Be Careful When Designing Semantic Databases: Data and Concepts Redundancy

Chedlia Chakroun
LIAS/ISAE-ENSMA
Futuroscope, France
chakrouc@ensma.fr

Ladjel Bellatreche
LIAS/ISAE-ENSMA
Futuroscope, France
bellatreche@ensma.fr

Yamine Aït-Ameur
IRIT - ENSEEIHT
Toulouse, France
yamine@enseeiht.fr

Nabila Berkani
ESI
Algiers, Algeria
n_berkani@esi.dz

Stéphane Jean
LIAS/ISAE-ENSMA
Futuroscope, France
jean@ensma.fr

***Abstract*—Recently, ontologies have been widely adopted by small, medium and large companies in various domains. These ontologies may contain redundant concepts (computed from primitive concepts). At the beginning of the development of ontologies, the relationship between them and the database was weakly coupled. With the explosion of semantic data, persistent solutions to ensure a high performance of applications were proposed. As a consequence, a new type of database, called semantic database ($\mathcal{SDB}$) is born. Several types of $\mathcal{SDB}$ have been proposed and supported by different DBMS, where each one has its architecture and its storage layouts for ontologies and its instances. At this stage, relationship between databases and ontologies becomes strongly coupled. As a consequence, several research studies were proposed on the physical design phase of $\mathcal{SDB}$. To guarantee the similar success that relational databases got, $\mathcal{SDB}$ has to be supported by complete design methodologies and tools including the different steps of the database life cycle. Such methodology should identify the redundancy embedded into ontology. In this paper, we propose a design methodology dedicated to $\mathcal{SDB}$ including the main phases of the lifecycle of the database development: conceptual, logical, deployment and physical. The conceptual design of $\mathcal{SDB}$ can be easily performed by exploiting the similarities between ontologies and conceptual models. The logical design phase is performed thanks to the incorporation of dependencies between concepts and properties in the ontologies. These dependencies are quite similar to the principle of functional dependencies defined in the traditional databases. Due to the diversity of the $\mathcal{SDB}$ architectures and the variety of the used storage layouts (horizontal, vertical, binary) to store and manage ontological data, we propose a $\mathcal{SDB}$ deployment *à la carte*. Finally, a prototype implementing our design approach on Oracle 11g is outlined.**

I. INTRODUCTION

Nowadays, ontologies became a complete technology supported by modeling languages such as OWL (the Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things[1], PLIB [1] (a language for the engineering domain), etc., editors (e.g. Protégé), reasoners (e.g., Pellet [2]), evaluation methods, etc. An ontology is defined by Gruber [3] as *an explicit specification of a conceptualization*. Ontologies' key benefit is interoperability. They showed their contributions in various interoperability-sensitive research and industrial domains such as data integration, semantic Web, inference engines, semantic indexation, crawlers, data mining, etc. Chronologically, ontologies have been adopted by three main research and industrial communities: the *linguistic*, the *artificial intelligence* and the *database*. Consequently, each community has its own interpretation and use of the term "ontology". Due to this diversity, Jean et al. [4] propose a taxonomy of ontologies, namely the onion model (left part of the Figure 1). It is composed of three layers: *linguistic* ontologies, *conceptual canonical* ontologies and *non conceptual canonical* ontologies.

Linguistic Ontologies (LOs) are those ontologies whose scope is the representation of the meaning of the words used in a particular universe of discourse, in a particular language. Beyond the textual definitions, a number of linguistics relationships (synonymous, hyponym, etc) are used to capture, in an approximate and semi-formal way, the relation between the words. LOs may be used to localize similarities between source schemas, to document existing databases or to improve the user dialog. Wordnet[2] is an example of such ontologies.

Conceptual Canonical Ontologies (CCOs) contain ontologies which describe concepts of a domain without any *redundancy*. CCOs adopt an approach of structuring of information in term of classes and properties and associate to these classes and properties a single identifiers reusable in various languages. CCOs can be considered as shared conceptual models.

Non Conceptual Canonical Ontologies (NCCOs) represent not only primitive concepts (canonical), but also definite concepts (non-canonical), i.e. which can be defined from primitive concepts and/or other definite concepts by the use of expression languages such as OWL and PLIB. NCCOs provide mechanisms similar to views in databases. Non-canonical concepts can be seen as virtual concepts defined from canonical concepts. These mechanisms may be used to represent mappings between different databases. Some existing ontology models provide users with appropriate builders, functions and procedures to express such definitions. For example, the PLIB model, using the EXPRESS modeling language, offers the possibility to define the derived data and/or object properties across functions. OWL uses different constructors to build non-canonical concepts such as restrictions (e.g, the Man class can be defined as all persons having the 'male' value for the gender property) or Boolean operators (e.g., the Human class can be defined as the union of Man and Woman).

From the database point of views, conceptual ontologies leverage conceptual models proposed by Dr. Peter Chen [5]. Recall that he argued that the world may be represented/modeled by the use of two concepts, named, entity and relationship.

[1]www.w3.org/2001/sw/wiki/OWL

[2]http://wordnet.princeton.edu/

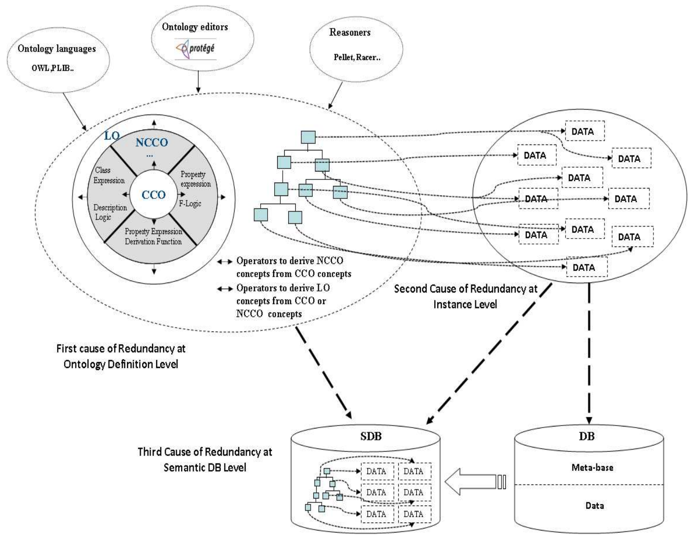

Ontology languages
OWL,PLIB..
Ontology editors
protégé
Reasoners
Pellet, Racer..
LO
NCCO
...
Class Expression
Property expression
CCO
Description Logic
F-Logic
Property Expression Derivation Function
Operators to derive NCCO concepts from CCO concepts
Operators to derive LO concepts from CCO or NCCO concepts
DATA
Second Cause of Redundancy at Instance Level
First cause of Redundancy at Ontology Definition Level
Third Cause of Redundancy at Semantic DB Level
SDB
DB
Meta-base
Data

Fig. 1: Redundancy caused by Ontologies

In addition to conceptual models, ontologies brought two main issues: (1) an ontology may contain non-canonical concepts (concepts derived from other ones), whereas, a conceptual model stores only canonical concepts. (2) An ontology offers the reasoning capabilities. If the second problem has been solved by using external reasoner or processing the reasoning with database mechanisms, the first one has been mostly ignored even if it raises the important issue of introducing redundancy in the database. The following example illustrates this type of redundancy.

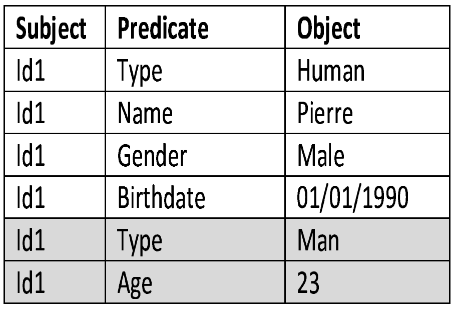
Example 1: Let us consider the canonical class $Human$ domain of the properties $name$, $gender$ and $birthday$. Using the description logic syntax, the non-canonical class Man can be defined with an OWL restriction as follow: $Man \equiv Human \sqcap gender : Male$ The non-canonical property age can be defined with PLIB derivation function: $age = current\_date - birthdate$

The Figure 2 presents the storage of the class $Human$ in the classical triple table (a direct translation of the RDF model). The representation of non-canonical concepts in this table (triples in grey) introduces two anomalies:

- if the gender of the instance is modified, the triple $(Id1, type, Man)$ becomes incorrect.
- if the birthdate of the instance is modified, the triple $(Id1, age, 23)$ becomes incorrect.

