal BENSLIMANE
LIRIS Laboratory, University of Claude Bernard Lyon
8 Bld Niels Bohr, 69622, Villeurbanne Cedex, France
e-mail: djamal.benslimane@liris.cnrs.fr

Received: August 2005

Abstract. The advance of the Web has significantly and rapidly changed the way of information organization, sharing and distribution. The next generation of the web, the semantic web, seeks to make information more usable by machines by introducing a more rigorous structure based on ontologies. In this context we try to propose a novel and integrated approach for a semi-automated extraction of ontology-based semantic web from data-intensive web application and thus, make the web content machine-understandable. Our approach is based on the idea that semantics can be extracted by applying a reverse engineering technique on the structures and the instances of HTML-forms which are the most convenient interface to communicate with relational databases on the current data-intensive web application. This semantics is exploited to produce over several steps, a personalised ontology.

Key words: semantic web, reverse engineering, ontology, HTML-forms, data-intensive web application.

1. Introduction

The actual web has been moving away from static, fixed web pages to dynamically-generated at the time of user request. This kind of web site is called data-intensive web site (Fraternali, 1999), and usually realized using relational databases (i.e., e-commerce application). Data-intensive web pages are characterized by an automated update of the web content and a simplified maintenance of the web design (Stojanovic *et al.*, 2002). Nevertheless they suffer from two limitations. First, they form a hidden web since its content is not easily accessible to any automatic web content processing tools including the search engine indexing robots. Second the content of the database-driven web pages presented by using HTML is not machine-understandable. The simplicity and proliferation of the World Wide Web has taken the availability of information to an unprecedented level. The next generation of the web, the semantic web, seeks to make information more usable by machines by introducing a more rigorous structure based on ontologies, and thus resolve the second problem of data-intensive Web pages. Ontology is one of the most important concepts in knowledge representation. It can be generally defined as shared formal conceptualization of particular domain between members of a community of interest, which help them exchange information (Gruber, 1995). Lately, ontologies have become the focus for research in several other areas, including knowledge engineering and management, information retrieval and integration, agent systems, the semantic web, and e-commerce. The availability of formal ontologies is crucial for the success of the semantic web. Nevertheless building ontologies is so costly that it hampers the progress of the semantic web activity. Manual construction of ontologies (Erdmann *et al.*, 2000; Volz *et al.*, 2004) is a difficult, time-consuming and error-prone task and easily causes a knowledge acquisition bottleneck. Fully automated tools are still at the very early stage to be implemented. Therefore, the use of a semi-automatic ontologies extraction is seen as the practical short terms solution which allows knowledge extraction from data-intensive web applications. Reverse engineering technique appears as an interesting solution to reach this objective. It's defined as a process of analyzing a "legacy" system to identify all the system's components and the relationships between them (Chiang *et al.*, 1994).

However, because of the novelty of that area, there are few approaches that consider ontologies as the target for reverse engineering. These approaches usually require more input information than is possible to provide in practice, as the complete information about a relational database is usually unavailable, and make unrealistic assumptions about the input. E.g., the relational database is in third normal form (3NF). As an attempt to overcome these limitations, we propose in this paper a novel approach to reverse engineering data-intensive web application into ontology-based semantic web.

This paper is organized as follows: in Section 2, we discuss some of the related works in reverse engineering relational databases into ontologies. Section 3 explains the overall reverse-engineering architecture and Section 4 details our proposed approach. Section 5 presents a portal prototype implementation of the ontology construction approach. Finally, Section 6 contains concluding remarks and suggests some future works.

2. Related Works

Several researches have been done on relational databases reverse engineering, suggesting methods and rules for extracting entity-relationship and object models from relational databases (Chiang *et al.*, 1994; Behm *et al.*, 1997; Malki *et al.*, 2002). Recently, some approaches that consider ontologies as the target for reverse engineering have been proposed. These approaches fall roughly into one of the five categories:

– *Approaches based on an analysis of user queries*: e.g., Kashyap's approach (Kashyap, 1999) 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, which are part of the ontology. Moreover, the semantic characteristics of the database schema are not always analyzed.
– *Approaches based on an analysis of relational schema*: e.g., Stojanovic *et al.*'s approach (Stojanovic *et al.*, 2002) provides a set of rules for mapping constructs in the relational database to semantically equivalent constructs in the ontology. These rules are based on an analysis of relations, keys and inclusion dependencies. Rubin *et al.*'s approach (Rubin *et al.*, 2002) proposes the automation of the process of filling the instances and their attributes' values of an ontology using the data extracted from external relational sources. This method uses a declarative interface between the ontology and the data source, modelled in the ontology and implemented in XML schema. This approach needs several components: ontology, XML schema, and XML translator.
– *Approaches based on an analysis of tuples*: e.g., Astrova's approach (Astrova, 2004) builds an ontology based on an analysis of relational schema. Since the relational schema often has little explicit semantics (Noy and Klein, 2004), this approach also analyzes tuples in the relational database to discover additional "hidden" semantics (e.g., inheritance). However, this approach is very time consuming with regard to the number of tuples in a relational database.
– *Approaches based on an analysis of HTML-table*: e.g., Tijerino's approach (Tijerino *et al.*, 2005) based on conceptual modeling extraction technique attempts to understand a table's structure and conceptual content, discover the constraints that hold between concepts extracted from the table, match the recognized concepts with ones from a more general specification of related concepts, and merge the resulting structure with other similar knowledge representations. However, this approach requires auxiliary informations including dictionaries and lexical data (i.e., WordNet, Natural language parsers, and data frames library).
– *Approaches based on an analysis of HTML-forms*: e.g., Astrova's approach (Astrova and Stantic, 2005) constructs an ontology based on an analysis of HTML-forms by analyzing the HTML-forms to extract a form model schema, transforming the form model schema into ontology and creating ontological instances from data contained in the pages. The main drawback of this approach is that it does not offer any way to identify inheritance relationship.

3. Our Approach

To overcome the drawbacks of the approaches described above, we propose a novel approach for reverse engineering data-intensive web sites into ontology-based semantic web. Our approach is based on the idea that semantics of the relational database can be extracted by analyzing the related HTML pages. This semantics is augmented with those captured in the relational schema to build ontology. Unlike (Benslimane *et al.*, 2005) that uses Frame Logic, as an ontology description language, this paper adopts the latest standard recommended by W3C, namely OWL (Ontology Web Language).

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 (Choobineh *et al.*, 1992).
– 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 (Muller, 1999).
– 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.

3.2. *Proposed Architecture*

This section describes the ontology building framework. It gives a description of the architecture components (see Fig. 1).

The Extraction Engine consists of tree sets of extraction rules. The first set of rules analyses the HTML pages to identify constructs in the form model schema. The second set of rules permits the extraction of a form XML schema from the constructs of the form model schema, whereas the third set of rules derives the domain semantics by extracting the relational sub-schemas of forms and their dependencies.

The Transformation Engine consists of two sets of transformation rules. The first set of rules transforms the relational sub-schemas of forms into an UML class diagram. The second set of rules translates the obtained diagram into OWL ontology.

The Migration Engine consists of a set of data migration rules responsible of the creation of ontological instances from the relational tuples.

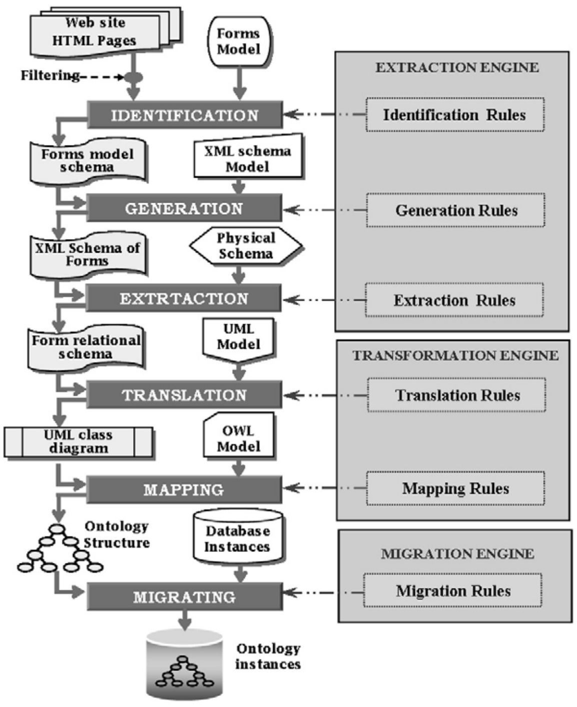


Fig. 1. Ontology building framework.

4. Reverse Engineering Process

Our approach articulates around six steps performed by the six set of rules. We'll use the HTML pages in Fig. 2 to illustrate these steps. This displays a query result for booking flight at British airlines company Web site[1].

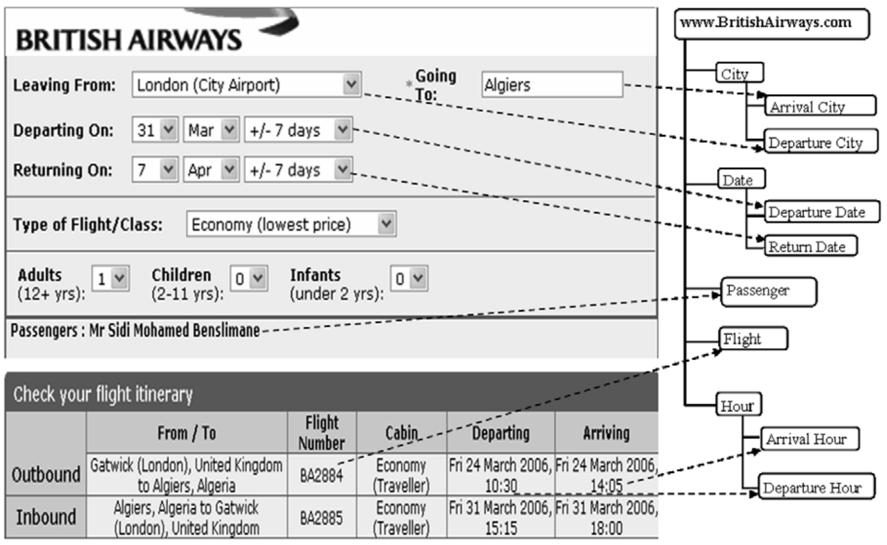
4.1. *Analysis of HTML Pages Structure*

The main goal of this phase is to understand the form meaning and explicit its structure by analyzing HTML forms (both their structure and data they contain) to identify its components and interrelationships and extract a form model schema.

4.1.1. *The Form Model*

A form model schema was originally proposed, suitable for databases reverse engineering task (Malki *et al.*, 1999). The model (see Fig. 3) allows abstracting any database form to make explicit its components, fields, and their interrelationships. This model is similar but not identical to the models presented in (Choobineh *et al.*, 1992). Basically, this model consists of:

[1]`http://www.BritishAiways.com`


BRITISH AIRWAYS

Leaving From:	London (City Airport)	Going To:	Algiers
Departing On:	31 Mar +/- 7 days		
Returning On:	7 Apr +/- 7 days		
Type of Flight/Class:	Economy (lowest price)		
Adults (12+ yrs): 1	Children (2-11 yrs): 0	Infants (under 2 yrs): 0	

Passengers : Mr Sidi Mohamed Benslimane

Check your flight itinerary

	From / To	Flight Number	Cabin	Departing	Arriving
Outbound	Gatwick (London), United Kingdom to Algiers, Algeria	BA2884	Economy (Traveller)	Fri 24 March 2006, 10:30	Fri 24 March 2006, 14:05
Inbound	Algiers, Algeria to Gatwick (London), United Kingdom	BA2885	Economy (Traveller)	Fri 31 March 2006, 15:15	Fri 31 March 2006, 18:00

Fig. 2. HTML pages along with HTML-form and HTML-table.