Thus the identification of non-canonical concepts is crucial for the development of database applications since it reduces the redundancy and inconsistency of databases constructed from ontologies. Functional dependency between properties of a given class may also be used to reduce redundancy. As a consequence, dependencies between ontological classes and properties are critical when designing databases from ontologies.

Initially, the link between ontologies and databases weakly coupled, because ontologies were used at the conceptual level. The massive use of ontologies generates a big amount



Subject	Predicate	Object
Id1	Type	Human
Id1	Name	Pierre
Id1	Gender	Male
Id1	Birthdate	01/01/1990
Id1	Type	Man
Id1	Age	23

Fig. 2: Example of redundancy in Semantic Database

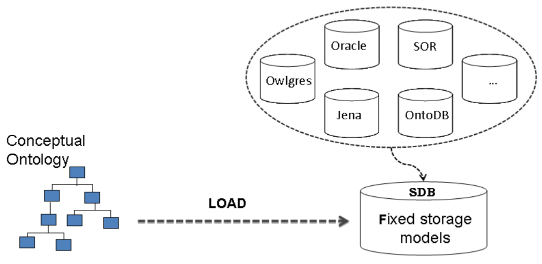


Fig. 3: Ontology storage in $\mathcal{SDB}$.

of semantic data. To facilitate their managements, persistent solutions to store and query these mountains of semantic data were proposed. These gave raise to a new type of databases, called semantic databases ($\mathcal{SDB}$). Academicians and industrials propose a large panoply of $\mathcal{SDB}$ [6], [7], [8], [9], [10], [11], [12]. Relationship between ontologies and databases becomes strongly coupled. Three main architectures of Database Management Systems managing such databases are distinguished. In the first type (that we called $Type_1$), the traditional database architecture was reused to store in the data part both ontologies and data referencing them. This way, the ontology and its associated data are stored in a *unique part*. To make an identified separation between ontology and data, a second type (called $Type_2$) was proposed where ontology and its ontological data are stored independently into *two different schemes*. Therefore, the management of ontology and data parts is different. The $Type_1$ and $Type_2$ architectures hard-coded the ontology model (RDF or RDFS, etc.). To enable the evolution of ontology models, a third database architecture (called $Type_3$) extending the second one by adding a new part, called the *meta-schema part* was proposed [9]. The presence of the meta-schema offers the following characteristics: (1) a generic access to the ontology part, (2) a support of the used ontology model's evolution by adding non functional properties such as preferences [13], web services [14], etc. and (3) a storage of different ontology models (OWL, PLIB, etc.).

When ontologies became a part of databases, several research efforts were mainly concentrated on the physical design phase, where storage layouts dedicated for ontologies and their instances were discussed. Three main storage layouts are distinguished: vertical, horizontal and binary. Vertical representation stores data in a unique table of three columns (subject, predicate, object) (eg. Oracle [8]). In the binary representation, classes and properties are stored in different tables (eg. IBM SOR [10]). The horizontal representation translates each class as a table having a column for each property of the class (eg. OntoDB [9]). Note that each storage layout has its advantages and limitations based on the ontological data and the queries. Nowadays, DBMS proposes fixed storage layouts for its different components (data, meta-schema, ontology model). In real life applications, database administrators may choose the target DBMS based on her/his architectures and storage layouts.

A. *Summary.*

The main studies in $\mathcal{SDB}$ were mainly concentrated on the physical phase of database development. This situation is similar to what we have seen during the development of the database technology. When Prof. Edgar Frank Codd proposed his relational model in 70's [15], the lifecycle of database applications had only two phases: logical and physical phases. In the logical phase, the database schema may be normalized using the functional dependencies defined on properties to reduce the redundancy and ensure the quality of the final database schema. In 1975, when Peter Chen proposed his entity relationship model, he contributed in extending the lifecycle of databases by adding the conceptual phase. Chen's work allows database technology to have its lifecycle. As a consequence, several methodologies were proposed according to this lifecycle: conceptual, logical and physical design phases. We can cite the MERISE method [16], the Unified Process, the Rational Unified Process, Two Tracks Unified Process, etc. These methodologies were supported by academic and industrial tools such as Sybase PowerAMC[3] and Rational Rose[4]. The maturity of $\mathcal{SDB}$ technology requires the development of design methodologies including different phases of database lifecycle. Recall in traditional databases, usually the relational table layout is advocated when deploying a database which makes easier the deployment process. In the $\mathcal{SDB}$, the diversity of storage layouts and the architectures of the target DBMS makes the deployment more complicated. Another aspect that has to be considered is the loading of ontological instances into the $\mathcal{SDB}$ (Figure 3). This problem is quite similar to ETL process (extraction, transformation, loading) [17].

B. *Outline and Contribution*

The contributions of our work are:

- The identification of the causes of redundancies and inconsistencies of $\mathcal{SDB}$ which represent the presence of the non-canonical concepts in ontologies and dependencies between properties.
- The proposition of a formal model for describing conceptual ontologies embedding dependencies between their concepts and properties.
- The presentation of a complete methodology for designing $\mathcal{SDB}$, where deployment (including ETL) is ensured through services.

[3]http://www.sybase.com/products/modelingdevelopment/poweramc

[4]http://www-01.ibm.com/software/awdtools/developer/rose/

The remainder of this paper is organized as follows. Section 2 presents the related works. Section 3 exposes the weak exploration of ontology characteristics related to dependencies relationships. In Section 4, we propose a formalization of ontological dependencies and $\mathcal{SDB}$. Our approach exploiting conceptual dependencies to improve the semantic database process is given in section 5. Section 6 shows an application of our approach to improve the $\mathcal{SDB}$ design methodology in the Oracle 11g. Finally, section 6 gives a conclusion and some future research directions.

II. Related Work

Since the 70s, the functional dependencies have been widely studied in the database theory. These dependencies, usually defined on the attributes, were especially exploited in the databases design process. They are used to model the relationships between attributes of a relation, compute primary keys, define the normalized logical model to avoid redundant data, check the data consistency, etc. For the description logics (DL), functional dependencies have also been the subject of several studies [18], [19], [20], [21], [22], [23]. In [18], Borgida et al. have expressed the need to add unique constraints for semantic data models, particularly for the description logic, while in [19], the authors studied the possibility to make them explicit in this language.

In [24], the authors extend $\mathcal{DLR}$ with identification constraints and functional dependencies. Note that $\mathcal{DLR}$ is an expressive Description Logic with n-ary relations, particularly suited for modeling database schemas. The resulting DL, called DLR_{ifd}, allows one to express these constraints through new kinds of assertions in the TBox. For example, a functional dependency assertion has the form ($fd\ R\ i_1$,..., $i_h \rightarrow j$) where R is a relation, h$\geq$2 and i_1,...,i_h,j denote components of R. In [20], Motik et al. showed the role of constraints in the ontologies while drawing a comparison between the constraints in databases and those in ontologies.

In [21], [22], the authors were interested in the study of dependency relationships and their implications in ontologies. In [22], the authors propose a new OWL constructor to define functional dependencies. They describe a FD by the following quadruplet $FD = (A, C, R, f)$, where:

- A is the antecedent i.e. is a list of paths (A = $\{u_1, u_2, .., u_n\}$). A path u_i is in turn composed of roles r_i (r_i (u_i=$\{r_{i,1}, r_{i,2}, .., r_{i,n}\}$).
- C is a consequent composed by a single path (C = $\{u\}$).
- R is a root concept representing the starting point of all paths in the antecedent and consequent.
- f a deterministic function which takes as parameters the ranges of the last roles of the antecedent paths.

These dependencies are then translated into a set of SWRL rules for the reasoning process.