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. Fig. 2 shows two instances of *Booking* and *flight itinerary* 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, CHECKBOX, 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 in-

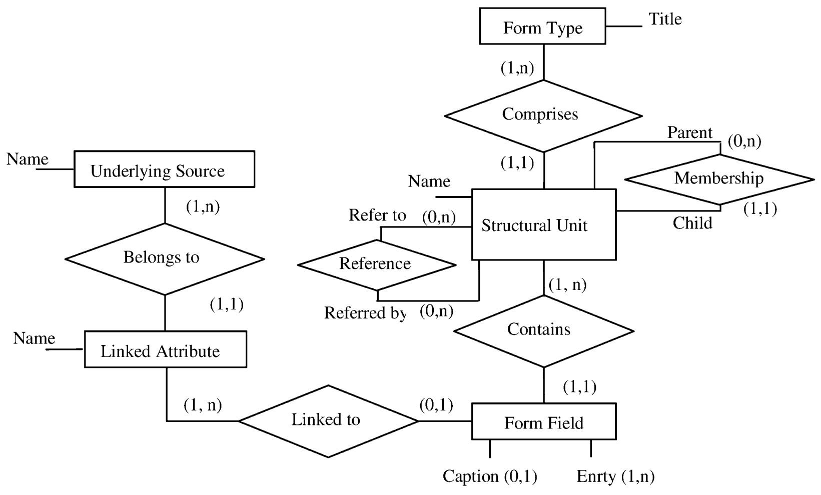


Fig. 3. Extended E/R schema of a form.

stance, a cardinality constraint specifies for an association relationship the number of instances that a SU can participate in.

4.1.2. *Form Schema Identification Rules*

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 Fig. 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;
- 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>` (Tijerino *et al.*, 2005).

If the linked attributes are not separated with the structural tags (merged data), we use visual cues (Yang and Zhang, 2001; Wang and Lochovsky, 2003). 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 (Yang and Zhang, 2001). E.g., users might consider $FirstName$, $LastName$, and Age in Fig. 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. By clicking on a hyperlink in one structural unit (at some page), we can go to another structural unit (possibly at another page).

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.

For example, in Fig. 2 we could identify a constraint $NotNull$ on the linked attributes $DepartureCity$ and $ArrivalCity$.

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.

Table 1

XML schema of *Booking* Form

```
<?xml version="1.0"?>
<xsd:schema=xmlns:xsd="http://www.w3.org/2001/XMLSchema>
    <xsd:complexType name="BookingForm">
    <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/>
```

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 1.

4.3. *Extraction of the Domain Semantics*

The goal of this phase of extraction is to derive the relational sub-schemas of forms from their hierarchical structure and their instances according to the physical schema of the underlying database. First, the relations and their primary keys are respectively identified with regard to both structural units (nodes) of form and underlying database, then the functional and inclusion dependencies are extracted through both their hierarchical structure and instances.

4.3.1. *Form Relations Extraction*

The forms either permit the updating of relations in underlying database or represent a view that is a joint of relations. Therefore, each field entry is generally linked to an attribute of one relation in the underlying database. However, the identification of form relations and their primary keys respectively, consists of determining the equivalence and/or the similarity between structural units (nodes) of hierarchical structure and relations in the underlying database. This is a basis point from a reverse engineering point of view (Malki *et al.*, 2002). A node of a 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.

In addition, for dependent nodes (or form relation), primary keys are formed by concatenating the primary key of its parent with its local primary key.

This process of identification is semi-automated because it requires the interaction with the analyst to identify objects that do not verify proprieties of equivalence and similarity.

While applying this process on the hierarchical structure of *Booking* form and the physical relational schema of underlying database, we extract the following relational sub-schemas:

```
City(CityID, CityName)
Passenger(PassengerID, FirstName, LastName, Age)
DepartureCity(DepartureCityID)
ArrivalCity(ArrivalCityID)
Date(DepartureDate)
```

From the *Flight itinerary* form, the following relational sub-schema is extracted:

```
DepartureHour(HourID, type)
ArrivalHour(HourID, type)
Plane(PlaneID, capacity)
Flight(FlightID, DepartureCityID, ArrivalCityID,
DepartureHourID, ArrivalHourID, PlaneID)
```

From the relationships between the hierarchical structure of *Booking* and *Flight itinerary* forms, the following relational sub-schema is identified:

```
Book(PassengerID, FlightID, DepartureDate, Class).
```

4.3.2. *Functional Dependencies Extraction*

The extraction of functional dependencies from the extension of database has received a great deal of attention (Anderson, 1994; Mannila and Raiha, 1994; Petit *et al.*, 1995). In our approach we use the algorithm introduced by (Malki *et al.*, 2002) to reduce the time for exacting functional dependencies by replacing database instances with a more compact representation that is, the form instances.

While applying this algorithm on the sub-schema of *Flight itinerary* forms and their instances, one finds the FDs:

PlaneID $\rightarrow$ FlightID.

4.3.3. *Inclusion Dependencies Extraction*

In our approach, we formulate possible inclusion dependencies between relations' key of relational sub-schema of form. The time of this process is more optimized with regard to the other approaches (Petit *et al.*, 1995; Chiang *et al.*, 1994) because the possible inclusion dependencies are verified by analyzing the form extensions which are more compact representation with regard to the database extension.

In this algorithm, attributes of dependencies are the primary keys and foreign keys. Thus, the time complexity is reduced to the test of the inclusion dependency on the form instances. The set of the inclusion dependencies extracted is:

```
DepartureCity.DepartureCityID ≺≺ City.CityID
ArrivalCity.ArrivalCityID ≺≺ City.CityID
```

4.4. *Transforming the Relational Sub-Schema of Form into Object-Oriented Sub-Schema*

The task of conceptual modelling plays a crucial role in the process of information systems development. Conceptual models translate and specify the main data requirements of the user requirements in an abstract representation of selected semantics about some aspects of a real-world domain. Systems analysts seek to capture and represent all relevant problem domain entities and their relationships. In addition, conceptual modelling languages and notations were introduced to represent conceptual models using a collection of modelling elements.

4.4.1. *The Transformation Process*

The transformation is usually a collection of mapping rules that replace constructs in the form relational schema with (semantically equivalent) conceptual entities in the UML class diagram (see Fig. 4). Our rules are similar to those used in (Malki *et al.*, 2002) to perform a transformation into an object oriented model. Basically, the process uses the constructs generated from the precedent step as the main input (i.e., form relational schema, functional dependency and inclusion dependency). It goes through four steps:

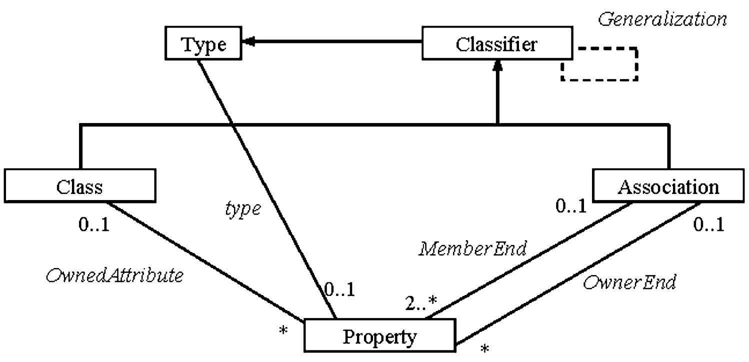


Fig. 4. Key aspects of UML class diagram.

(1) Identification of classes, (2) Identification of binary association, (3) identification of n-ary association, (4) identification of inheritance relationships.

This process is based on the classification of relations. Relation can be classified into one of the three categories.

Base relation: if a relation is independent of any other relation in a form relation schema.

EXAMPLE. `Passenger(PassengerID, FirstName, LastName, Age)`

Dependent relation: if a primary key of a relation depends on another relation's primary key.

EXAMPLE. `Book(PassengerID, FlightID, DepartureDate, Class)`

Composite relation: if it is neither base nor dependent.

EXAMPLE. `Flight(FlightID, DepartureCityID, ArrivalCityID, DepartureHourID, ArrivalHourID)`

Rule *id1*: Identification of object class. The general assumption is that each base relation is mapped into an object class. These object classes have the same attributes as those contained in the relations. The relation `Passenger(PassengerID, FirstName, LastName, Age)` is translated to class shown if the Fig. 5.

Rule *id2*: Identification of binary association. The foreign keys of class-relation and the corresponding functional dependencies identify a binary association between class-relations. Therefore, this referential link is translated in binary association in the UML class diagram. The target will be, in general, a role attribute typed by the other class.

While applying this transformation rule on the two class-relations $Flight$ and $DepartureCity$ and their functional dependencies:

`FlightID` $\rightarrow$ `DepartureCityID`,

we generate the following object schema (see Fig. 6).

Rule *id3*: Identification of association class. For every n-airy class-relation whose primary key is entirely composed of foreign keys, we create an association class between all the classes corresponding to the class-relation that foreign keys refer to.

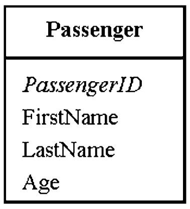


Fig. 5. UML class.

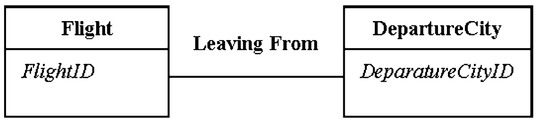


Fig. 6. Binary UML association.

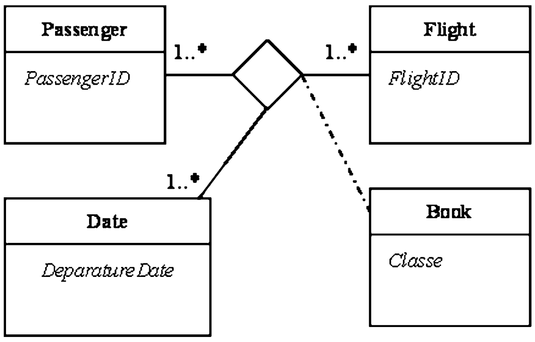


Fig. 7. N-ary UML association.

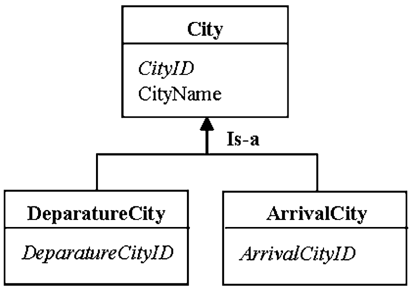


Fig. 8. Inheritance relationship.

The relation `Book(PassengerID, FlightID, DepartureDate, Class)` is translated into Association-class as show in Fig. 7.

Rule ***id4*****: Identification of inheritance relationships.** Extracting inheritance relationship from a relational schema usually requires behavioural information. Every pair of relations $(R1, R2)$ that have the same primary key (noted X) and the corresponding inclusion dependencies (i.e., $R1.X \prec\prec R2.X$) may be involved in an inheritance relationship, i.e., $R1$ *is-a* $R2$.