In [21], the authors propose an approach to define functional dependencies for a domain ontology based on the concepts and roles defined in such ontology. They propose an algorithm for the identification of a set of FD exploiting the capabilities of DL_{LiteA} reasoning. Note that the DL_{LiteA} is a variant of the DL-Lite family which is a new description logic specially adapted to capture basic ontology languages. The proposed algorithm computes (i) a set of functional dependencies between ontological classes and (ii) the transitive closure. Romero et al. [21] introduce a functional dependency as a relationship between classes. For two concepts C_1 and C_2, the authors established that each instance $i_1 \in C_1$ determines a single instance of C_2 if (1) there exists a functional role (r_i) valued for i_1 and (2) r_i connects i_1 to a unique instance of C_2. This dependency relationship is denoted by $C_1 \rightarrow C_2$. These dependencies are then exploited during the data warehouse design process to make the logical schema definition automated.

In parallel, semantic databases, storing in the same repository ontological data and the ontology describing their meanings, have been introduced. To support such a database, several architectures have been proposed [6], [7], [8], [25], [10], [11], [12]. They have been mainly focused on the scalability of these databases. Considering the support of ontology, each $\mathcal{SDB}$ supports the semantics of a given ontology model using hard coded techniques either by using database mechanisms or by relying on an external logical engine. Therefore, the $\mathcal{SDB}$ may contain anomalies like redundant and inconsistent data. For example, let us assume that the individual i_1 is instance of C_1 and C_2. Considering the storage process, i_1 description is duplicated in the $\mathcal{SDB}$ and a redundancy case is raised. Unfortunately, there is no available methodology dedicated to the $\mathcal{SDB}$ design and avoiding these anomalies.

To summarize, we can easily say that studies on ontological functional dependencies have been mainly focused on objects properties (properties relating concepts to concepts). In the other word, functional dependencies are defined at the ontology level. A few studies on dependencies between data properties (properties having as a range a simple type) and the defined classes exist. One of the main objectives of our paper is to study the characteristics of ontologies in order to identify dependencies between properties and classes and exploit them to propose a consistent $\mathcal{SDB}$. Such characteristics are described in details in the next section.

III. Weak exploration of ontology features

A long this paper, we concentrate on conceptual ontologies that offer two types of dependencies: (i) the dependency relationships between the ontological properties and (ii) the dependency relationships between ontological classes.

A. Dependency relationships between properties

In an ontology, two types of properties are distinguished: (1) the data properties that link individuals to data values and (2) the object properties that connect individuals to other individuals. By examining existing ontologies, we observe that the property definition may be expressed in terms of other properties. Therefore, dependency relationships may be raised. This section studies dependency relationships between data properties and object properties.

1) Dependency relationships between data properties: A definition of a data property may be expressed from other primitives and/or defined data properties sharing the same domain. These derived definitions may be characterized as algebraic. An algebraic concept definition is a non-canonical definition that can be computed with the use of algebraic operators such as the composition, union, intersection, restrictions, etc. while a structural non-canonical concept definition consists in defining and deducing a notion without using algebraic operators.

Example 2: For a better understanding, let us consider the data properties *SSN* (describing social security number), *birthdate*, *age*, *Student*$_{status}$ (describing if a student is major or minor) and *country* of the class $Student$ of the ontology of Lehigh University benchmark[5] (Figure 4). This ontology is used along this paper. These properties have as a domain the $Student$ class and as a range an integer, a date or a string, respectively. Assuming that the birth_date is a primitive property, therefore, the property age is considered as a non-canonical algebraic property since it computed from the birth_date and the current date. Based on this definition, the functional dependency between the properties birth_date and age ($birthdate \rightarrow age$) can be defined.

2) Dependency relationships between object properties: An object property is a binary relation between two individuals belonging either to the same ontological class or to two different classes.

Example 3: The object property *teaches* describes a binary relation between *Professor* and *Course* classes. Data properties can be primitive or derived (computed from the primitive and/or defined object properties). Let us consider the properties $headOf$, $headOf_{Departement}$ and $GeneralHeadOf_{University}$ describing respectively the fact of being a headmaster, a department headmaster and a university general headmaster. The third property can be derived from the first two ones ($GeneralHeadOf$ = $headOf \circ headOf_{Departement}$) since a university general headmaster is the headmaster of a departement headmaster. As a consequence, a dependency relationship ($headOf$, $headOf_{Departement} \rightarrow GeneralHeadOf$) can be identified between $headOf$, $headOf_{Departement}$ and the non-canonical algebraic property $GeneralHeadOf$.

B. Dependency relationships between classes

As we said in the Section 1, an ontology may have two types of classes: the canonical (primitive) classes and the non-canonical (defined) ones. The derived classes may be classified in algebraic concepts or structural ones.

In the first case, classes are defined as class expressions based on set operators and/or property restrictions (∪, ∩, ¬, ∃, ∀, etc.). For example, let us assume that a data property $Status_{University}$ describing the status of a university (public or private) has as a domain the primitive concept $University$ describing all the universities. A $PublicUniversity$ class that specifies the public universities may be defined as the restriction on the property $University_{Status}$ on the public value (PublicUniversity $\equiv \exists$ $University_{Status}$.{public} and domain ($University_{Status}$) = University). Based on this definition, a class dependency may be identified as follows: $University \rightarrow PublicUniversity$. It means that

[5]http://swat.cse.lehigh.edu/projects/lubm/

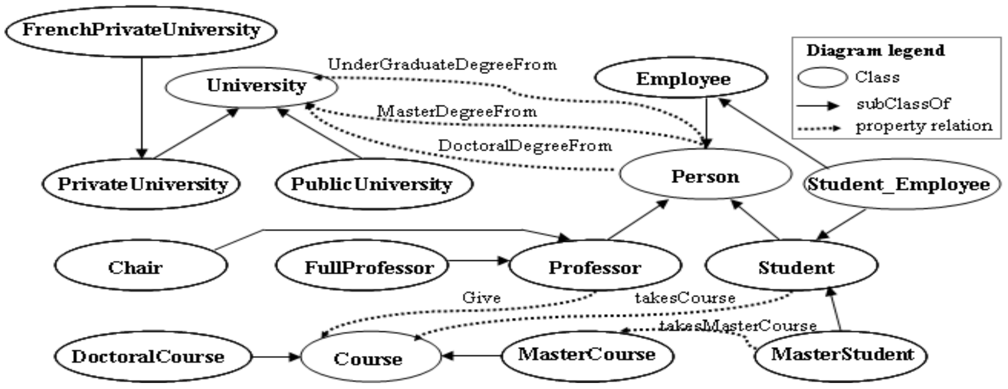


Fig. 4: A fragment of the Lehigh University ontology.

the knowledge of the whole instances of the $University$ class determines the knowledge of the whole instances of the $PublicUniversity$ concept.

In the second case, classes are expressed using a direct enumeration of its members. For a better understanding, let us consider this example. Let $CharenteUniversity$ concept be the class describing the universities located at the Charente department (in France). It is defined as one of the following University instances: Poitiers University, ENSMA, Angouleme University, ENSIP ($CharenteUniversity$ $\equiv$ $oneOf$ {PoitiersUniversity, ENSMA, AngoulemeUniversity, ENSIP}). Therefore, the $CharenteUniversity$ definition depends on the University instances definition since the knowledge of the whole instances of the enumareted universities determines the knowledge of the whole instances of the $CharenteUniversity$.

Ontology models provide the means to define derived classes. For example, the PLIB [1] model allows the definition of the derived classes across basic set operators, functions and enumeration. In OWL, various builders are given to define such classes (owl:unionOf, owl:intersectionOf, owl:hasValue, owl:oneOf, etc.).

C. Synthesis

Our study shows that ontology models allow defining concepts and properties with dependencies. But, some functional dependencies among properties cannot be captured by these definitions. Let us consider the SSN property. Knowing a student SSN allows identifying exactly other data property values. To offer designers the means to express such dependencies and to define explicitly conceptual dependencies, we propose to extend the expressive power of ontologies by incorporating these new concepts in the ontology models. In the next section, we propose a formal model of ontologies considering dependencies between ontological properties and classes.

IV. Formalization

In this section, we first propose a formal model of ontologies followed by a formalization of semantic databases.

A. A formal model of ontologies

The existing formalization of ontologies ignore the dependencies that may exist between classes and properties [26]. In this section, we propose a complete formalization of conceptual ontology. Therefore, an ontology becomes a 7-tuple $\mathcal{O}$:$\langle C, R, Ref(C), Ref(R), FD(R), FD(C), Formal\rangle$, where:

- C denotes concepts of the model (atomic concepts and concept descriptions).
- R denotes roles (relationships) of the model. Roles can be relationships relating concepts to other concepts, or relationships relating concepts to data-values (like Integer, Float, String, etc).
- $Ref(C)$: $C \rightarrow (Operator, Exp(C, R))$. Ref(C) is a function defining classes of the DL TBOX. Operators can be inclusion ($\sqsubseteq$) or equality ($\equiv$). Exp(C,R) is an expression over concepts and roles of $\mathcal{O}$ using constructors of description logics such as union, intersection, restriction, etc.(e.g, Ref(Professor)$\rightarrow$ ($\sqsubseteq$, Person $\cap$ $\forall$ givesCourse(Person, Course))).
- $Ref(R)$: $R \rightarrow (Operator, Exp(C, R))$. Ref(R) is a function defining roles of the DL TBOX. Operators can be inclusion ($\sqsubseteq$) or equality ($\equiv$). $Exp(C, R)$ is an expression over concepts and roles of $\mathcal{O}$ using constructors of description logics such as union, intersection, restriction, etc.(e.g, Ref($GeneralHeadOf$) $\rightarrow$ ($\equiv$, (headOf(Person, Person) $\circ$ headOf$_{Departement}$(Person, Department))).
- $FD(R)$: C x $2^R \rightarrow R$ a mapping from the powerset of R onto R representing dependencies defined on the applicable roles R of a class $C_i \in \mathcal{C}$. A $FD(R)$ is defined as a pair $FD(R)$:$< FD(R).RP, FD(R).LP >$ where:
  - $FD(R).RP$ is a role R describing the dependent attribute of the role functional dependency (right part) i.e. $FD(R)(C_i, (R_1, .., R_n))$;
  - $FD(R).LP$ is a powerset of R describing the determinant set of the role functional dependency (left part) i.e. domain($FD(R)\,(C_i)$);
- $FD(C)$ a mapping from the powerset of C onto

C representing class dependencies. A $FD(C)$ is defined as a pair $FD(C)$:$< FD(C).RP, FD(C).LP >$ where:
  - $FD(C).RP$ is a class C describing the dependent class of the class dependency (right part) i.e. $FD(C)((C_1, .., C_n))$;
  - $FD(C).LP$ is a power set of C describing the determinant set of the class dependency (left part) i.e. domain($FD(C)$);
- $Formal$ is the formalism followed by the ontology model like RDF, OWL, PLIB, etc.

B. Semantic database formalization

Since ontology is a part of a $\mathcal{SDB}$, a formalization of such database becomes necessary. $\mathcal{SDB}$ is then defined as a 8-tuple $\mathcal{SD}$:$\langle IM, I, Pop, SM_{IM}, SM_I, Ar, Rel, RS\rangle$.

- IM is the information model of the source. It describes a fragment of the ontology $\mathcal{O}$ that defines the user requirements. Three scenarios may occur:
  - $IM \subset \mathcal{O}$ means that $\mathcal{O}$ is rich enough to cover the user requirements;
  - $IM = \mathcal{O}$ means that $\mathcal{O}$ covers all the designer requirements;
  - $IM \supset \mathcal{O}$ means that $\mathcal{O}$ does not fulfill the whole designer requirements. The designer extracts from the $\mathcal{O}$ a fragment corresponding to the requirements and enriches it by adding new concepts/properties/dependencies to define the IM;
- I presents the instances or data of the source;
- Pop: $C \rightarrow 2^I$ is a function that relates each concept to its instances;
- SM_{IM} is the storage model of the information model (vertical, horizontal, binary);
- SM_I is the storage model of the instance part I;
- Ar is the architecture model of the source;
- Rel presents the relations (tables) for instances store;
- RS: $C \rightarrow 2^{Rel}$ is a function that relates each class to its relations;

C. Instantiation of the Semantic Database Model

To instantiate the above model, let us consider an extended fragment (F_{LU^O}) of the Lehigh University ontology benchmark (LU^O) as shown in Figure 4, where some classes and dependencies are added. For example, a dependency between the classes Student and Master Studend ($Student \rightarrow MasterStudent$) is defined, since the $MasterStudent$ class may be expressed as a student with a master value (MasterStudent $\equiv \exists$ level.Master; Domain(level) = Student). The dependency $idUniv \rightarrow name$ meaning that the university identifier ($idUniv$) determines the university name ($name$) is an example of dependencies defining on role of the University class.

We consider two examples of instantiation of our model by considering one commercial DBMS which is Oracle and an academic DBMS which is OntoDB.

a) Oracle DBMS: The different components of $\mathcal{SDB}$ are

- ($IM = F_{LU^O}$)$\supset\mathcal{O}$ since F_{LU^O} = fragment($\mathcal{O}$) $\cup$ C $\cup$ R $\cup$ FD(C) $\cup$ FD(R);
- I presents the F_{LU^O} instances. For example, the triplets ($University\#1$, type, University) ($University\#1$, idUniv, 'M50421') describe that $University\#1$, instance of University class, has 'M50421' as an identifier value;
- For each $C_i \in C_{IM}$, $Pop(C_i)$ is defined. If we consider the $University$ class, $Pop(University)$ = $\{University\#1$, $University\#2$,..., $University\#100\}$;
- Oracle uses a vertical representation to store its information system;
- A vertical table is used to store instances. Note that this table store both IM and I;
- Oracle has a $Type_1$ architecture (two parts: meta-base and data);
- Since Oracle uses vertical representation, a unique table is created;
- RS = ϕ.

b) OntoDB DBMS:

- $IM = F_{LU^O}$;
- I presents the F_{LU^O} instances.
- As in Oracle, $Pop(University)$=$\{University\#1$, $University\#2$,..., $University\#100\}$;
- OntoDB uses the hybrid storage layout;
- OntoDB uses the horizontal representation to store its instances;
- OntoDB has a $Type_3$ architecture (four parts: meta-base, ontology, meta-schema and data);
- Since OntoDB uses the horizontal representation, a table is generated for each class. Therefore, Rel = $\{Rel_{University}$, $Rel_{Student}$, $Rel_{Professor}$, Rel_{Course}, Rel_{Person}, etc.$\}$
- A relation Rel_i is generated for each $C_i \in C$. Note that the class and its associated relation has the same name in OntoDB. Thus, RS(C_i) = Rel_{C_i}. For example, RS($University$) = $Rel_{University}$ since a table University is generated and associated to the $University$ class.

V. Design of Consistent semantic database logical models

In this paper, we are interested in exploiting dependency relationships between ontological concepts to improve the $\mathcal{SDB}$ design process. Inspired from relational database design [27], we propose to incorporate dependency relationships in the $\mathcal{SDB}$ design process in order to (1) reduce redundancy by generating a normalized logical model, and (2) improve the

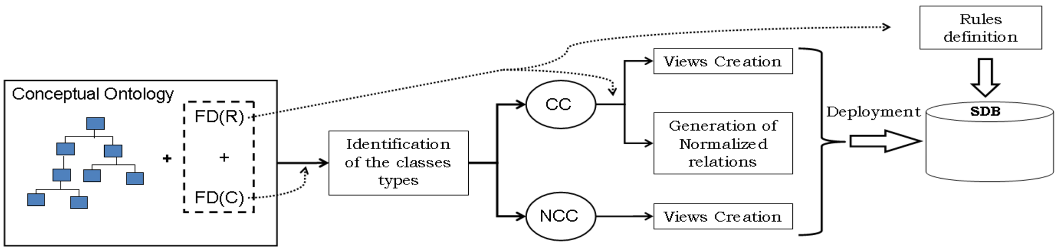


Fig. 5: Semantic database design approach

database quality by detecting the inconsistent data caused by the violation of integrity constraints identified from dependencies definitions.