In Fig. 8, the Relations $City, DepartureCity$ and $ArrivalCity$ have the same primary key and the corresponding inclusion dependencies:

`DepartureCity.DepartureCityID` $\prec\prec$ `City.CityID`

`ArrivalCity.ArrivalCityID` $\prec\prec$ `City.CityID`

Therefore $City$ is a superclass and $DepartureCity$ and $ArrivalCity$ are a subclass.

4.4.2. *Integration of Object-Oriented Sub-Schemas*

In the precedent phase of reverse engineering using forms as machine-analyzable source, relational sub-schemas were transformed into object oriented sub-schemas. These object sub-schemas will be merging into a global object-oriented schema that represents the whole underlying database. However, we apply the techniques of integration schema. We assume, in agreement with (Batini *et al.*, 1986) that the integration schema process consists in two phases: comparison and merging of schemas.

The comparison phase performs a parities comparison of objects (of the sub-schemas) and finds possible objects pairs, which may be semantically similar with respect to some proprieties, such as synonyms (name of attribute and class) of equal primary key attribute and equivalent of classes. The merging phase generates an integrated schema from two component schemas that have been compared. The intermediate results are analyzed and restructured in order to eliminate the symmetrical and transitive relationship between objects. For more details see (Malki *et al.*, 2002).

Fig. 9 presents the integrated schema as an UML class diagram.

4.5. *UML into OWL Mapping Rules*

UML conceptual models can be translated into other ontology languages like RDFS, DAML, OWL or even in to object oriented database systems. Some proposals for defining this transformation have been addressed. Cranefield in (Cranefield, 2001) has proposed mappings to transform UML ontology models in to RDF and to generate Java classes

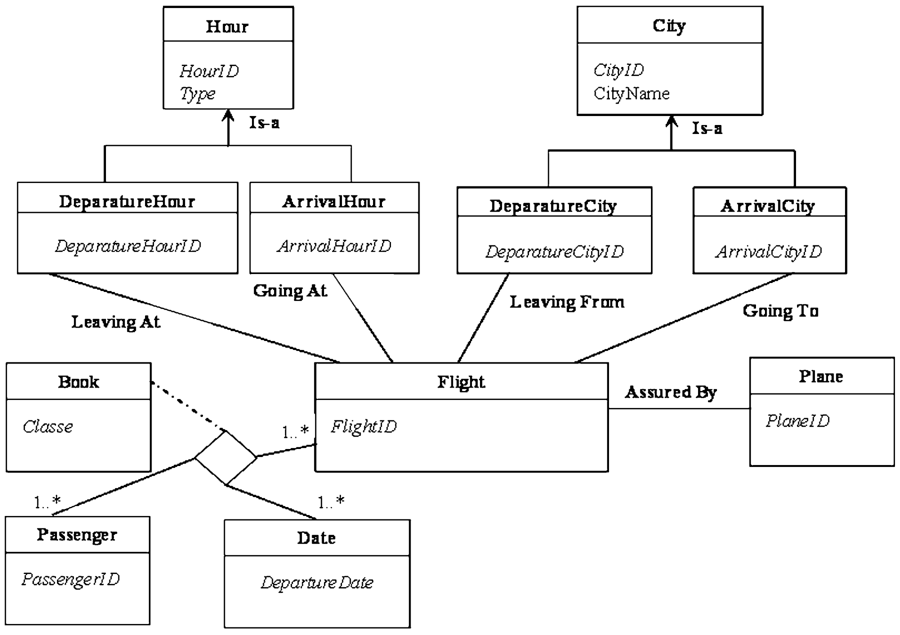


Fig. 9. UML class diagram.

from UML using XSLT. Baklawski (Baclawski *et al.*, 2001) has presented some mappings for translating in between DAML and UML concepts and from UML to DAML. Various approaches concerned with conversion between UML and OWL exist. These are summarised by Falkovych *et al.* (2003), which addresses the problem of reusing knowledge previously specified as UML in a form that allows it to be 'on the Web' and can be reasoned with.

The rules below briefly summarise the transformation rules used in the mapping between UML and OWL constructs.

Rule *mp1***.** 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. `<owl: class rdf: ID="Passenger"/>`

Rule *mp2***.** Both OWL and UML are based on classes. So, in order to translate the UML class of Fig. 5, an OWL class is declared by assigning a name to the relevant type.

EXAMPLE. `<owl: class rdf: ID="Passenger"/>`

Rule *mp3***.** By default a property is a binary relation between thing and thing. It comes from two different sources in the UML class diagram:

– First, an instance of class ownedAttribute Property would translate as properties whose domain is Class and whose range is the type of Property. The UML ownedAttribut instance would translate to $owl : ObjectProperty$ if the type of Property were a UML class, and $owl : DatatypeProperty$ otherwise.
  Table 2 show the translation of classes in Fig. 6.
– Second an instance of a binary UML association translates directly to an $owl : ObjectProperty$.
  The translation of the binary association of the Fig. 6 is given in Table 3.

Table 2
Classes translation

UML class	Owned Property	Type of Owned Property	OWL equivalent
LeavingFrom	FlightID	Integer	<owl:DatatypeProperty rdf:ID="FlightID"> <rdfs:domain rdf:resource="#Flight"/> <rdfs:range rdf:resource="http://.../XMLSchema# integer"/> </owl:DatatypeProperty>
DepartureCity	DepartureCityID	Integer	<owl:DatatypeProperty rdf:ID="DepartureCityID"> <rdfs:domain rdf:resource="#DepartureCity> <rdfs:range rdf:resource="http://.../XMLSchema# integer"/> </owl:DatatypeProperty>

Table 3

Binary association translation

UML Association	Member 1 Property Type	Member 2 Property Type	OWL equivalent
Flight	FlightID	Integer	<owl:objectProperty rdf:ID="LeavingFrom"> <rdfs:domain rdf:resource="#Flight"/> <rdfs:range rdf:resource="#DepartureCity"/> </owl:objectProperty>

Rule *mp4***.** N-ary relation among types $T_1 \ldots T_n$ is formally equivalent to a set R of identifiers together with N projection functions $P_1, \ldots, P_n$, where P_i: $R \rightarrow T_i$. Thereby N-ary UML associations are translated to OWL classes with bundles of binary functional properties. For an N-ary UML association, any multiplicity associated with one of its UML properties will apply to the OWL property translating the corresponding projection.