Therefore, we propose our methodology to design $\mathcal{SDB}$ with good quality. Our methodology has the following steps (Figure 5):

1) **Definition of Conceptual Model:** the designer defines the conceptual model by extracting a fragment of the used ontology according to her/his requirements
2) **Identification of Canonical Concepts:** as we said in the previous sections, the exploitation of dependencies between classes facilitates the identification of canonical and non-canonical classes. To do so, we propose a graph structure. It is built from $FD(C)$, where its nodes represent the ontology classes. An arc exists between two nodes n_i and n_j if a dependency exists. This graph is then used as the input of our proposed algorithm to determine the minimum set of canonical concepts (CC) and generates the set of non-canonical concepts (NCC). This algorithm starts by computing the isolated classes (classes not involved in FD(C)). Note that these classes will be canonical since they can not be derived from other ones. Then, the minimum coverage-like classes (C+) is computed. It represents the minimum subset of basic FD(C) to generate all the others. C+ has to be treated so as to eliminate circular relationships between classes. Finally, the set of CC is identified and a consequence, the set of NCC is deduced. Recall that this identification contributes in reducing the redundancy in the final $\mathcal{SDB}$.
3) **Generation of a Normalized Logical Schema:** for each canonical class $cc_i \in$ CC, we exploit the FD(R) defined on their properties to generate the normalized logical model. Note that, for each cc_i , a primary key is computed and relations in 3NF are generated.
4) **Hiding the physical implementation:** to facilitate the user access, we define a view on the normalized relations corresponding to each canonical class. So, the users may query the $\mathcal{SDB}$ without worrying about the physical implementation of those classes.
5) **Access Support to non-canonical classes:** to ensure the transparency in accessing data through non-canonical classes and reduce redundancy in the target $\mathcal{SDB}$, we propose the use of views. For each $ncc_i \in$ NCC, a relational view is computed. For example, let ncc_j be a NCC defined as the intersection of the two canonical classes $cc_1(r1, r2, r3)$ and $cc2(r1, r2, r4)$ where $\{r1, r2, r3, r4\} \subset R$, a view corresponding to ncc_j is defined as illustrated in the Listing 1.

Listing 1: Example of a relational view definition

```
CREATE VIEW nccj AS
((SELECT r1, r2 FROM cc1) INTERSECT
(SELECT r1, r2 FROM cc2))
```

6) **Deployment:** Due to multitude choices of architectures of the target DBMS and storage layouts, the deployment process become more harder than the classical databases. To illustrate this difficulty, let us study its complexity. Let $\mathcal{A} = \{A_1, A_2, .., A_n\}$, $\mathcal{P} = \{P_1, P_2, .., P_n\}$ and $\mathcal{SM} = \{SM_1, SM_2, .., SM_n\}$ be the set of all possible architecture types, set of various needed parts according to the design approach and the set of existing storage models. The number of deployment possibilities $\mathcal{D}$ that designer needs to consider is given by the following equation:

$$\mathcal{D} = card(\mathcal{A}) \times (card(\mathcal{SM}))^{card(\mathcal{P})}. \quad (1)$$

Our analysis on databases architectures [25], [28], [9], [29], [7] gives rise to three main architectures. Each identified architecture may have at most four parts: meta-base, meta-schema, ontology and data. According to the previous steps, the designer have to deploy only in three parts: the meta-schema part to store the extended ontology model (supporting dependencies definition and classes types), the ontology part to store ontology definition and the data part to store the generated logical model and ontological data. Note that three main storage models may be used for each one of these parts as follows [11]: (SM_1) vertical table, (SM_2) horizontal representation and (SM_3) binary representation. Therefore, the deployment complexity is equal to $O(3 \times (3^3))$. Studying 81 cases becomes a difficult task.
The designer is constrained to consider a significant number of choices. This makes the deployment process more complex. To overcome this situation, the $\mathcal{SDB}$ deployment process must be done in a generic way. For that reason, we propose a deployment on demand implemented as a service which follows the standards such as WSDL[6], SOAP[7] and UDDI.
Based on the different parts of $\mathcal{SDB}$ architectures, three kinds of deployment services are identified:

[6]http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
[7]http://www.w3.org/TR/wsdl

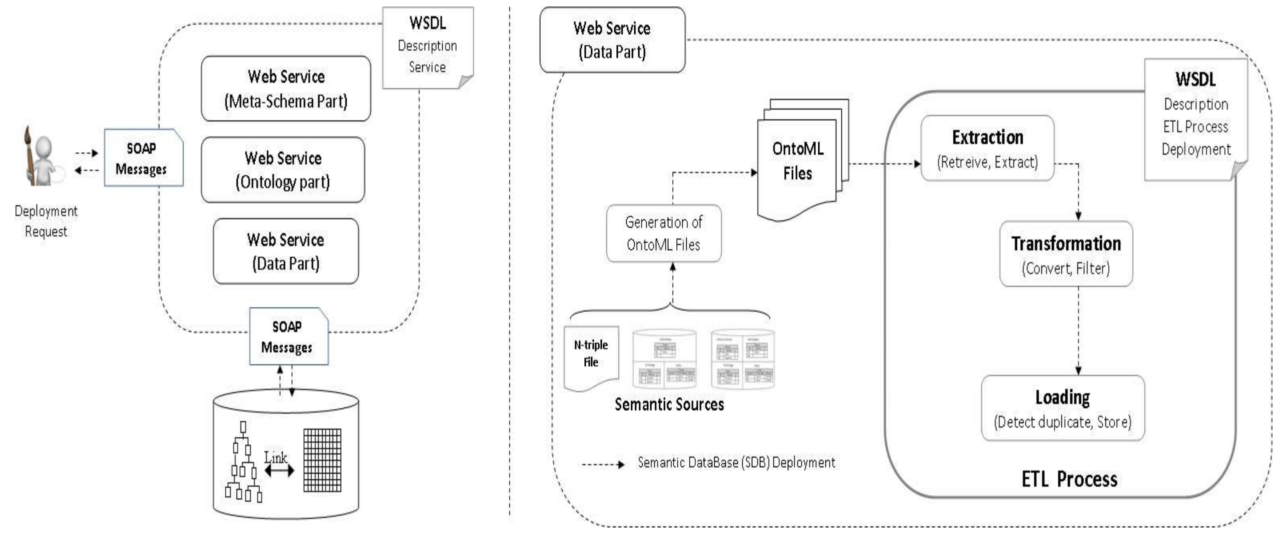


Fig. 6: Deployment of $\mathcal{SDB}$ as a Service.

(1) a service concerns the meta-schema level, (2) a service dedicated to the ontology level and (3) a service for the data level. Note that a $\mathcal{SDB}$ may materialize ontological data that come from various sources (Web, files, databases, etc.) with different formats. A similar ETL service has to be developed to clean, transform and load data into the $\mathcal{SDB}$. [17] defined ten generic operators typically encountered in an ETL process, which are:

a) *EXTRACT (S,C)*: extracts, from incoming record-sets, the appropriate portion.
b) *RETRIEVE(S,C)*: retrieves instances associated to the class C from the source S.
c) *MERGE(S,I)*: merges instances belonging to the same source.
d) *UNION (C,C')*: unifies instances whose corresponding classes C and C' belong to different sources S and S'.
e) *JOIN (C, C')*: joins instances whose corresponding classes C and C' are related by a property.
f) *STORE(S,C, I)*: loads instances I corresponding to the class C in a target data store S.
g) *DD(I)*: detects duplicate values on the incoming record-sets.
h) *FILTER(S,C,C')*: filters incoming record-sets, allowing only records with values of the element specified by C'.
i) *CONVERT(C,C')*: converts incoming record-sets from the format of C to the format of C'.
j) *AGGREGATE (F, C, C')*: aggregates incoming record-sets applying the aggregation function $\mathcal{F}$ (e.g., COUNT, SUM, AVG, MAX) defined in the target data-store.

For our case, all these operations are considered except the aggregation operator.
The data are then extracted from different sources, saved in a temporary file generated into OntoML[8] format (Ontology Mark-up Language) which allows ontologies representation and exchange. The OntoML file is used as a generic format of semantic data exchange, transformed according to the format of the target schema (vertical, horizontal, etc.) and loaded into the $\mathcal{SDB}$. Figure 6 illustrates the deployment process.

Algorithm 1 shows an implementation of our ETL operators.

A validation of our approach is proposed in the next section.

VI. A CASE STUDY

A case study implementing the proposed $\mathcal{SDB}$ design approach in Oracle 11g is proposed. The choice of this database is justified by the leading position that has Oracle 11g in the database area. To lead our validation, we use an extended fragment of the Lehigh University ontology as shown in Figure 4. The validation process contains two main steps: (1) the design step and (2) the deployment step where a services-based case tool is proposed.

A. The design step

The design stage requires the execution of a set of steps as follows: (1) the OWL meta-schema extension, (2) the primary key computation, (3) the analysis of classes and (4) the conversion of the OWL ontology to a N-Triple format.

1) The OWL Meta-schema extension: Since the used ontology language does not handle dependencies representation, we propose to extend the OWL meta-schema by adding the following meta-classes: FD(R), FD(R).RP, FD(R).LP, FD(C), FD(C).RP and FD(C).LP. These meta-classes describe respectively a functional dependency defined between properties, its right part, its left part, a class dependency, the FD(C) right part and the FD(C) left part. For each added meta-class, a

[8]http://wiki.eclass.eu/wiki/ISO_13584-32_ontoML

```
begin
    Input: $\mathcal{SDB}$ (the schema only) and Sources ($S$)
    Output: Populated $\mathcal{SDB}$ (schema + instances)
    for Each C : Class of ontology do
        $I_{SDB} = \phi$
        for Each source $S_i \in S$ do
            if $Cs \equiv C$ /* instances in $S_i$ satisfying all constraints imposed by SDB*/ then
                $C' = IdentifyClasse(S_i, C)$
            end
            else
                if $Cs \subset C$ /*Instances in Si satisfy all constraints imposed by SDB, plus additional ones */ then
                    C'= IdentifyClasse (Si, C)
                end
                else
                    if $Cs \supset C$ /* Instances satisfy only a subset of constraints imposed by SDB*/ then
                        if format(C) $\neq$ format(Cs) then
                            C'= CONVERT (C, Cs) /*identify the format conversion from the source to the target SDB*/
                        end
                        if C represents filter constraint then
                            C'= FILTER ($S_i$, C, Cs) /*identify the filter constraint defined in the target SDB*/
                        end
                    end
                end
            end
            $I_{si} = RETRIEVE(S_i, C)$ /*Retrieve instances of C*/
            if more than one instance are identified in the same source then
                $I_{SDB}$= MERGE ($I_{SDB}$, $I_{si}$) /*Merge instances of Si*/
            end
            if more than one instance are identified in different sources then
                $I_{SDB}$= UNION ($I_{SDB}$, $I_{si}$) /*Unites instances incoming from different sources*/
            end
        end
        if Source contain instances more than needed then
            $I_{SDB}$= EXTRACT ($I_{SDB}$, $I_{si}$) /* Extract appropriate instances*/
        end
    end
    STORE(SDB,C, DD($I_{SDB}$)) /*Detects duplicate; loads them in SDB*/
end
```

Algorithm 1: ETL operators

set of meta-properties is defined. For example, the *Its_Class* meta-property references the meta-class to which the FD(R) is associated. These dependencies may be exploited to compute the types and the primary keys of classes. So, to represent such a data, we propose to enrich the OWL meta-model by adding the meta-classes PrimaryKey, NonCanonic and Canonic describing respectively the primary key concept and the canonicity of classes. A set of meta-properties is associated to the PrimaryKey meta-class: the PK.class references its associated class while the PK.prop describes the set of the properties composing this key. Figure 7 shows a fragment of the UML model describing the extended OWL meta-model.

2) Compute primary keys: To compute the appropriate primary key for each ontological class, we exploit the FD(R) defined on the properties of each class. To do so, we apply a java program that we have developed, on the extended OWL ontology. Based on the FD(R), this program computes and associates a primary key for each ontological class. To lead our validation, we use an extended fragment of the Lehigh University ontology as shown in Figure 4. Let us assume that the $idUniv$ property that describes the university identifier is generated as the primary key of the $University$ class. Therefore, this step triggers the meta-schema instantiation by adding the appropriate ontological data in the University.owl file to define the generated primary key as described in the Listing 2.

Listing 2: Example of the meta-schema instantiation

```
<PrimaryKey rdf:ID="PrimaryKey_10">
<PK.prop rdf:resource="#idUniv"/>
<PK.class rdf:resource="#University"/>
</PrimaryKey>
```

Once the primary keys are computed, an analysis to identify canonical and non-canonical classes is performed as described in the next step.

3) Analysis of classes: To identify the types of classes, we apply a java program, exploiting the class dependencies, on the the Lehigh University ontology. Once the canonical and non-canonical classes are identified, we instantiate the meta-scheme classes "*Canonic*" and "*NonCanonic*" by generating the owl statements describing the classes types definition. Figure 8 illustrates the class analysis process of the used ontology.

4) Ontology conversion: Oracle 11g offers only the data loading under the N-TRIPLE format (.nt). To meet this requirement, we use the converter *rdfcat* provided by the Jena API (version 2.6.4) as shown in Figure 9. This tool enables the transformation of an OWL file (.owl) to a N-TRIPLE file (.nt). For example, the Listing 3 describes the instruction allowing the transformation of the Lehigh University ontology from the OWL format to the N-TRIPLE format.

Listing 3: Conversion of the Lehigh University ontology

```
C:\programs\Jena-2.6.4>java jena.rdfcat -out ntriple
 University.owl > University.nt
```

By this conversion, the OWL statements defining the primary key *idUniv* of the University class (see previous paragraph) are transformed into a set of triplets as described in Listing 4.

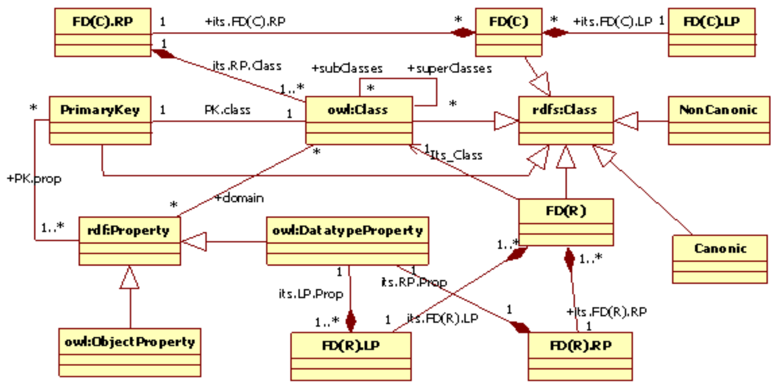


Fig. 7: OWL meta-schema extension.