The N-ary association in Fig. 7 would be translated as shown in Table 4.

Rule *mp5***.** In UML, a class can exist as a generalisation for one or more other classes. The generalisation element is synonymous with the $OWL : subClassOf$ construct. The inheritance relationship in the Fig. 8 is translated as shown in Table 5.

Rule *mp6***.** 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 UML an association can have minimum and maximum cardinalities (multiplicity) specified for any of its ends. So if a binary UML association has a multiplicity on a navigable end, the corresponding OWL property will have the same multiplicity. If a binary UML association has a multiplicity on its both ends, then the corresponding OWL property will be an inverse pair, each having one of the multiplicity declarations.

4.6. *Data 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. According to the metadata-level mapping rules (e.g., concepts, properties, restrictions), the tuples of the relation can be transferred to the instances for data exchanging. The data migration process has to be performed in two phases based on the following rules:

Rule *m1*. 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 meta-data.

Table 4
OWL classes

```
<owl:Class rdf:ID="book">
    <owl:subClassOf>
        <owl:Restriction>
      <owl:onProperty rdf:resource=" bookPassenger"/>
      <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">1
      </owl:minCardinality>
        </owl:Restriction>
    </owl:subClassOf>
    <owl:subClassOf>
        <owl:Restriction>
      <owl:onProperty rdf:resource=" bookFlight"/>
      <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">1
      </owl:minCardinality>
        </owl:Restriction>
    </owl:subClassOf>
    <owl:subClassOf>
        <owl:Restriction>
      <owl:onProperty rdf:resource=" bookDate"/>
      <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">1
      </owl:minCardinality>
        </owl:Restriction>
    </owl:subClassOf>
...
</owl:Class>
```

Table 5
OWL inheritance

```
<owl:class rdf:about="DepartureCity">
   <owl:subClassOf rdf:resource="#City" />
</owl:class>
<owl:class rdf:about="ArrivalsCity">
   <owl:subClassOf rdf:resource="#City" />
</owl:class>
```

Rule *m2.* 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.

Table 7, illustrates an example result of the data migration process from the relational database instances of Table 6.

5. 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.