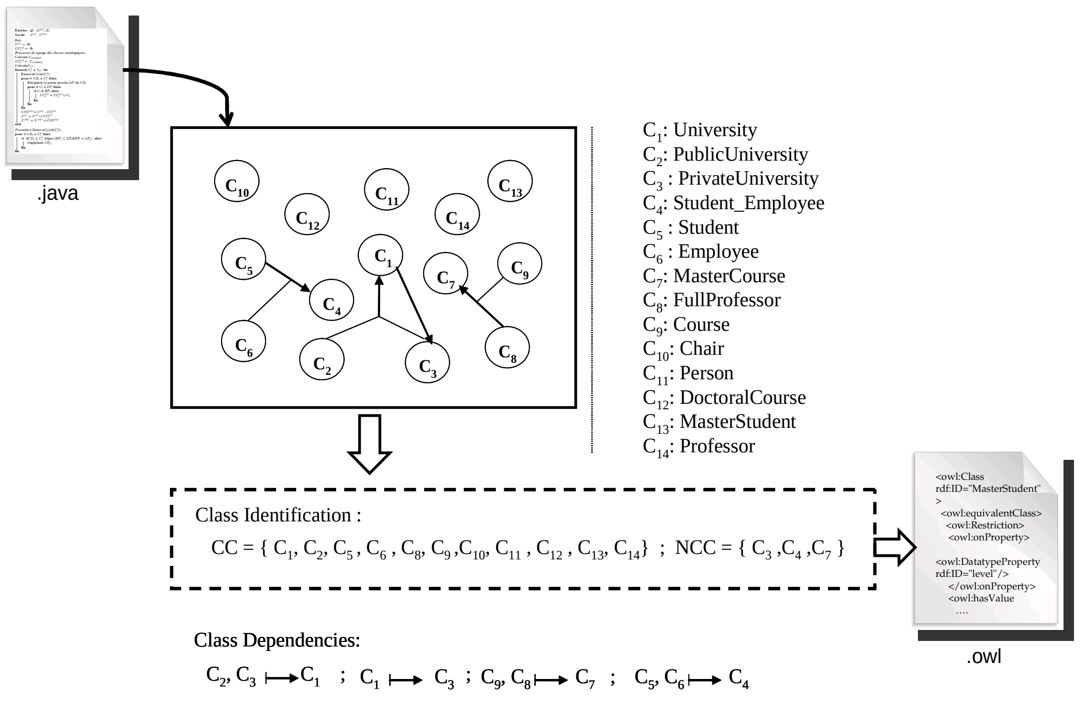


Fig. 8: Example of the classes analysis process.

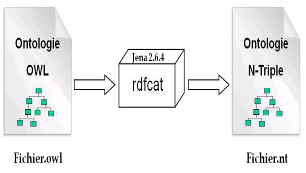


Fig. 9: Conversion of an OWL ontology to N-Triple ontology.

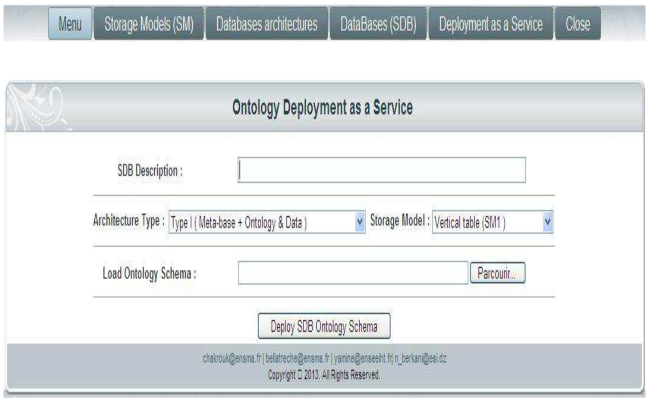
Listing 4: Example of the primary key conversion

```
(PrimaryKey_10, type, PrimaryKey)
(PrimaryKey_10, PK.prop, idUniv)
(PrimaryKey_10, PK.class, University)
```

Once the ontology conversion is done, the deployment step is applied as described in the next section.

B. OntoDep: a services-based case tool

In this section, we propose a service-based case tool enabling the deployment of our approach on $\mathcal{SDB}$. The proposed tool is implemented in Java language and uses OWL API to access ontologies. Each service mentioned in section V is implemented as a Web Service. According to the chosen architecture and storage layout, the appropriate Web Service is invoked via SOAP messages.



Ontology Deployment as a Service

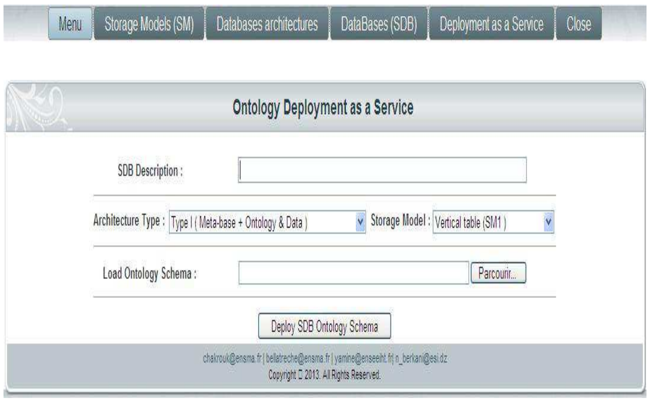
SDB Description :

Architecture Type : Type I (Meta-base + Ontology & Data) Storage Model : Vertical table (SM1)

Load Ontology Schema :

chakrouk@ensma.fr | bellatreche@ensma.fr | yamine@enseeiht.fr| n_berkani@esi.dz
Copyright 2013. All Rights Reserved.

Fig. 10: Deployment process: the meta-schema service invocation.



Ontology Deployment as a Service

SDB Description :

Architecture Type : Type I (Meta-base + Ontology & Data) Storage Model : Vertical table (SM1)

Load Ontology Schema :

chakrouk@ensma.fr | bellatreche@ensma.fr | yamine@enseeiht.fr| n_berkani@esi.dz
Copyright 2013. All Rights Reserved.

Fig. 11: Deployement process: the ontology service invocation.

The deployment process is performed by invoking the suitable Web Service that translates the semantic data to either a vertical, horizontal or binary representation, by generating XML files. The data loaded into our database are issued from sources. As a consequence ETL operators (Extract, Filter, Convert, Merge, etc.) have to be implemented using the appropriate ontological language (SPARQL for vertical representation, OntoQL for Horizontal, etc.). During this step, the implemented ETL algorithm is executed in order to extract semantic data from XML files, to transform and to load them into the target $\mathcal{SDB}$. The description of each Web Service is implemented in a WSDL file.

Our tool is implemented so that technical details are hidden to the user. Access to the persistent storage is implemented using Data Access Object (DAO) Design patterns [30]. The DAO pattern provides flexible and transparent accesses to different storage layout. Based on the architecture of the $\mathcal{SDB}$, the right object DAO is selected. In order to obtain a generic deployment process, our solution follow a service oriented architecture (SOA). SOA offers the loose coupling of the defined Web Services and interaction among them. It allows the integration of new web services without affecting the existing one. This provides the flexibility of the deployment process. To deploy our approach on Oracle 11g, three main steps are required as follows: (1) the meta-schema deployment, (2) the ontology deployment and (3) the data deployment. Note that Oracle 11g is a $Type_1$ architecture and uses the vertical table as a storage model for all types of data.

1) The meta-schema deployment: Oracle 11g offers several techniques for the loading: (i) the bulk load, (2) the batch load and (3) a load into tables using SQL INSERT statements. We chose the first method for its fast loading. It loads the ontological data in a staging table using the SQLLoader utility (sqlldr) before being sent to the database. Since our design approach requires the ontology meta-schema extension, the deployment meta-schema service is invoked. This service extends the used $\mathcal{SDB}$ meta-model by adding the needed meta-classes and meta-properties to handle dependencies and canonicity representations. The Figure 10 illustrates the invoking of the meta-schema service.

2) The Ontology deployment: Once the meta-schema is deployed, the ontology has to be loaded. To do so, we invoke the ontology deployment service. Firt, we specify the ontology deployment characteristics: the $\mathcal{SDB}$ architecture (Type$_1$) and the used storage model (vertical table). Then we load the appropriate file describing the used ontology (University.nt). This loading involves the storing of classes, properties, concepts hierarchy, conceptual dependencies, primary keys and classes types. The Figure 11 illustrates the invoking of the ontology service.

3) The Data deployment: To deploy data on the $\mathcal{SDB}$, the data deployment service is invoked. First, we specify the data deployment characteristics: the $\mathcal{SDB}$ architecture (Type$_1$) and the used storage model (vertical table). Then, the ETL process is applied. It consists in extracting instances from the N-Triple file, filtering them, converting these instances according to the $\mathcal{SDB}$ schema, merging instances related to the same classes and loading them into Oracle 11g. The Figure 11 illustrates the invoking of the ontology service. The demonstration video illustrating the deployment process of $\mathcal{SDB}$ is available at http://www.lias-lab.fr/forge/rcisvideo/video.html.

C. Synthesis

In the most semantic databases, storage layouts are frozen. For example, in Oracle 11g, the vertical representation is used for the storage of ontological concepts and instances. Therefore, our approach may be improved to consider all possible combinations of storage layouts. Indeed, based on the class dependencies, the class's types are identified and stored in the ontology-based data models. The property functional dependencies modeling allows computing primary keys for each ontological class and subsequently, storing them in the $\mathcal{SDB}$. Based on these keys, rules helping to detect a set of inconsistent data may be defined. Therefore, the data quality may be improved by detecting a set of inconsistent and duplicated data violating these integrity constraints. Thus, exploiting dependencies ensures that the stored data correspond to the boundaries of the modeled universe and reduces the inconsistency and redundancy in the semantic database models.

VII. Conclusion

This paper shows the need to develop a complete methodology for designing consistent semantic databases including the main steps of database lifecycle: conceptual, logical, physical and deployment. This is motivated by the spectacular development of academic and industrial solutions to support such databases. We tried to show the *strong coupling* between ontologies and databases and to highlight ontologies in the design process of databases. In the past ontologies were used in the conceptual level of the database development. If we accept to push them along all steps of the lifecycle, we need to give database designers algorithms and tools to identify redundancy. Considering all ontological concepts (canonical and non-canonical) may cause inconsistent databases. To avoid this situation, we proposed to enrich the traditional definition of ontologies with dependency relationships between properties and classes of the ontology. Precise formalizations of ontology and semantic databases is given including different components (classes, properties, storage models for ontology and instances, architecture of the target DBMS, dependency between classes and properties). The presence of functional dependencies between properties allows generating normalized logical model. A graph-based algorithm is proposed to identify the canonical concepts of a given conceptual ontology. These concepts are stored into the database in normalized form. The non-canonical concepts are represented by relational views. Our approach offers a deployment à la carte based on Web Services including ETL process. A case tool dedicated to the deployment process is proposed considering semantic Oracle DBMS.

Currently, we are developing a design tool in order to assist designers during the $\mathcal{SDB}$ design process. Also, we are working in proposing cost models evaluating the cost for each deployment instances.

References

[1] G. Pierra, “Context-explication in conceptual ontologies: the plib approach,” in *Proceedings of the 10th ISPE International Conference on Concurrent Engineering (ISPE CE 2003)*, 2003, pp. 243–253.

[2] S. E. P. Bijan, “Pellet: An owl dl reasoner,” in *International Workshop on Description Logics (DL2004)*, 2004, pp. 6–8.

[3] T. R. Gruber, “A translation approach to portable ontology specifications,” vol. 5, no. 2, pp. 199–220, 1993.

[4] S. Jean, G. Pierra, and Y. A. Ameur, “Domain ontologies: A database-oriented analysis,” in *WEBIST (1)*, 2006, pp. 341–351.

[5] F. T. Fonseca and J. E. Martin, “Learning the differences between ontologies and conceptual schemas through ontology-driven information systems,” *Journal of the Association for Information Systems*, vol. 8, no. 2, 2007.

[6] J. Broekstra, A. Kampman, and F. V. Harmelen, “Sesame: A generic architecture for storing and querying rdf and rdf schema,” in *International Semantic Web Conference*, 2002, pp. 54–68.

[7] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, “Jena: implementing the semantic web recommendations,” in *13th international conference on World Wide Web (WWW’2004)*, 2004, pp. 74–83.

[8] S. Das, E. I. Chong, G. Eadon, and J. Srinivasan, “Supporting ontology-based semantic matching in rdbms,” in *VLDB*, 2004, pp. 1054–1065.

[9] H. Dehainsala, G. Pierra, and L. Bellatreche, “Ontodb: An ontology-based database for data intensive applications,” in *DASFAA*, April 2007, pp. 497–508.

[10] J. Lu, L. Ma, L. Zhang, J. S. Brunner, C. Wang, Y. Pan, and Y. Yu, “Sor: A practical system for ontology storage, reasoning and search,” in *VLDB*, 2007, pp. 1402–1405.

[11] Z. Pan and J. Heflin, “Dldb: Extending relational databases to support semantic web queries,” in *The First International Workshop on Practical and Scalable Semantic Systems*, 2003.

[12] M. Stocker and M. Smith, “Owlgres: A scalable owl reasoner,” in *The Sixth International Workshop on OWL: Experiences and Directions*, 2008.

[13] D. Tapucu, G. Diallo, S. Jean, Y. At-Ameur, M. O. nalir, and N. Belaidi, “Dfinition et exploitation des prfrences au niveau smantique,” in *Journes Francophones sur les Ontologies (JFO 2009)*, 2009, pp. 29–36.

[14] N. Belaid, Y. Aït Ameur, and J. F. Rainaud, “A semantic handling of geological modeling workflows,” in *International ACM Conference on Management of Emergent Digital EcoSystems*, 2009, pp. 83–90.

[15] E. F. Codd, “A relational model of data for large shared data banks,” vol. 13, no. 6, 1970, pp. 377–387.

[16] A. Rochfeld, “Merise, an information system design and development methodology, tutorial,” in *ER*, 1986, pp. 489–528.

[17] D. Skoutas and A. Simitsis, “Ontology-based conceptual design of etl processes for both structured and semi-structured data,” *International Journal on Semantic Web and Information Systems*, vol. 3, no. 4, pp. 1–24, 2007.

[18] A. Borgida and G. E. Weddell, “Adding uniqueness constraints to description logics (preliminary report),” in *Deductive and Object-Oriented Databases, 5th International Conference (DOOD’97)*, 1997, pp. 85–102.

[19] D. Calvanese, G. D. Giacomo, and M. Lenzerini, “Identification constraints and functional dependencies in description logics,” in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI’2001)*, 2001, pp. 155–160.

[20] B. Motik, I. Horrocks, and U. Sattler, “Bridging the gap between owl and relational databases,” *Journal of Web Semantics*, vol. 7, no. 2, pp. 74–89, 2009.

[21] O. Romero, D. Calvanese, A. Abelló, and M. Rodriguez-Muro, “Discovering functional dependencies for multidimensional design,” in *DOLAP*, 2009, pp. 1–8.

[22] J. P. Calbimonte, F. Porto, and C. M. Keet, “Functional dependencies in owl abox,” in *Brazilian Symposium on Databases (SBBD)*, 2009, pp. 16–30.

[23] A. Bakhtouchi, L. Bellatreche, S. Jean, and Y. A. Ameur, “Ontologies as a solution for simultaneously integrating and reconciliating data sources,” in *Sixth IEEE International Conference on Research Challenges in Information Science (RCIS)*, 2012, pp. 1–12.

[24] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, “Path-based identification constraints in description logics,” pp. 231–241, 2008.

[25] S. Jean, H. Dehainsala, D. Nguyen Xuan, G. Pierra, L. Bellatreche, and Y. Aït-Ameur, “Ontodb: It is time to embed your domain ontology in your database,” in *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA’07)*, April 2007, pp. 1119–1120.

[26] N. Berkani, S. Khouri, and L. Bellatreche, “Generic methodology for semantic data warehouse design: From schema definition to etl,” in *Fourth International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, 2012, pp. 404–411.

[27] P. P. Chen, “The enity-relationship model: Toward a unified view of data,” in *VLDB*, 1975, p. 173.

[28] M. J. Park, J. H. Lee, C. H. Lee, J. Lin, O. Serres, and C. W. Chung, “An efficient and scalable management of ontology,” in *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA’07)*, 2007, pp. 975–980.

[29] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle, “The ICS-FORTH RDFSuite: Managing voluminous RDF description bases,” in *Proceedings of the Second International Workshop on the Semantic Web (SemWeb)*, 2001.

[30] C. Mayr, U. Zdun, and S. Dustdar, “Model-driven integration and management of data access objects in process-driven soas,” in *ServiceWave*, 2008, pp. 62–73.