Table 6

Relational database instances

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

Table 7

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://.../XMLSchema# int">1</CompanyId>
    <CompanyName rdf:datatype="http://.../XMLSchema# string">
      Air Algerie</CompanyName>
  </Company>
  <Plane rdf:ID="Plane1">
    <capacity rdf:datatype="http://.../XMLSchema# int">150</capacity>
    <PlaneId rdf:datatype="http://.../XMLSchema# string">A330</PlaneId>
    <Possede rdf:resource="#Company1"/>
  </Plane>
  <Company rdf:ID="Company2">
    <CompanyName rdf:datatype="http://.../XMLSchema# string">
      Air France</CompanyName>
    <CompanyId rdf:datatype="http://.../XMLSchema# int">2</CompanyId>
  </Company>
  ...
  </Plane>
</rdf:RDF>
```

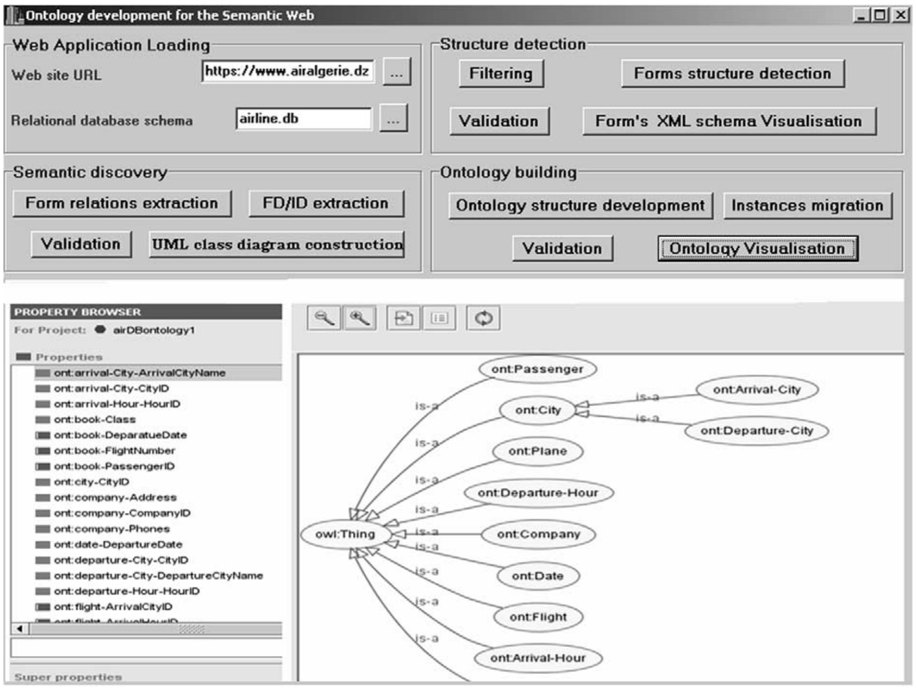
5.1. *Prototype*

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 (Fig. 10) 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 construct the global UML class diagram describing the Web application. The ontology transformation engine (Fig. 1) processes the UML class diagram and generates the corresponding OWL ontology based on the mapping rules previously described. The output ontology can be formalized in the following standard formats: OWL, RDF/XML, RDF/XML-ABBREV, N3 and N-Triples.

5.2. *Experimental Evaluation*

In order to evaluate our approach, we performed two experiments on tourism domain.



Web site URL https://www.airalgerie.dz

Relational database schema airline.db

For Project: airDBontology1

ont:arrival-City-ArrivalCityName
ont:arrival-City-CityID
ont:arrival-Hour-HourID
ont:book-Class
ont:book-DeparatueDate
ont:book-FlightNumber
ont:book-PassengerID
ont:city-CityID
ont:company-Address
ont:company-CompanyID
ont:company-Phones
ont:date-DepartureDate
ont:departure-City-CityID
ont:departure-City-DepartureCityName
ont:departure-Hour-HourID
ont:flight-ArrivalCityID

Fig. 10. Snapshot of the ontology development tool.

Table 8

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 prop.	16	09	1	0.56	0.90
Datatype prop.	77	34	3	0.44	0.92

In the first experiment, we analyzed an airlines company Web site[2]. The constructs of the obtained OWL ontology are presented in Table 8. The results are compared to the tutorial ontology for a Semantic Web of tourism[3]. 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, the following quality measures are computed.

$Precision = C/(C+I)$, 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[4], hotel[5], and leisure[6] tourism activities. The ontology development process was rather successful, with average recall and precision ratios of 94% and 92% respectively (see Table 9).

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.

6. Conclusions

In this paper we have presented a novel, integrated, and semi-automated approach for extracting personalised ontology from data-intensive Web applications that can be applied to a broad range of today's business Web sites, especially those that are dynamically

[2]http://www.britishairways.com.

[3]http://protege.stanford.edu/plugins/owl/owl-library/travel.owl.

[4]http://www.britishairways.com.

[5]http://www.hm-usa.com.

[6]http://www.travelandleisure.com.

Table 9

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 prop.	16	15	2	0.94	0.88
Datatype prop.	77	73	5	0.95	0.94

generated from a relational database. Instead of creating the ontology manually we have proposed an approach to build the ontology as automatically as possible. The approach starts with transforming the HTML-forms into a form model schema. This model, allows the generation of an XML schema, witch permit the extraction of domain semantics and the construction of an UML class diagram. A mapping process is done to translate the UML class diagram into OWL ontology. Finally, ontological instances are created based on the tuples of the relational database.

It can be seen that the proposed approach is practical and helpful to reduce the time consuming task of ontology creation, and to make the relational database information that is available on the Web machine-processable.

However, in the most circumstances, the obtained ontological structure is coarse. Because many constraints, relationships and other semantics in relational database are implicit, or even lacking, the ontology extracted from relational database is not complete in semantics, and need to be validated by experts, which depends on the domain knowledge and experiences. So refining obtained ontological structure is necessary. Existing repositories of lexical knowledge usually includes authoritative knowledge about some domains, we suggest as future work refining obtained ontology according to them, especially machine-readable dictionaries and thesauri.

References

Anderson, M. (1994). Extracting an entity-relationship schema from a relational database through reverse engineering. In *Proceeding of the 13th International Conference on Entity-Relationship Approach*. pp. 403–419.

Astrova, I. (2004). Reverse engineering of relational databases to ontologies. In *Proceeding of the 1st European Semantic Web Symposium* (*ESWS*), Heraklion, Greece. *LNCS*, vol. 3053. pp. 327–341.

Astrova, I., and B. Stantic (2005). An HTML forms driven approach to reverse engineering of relational databases to ontologies. In *Proceeding of the 23rd IASTED International Conference on Databases and Applications* (*DBA*), Innsbruck, Austria. pp. 246–251.

Baclawski, K., M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski and M. Aronson (2001). Extending UML to support ontology engineering for the semantic web. In *Proceeding of the Fourth International Conference on UML*, Toronto. pp. 342–360.

Batini, C., M. Lenzerini and S.B. Navathe (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, **18**(4), 323–364.

Behm, A., K. Geppert and K. Dittrich (1997). On the migration of relational schemas and data to object-oriented database systems. In *Proceeding of the 5th Int. Conference on Re-Technologies for Information Systems*, Klagenfurt. pp. 13–33.

Benslimane, S.M., M. Malki and D. Amar Bensaber (2005). Automated migration of data-intensive web pages into ontology-based semantic web: a reverse engineering approach. In R. Meersman *et al.*, (Eds.), *ODBASE*. Springer Verlag, vol. 2. *LNCS* 3761. pp. 1640–1649.

Chiang, R.H.L., T.M. Barron and V.C. Story (1994). Reverse engineering of relational databases: extraction of an EER model from a relational database. *Data and Knowledge Engineering*, **10**(12), 107–142.

Choobineh, J., M.V. Mannino and V.P. Tseng (1992). A form-based approach for database analysis and design. *Communication of the ACM*, **35**(2), 108–120.

Cranefield, S. (2001). UML and the Semantic Web. In *Proceeding of the International Semantic Web Working Symposium*, Palo Alto. pp. 18–29.

Embley, D. (2004). Toward semantic understanding – an approach based on information extraction. In *Proceeding of the 15th Australasian Database Conference* (*ADC*), Dunedin, New Zealand. pp. 3–12.

Erdmann, M., A. Maedche, H. Schnurr and S. Staab (2000). From manual to semi-automatic semantic annotation: about ontology-based text annotation tools. In P. Buitelaar and K. Hasida (Eds.), *Proceedings of the Workshop on Semantic Annotation and Intelligent Content* (*COLING*), Luxembourg. pp. 233–230.

Falkovych, K., M. Sabou and H. Stuckenschmidt (2003). UML for the Semantic Web: Transformation-Based Approaches. Knowledge Transformation for the Semantic Web, IOS Press, Amsterdam.

Fraternali, P. (1999). Tools and approaches for developing data-intensive web applications: a survey. *ACM Computing Surveys*, **31**(9), 227–263.

Gruber, T.R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *Human Computer Studies*, **43**(5–6), 907–928.

Kashyap, V. (1999). Design and creation of ontologies for environmental information retrieval. In *Proceeding of the 12th Workshop on Knowledge Acquisition, Modelling and Management* (*KAW*), Banff, Alberta, Canada. pp. 3–21.

Malki, M., M. Ayache and M.K. Rahmouni (1999). Rétro-ingénierie des bases de données relationnelles : approche basée sur l'analyse de formulaires. In *Actes du XVIIème Congrès INFORSID*, Toulon, France. pp. 340–348.

Malki, M., A. Flory and M.K. Rahmouni (2002). Extraction of object-oriented schemas from existing relational databases: a form-driven approach. *Informatica*, **13**(1), 47–72.

Mannila, H., and K.J. Raiha (1994). *The Design of Relational Databases*. Addison-Wesley publishing. Boston, MA, USA.

Muller, R.J. (1999). *Database Design for Smarties: Using UML for Data Modeling*. Morgan Kaufmann.

Noy, N., and M. Klein (2004). Ontology evolution: not the same as schema evolution. *Knowledge and Information Systems*, **6**(4), 428–440.

Petit, J.M., F. Toumani and J. Kouloumdjian (1995). Relational database reverse engineering: a method based on query analysis. *International Journal of Cooperative Information System*, **4**(2,3), 287–316.

Rubin, D.L., M. Hewett, D.E. Oliver, T.E Klein and R.B. Altman (2002). Automatic data acquisition into ontologies from pharmacogenetics relational data sources using declarative object definitions and XML. In R.B. Lihue *et al.* (Eds.), *Proceedings of the Pacific Symposium on Biology*. pp. 22–34.

Stojanovic, L., N. Stojanovic and R. Volz (2002). Migrating data-intensive web sites into the semantic web. In *Proceeding of the 17th ACM Symposium on Applied Computing* (*SAC'2002*), Madrid, Spain. pp. 1100–1107.

Tijerino, Y.A, D.W. Embly, D.W. Lonsdale, Y. Ding and G. Nagy (2005). Towards ontology generation from tables. **8**(3), 261–285.

Volz, R., S. Handschuh, S. Staab, L. Stojanovic and N. Stojanovic (2004). Unveiling the hidden bride: deep annotation for mapping and migrating legacy data to the semantic Web. *Journal of Web Semantics*: *Science, Services and Agents on the Word Wide Web*, **1**(2), 187–206.

Wang, J., and F. Lochovsky (2003). Data extraction and label assignment for web databases. In *Proceeding of the 12th International Conference on World Wide Web*, Budapest, Hungary. pp. 187–196.

Yang, Y., and H. Zhang (2001). HTML page analysis based on visual cues. In *Proceeding of the 6th International Conference on Document Analysis and Recognition* (*ICDAR*), Seattle, USA. pp. 859–864.

S.M. Benslimane is PhD candidate in computer science Department at Sidi Bel Abbes University from December 2002. He received the MS degree in computer science from Sidi Bel Abbes University, Algeria, in 2001. Starting from 2001, he is lecter in the Department of Computer Science, Sidi Bel Abbes University, Algeria. His research interests include, semantic Web, Web engineering, ontology engineering, knowledge management, information systems.

M. Malki is an assistant professor at the Department of Computer Science at Sidi Bel Abbes University. He received the PhD degree in computer science from Sidi Bel Abbes University, Algeria, in 2003. He heads the Evolutionary Engineering and Distributed Information Systems Laboratory. His research interests include, knowledg management, information retrieval, ontology engineering, semantic Web, Web services, and soft computing systems.

M.K. Rahmouni is a professor at the Computer Science Department of the University of Oran Es-Sénia, Algeria. He received the PhD degree in operational research from Southampton University UK, in 1987. He heads the Information Systems Laboratory and the local Doctoral School on STIC. His research interests include formal specifications, information management and integration, process modelling, and knowledge management

D. Benslimane is graduated with an "Ingenieur Informatique" degree in 1985 from the university of Tizi-ouzou, Algeria. He received his PhD degree in computer science from Blaise Pascal University in Clermont-Ferrand, France, in 1992. He was an assistant professor at University of Burgundy, Dijon, France from 1992–2001. He is currently a professor at Claude Bernard University, Lyon, France. His principals research interests include, databases, interoperability, ontology, context, web services, query processing.

Personifikuotos ontologijos gavimas iš daug duomenų apdorojančių interneto programų: atvirkštine HTML formų inžinerija grindžiamas metodas

Sidi Mohamed BENSLIMANE, Mimoun MALKI,
Mustapha Kamal RAHMOUNI, Djamal BENSLIMANE

Interneto vystymasis ženkliai pakeitė informacijos tvarkymo ir platinimo būdus. Naujoji interneto karta, grindžiama semantinio tinklo idėjomis, leis taikyti griežtesnį, ontologijomis grindžiamą informacijos struktūrą, taip pagerinant informacijos panaudojamumą kompiuteriniame lygmenyje. Būtent šiame kontekste svarbus autorių siūlomas novatoriškas integruotas sprendimas, leidžiantis pusiau automatiškai formuoti ontologija grindžiamą semantinį tinklą iš intensyviai duomenis naudojančių interneto programų. Tokiu būdu tinklo turinys taptų „suprantamu“ kompiuteriams. Straipsnyje pristatoma metodika grindžiama idėja, jog semantika gali būti išgaunama taikant atvirkštinį HTML formų inžineriją. Autoriai priima prielaidą, jog šių formų reinžinerija yra patogiausias šiuolaikinėse interneto programose naudojamų reliacinių duomenų bazių analizės būdas. Straipsnyje apibrėžti etapai, kuriais iš gautosios semantikos gali būti sudaroma personifikuota ontologija.