2012 Fourth International Conference on Intelligent Networking and Collaborative Systems

Generic Methodology for Semantic Data Warehouse Design: From Schema Definition to ETL

Nabila Berkani
† *National High School for Computer Science (ESI)*
Algiers, Algeria
Email: (n_berkani,s_khouri)@esi.dz

Selma Khouri *†
* *LIAS-ENSMA*
Futuroscope France
Email: selma.khouri@ensma.fr

Ladjel Bellatreche
* *LIAS-ENSMA*
Futuroscope France
Email: bellatreche@ensma.fr

***Abstract*—Actually, any company needs to collaborate with others to improve their performance and productivity. Ontologies can provide a way to promote collaboration between companies. They contribute on reducing the syntax and semantic conflicts that may occur during the collaboration process. Data warehouse technology is a serious candidate for data-sharing architecture that may be employed within the collaborating companies. The spectacular adoption of domain ontologies by several communities facilitates the explosion of semantic databases sources ($\mathcal{SDB}$) that become candidate for building the semantic data warehouses ($\mathcal{SDW}$). This situation motivates us to deeply formalize the structure of semantic sources in order to propose an automatic construction of a semantic data warehouse $\mathcal{SDW}$. In this paper, we first proposed a generic framework for handling semantic sources. Secondly, the generic ETL steps are incorporated to our framework. Our proposal is validated through a case study; considering *Oracle* $\mathcal{SDB}$, where each source references a global ontology of the Lehigh University BenchMark.**

***Keywords*-Semantic Data, Semantic Databases, Data Warehouse, ETL process, Data Integration Framework, Oracle.**

I. Introduction

In collaborative environments, teams and companies need to share knowledge and data to perform their day-to-day tasks. Usually, they are confronting to the syntactic and semantic heterogeneities of the used concepts. Providing semantics to the manipulated data is becoming a crucial issue. Ontologies have been proposed to reduce these heterogeneities. They have been increasingly used in different areas like e-commerce, biomedical, semantic web, e-learning, etc. As consequence, important amounts of *semantic data* ($\mathcal{SD}$) have been produced. Ontologies are linked to these data to explicit their semantic. These data may be managed in the main like *OWLIM* [1]. Such systems become very fastidious when data grow and a large amount of $\mathcal{SD}$ need to be managed and exploited. To overcome this limitation, *database* solutions have been proposed offering an efficient storage and querying mechanisms for $\mathcal{SD}$. Databases storing $\mathcal{SD}$ are called *Semantic DataBases* ($\mathcal{SDB}$). Both *academic* and *industrial* communities (like Oracle and IBM) proposed their own solutions to deal with this type of data. Usually, these solutions extend the existing solutions of traditional DBMS in terms of *storage*, *querying*, and *scalability*. Therefore, traditional operations may be also performed on $\mathcal{SDB}$ (insertion, suppression, interrogation) [2].

Actually, managing $\mathcal{SD}$ is no longer a challenge, but analyzing them to extract knowledge for decision makers of collaborative applications becomes a crucial issue. *Data Warehouse* ($\mathcal{DW}$) technology is more suitable for performing a such analysis [3]. Face to this situation, $\mathcal{SDB}$ become then serious candidates for participating in the construction of a $\mathcal{DW}$. Traditionally, $\mathcal{DW}$ gets data from relational databases, XML data, legacy, etc. Recall that the first step of the construction process of a $\mathcal{DW}$ is to *integrate* data coming from different sources. $\mathcal{DW}$ systems are actually defined as *Data Integration Systems* ($\mathcal{DIS}$), where data sources are duplicated in the same repository after applying an *Extract-Transform-Load* (ETL) process. $\mathcal{DIS}$ are formally defined by the following triplet: <G, S, M> [4], where G represents the global schema providing a unified view of data stored in different heterogeneous sources. Each source is associated with a local schema S. Mappings (M) represent correspondences defined between local and global schemas. In the first generation of $\mathcal{DIS}$ the mapping between global and local schemas are expressed by the means of relational using views. To be dependent from the logical implementation of each schema, some research efforts propose to express the mapping based on the conceptual layer. As a consequence, a conceptual model is expressed by *Description Logics* formalism (DL), or one of its fragments [4], [5]. Ontology-based $\mathcal{DIS}$ may easily fall into this second generation of integration systems. Note that domain ontologies were first proposed to represent the global schema of the $\mathcal{DIS}$. Some works proposed then to link a local ontology to each source, and to define mappings between the global and local ontology [6]. $\mathcal{DIS}$s considering $\mathcal{SDB}$ correspond to this architecture, where each $\mathcal{SDB}$ is a source storing its local ontology.

Two integration scenarios can be defined: (i) correspondences between global and local ontologies are defined *a priori* at the design time of $\mathcal{SDB}$. In such case, the integration process is simply assimilated to an integration of mappings. Designers agree to make efforts when designing

978-0-7695-4808-1/12 $26.00 © 2012 IEEE
DOI 10.1109/iNCoS.2012.108

the sources in order to get a '*free*' ETL process when data would need to be merged. (ii) Correspondences are discovered *a posteriori* either manually or automatically, which is an issue related to the domain of schema and ontology matching/alignment. Once the mappings discovered, the integration process resembles to the first scenario. We provide in this paper a process for integrating $\mathcal{SDB}$ in a $\mathcal{DW}$ system. The first issue that we need to solve is the rich diversity of the existing $\mathcal{SDB}$. Different $\mathcal{SDB}$ emerged in a short laps of time, mainly due to: (i) the existence of different ontological *formalism* (RDF, DAML, OWL), (ii) different *storage models* to represent the ontology schema and their instances (data), (iii) the final *architecture* of the $\mathcal{SDB}$ that can use one model or different models to represent all its data. The management of this diversity passes through two main steps: (a) the development of a generic integration process (<G, S, M>) based on description logics formalism and (b) the proposition of an generic ETL algorithm that dealing with heterogeneous $\mathcal{SDB}$ sources. To show the feasibility of our proposal, we instantiate this framework by considering sources managed by Semantic technology of Oracle DBMS through a case study using *LUBM* ontology benchmark.

The paper is organized as follows. Section 2 presents studies proposing to design $\mathcal{DW}$ for analyzing $\mathcal{SD}$. Section 3 presents the background introducing $\mathcal{DIS}$, Ontologies and $\mathcal{SDB}$. Section 4 presents our contribution where we first propose a generic integration framework, that we instantiate using Oracle $\mathcal{SDB}$. Section 5 concludes the paper.

II. Related Work

We focus on studies proposing to analyze semantic data using $\mathcal{DW}$ systems. Some works used semantics of data to facilitate the extraction of data and the design of the $\mathcal{DW}$, without formally defining the ETL process. For instance, [7] describes an extension of an ontology language which facilitates the loading of data from sources into a $\mathcal{DW}$ scheme according to the temporal characteristics of these sources.

Other works proposed formal methods for defining $\mathcal{DW}$s using semantic technologies.[8] uses an ontology formalized using a class-based model to generate mediators for loading data into a $\mathcal{DW}$ system. [4] proposed a method for integrating data sources into a $\mathcal{DW}$ using an ontology as a global schema playing an intermediary model between the target $\mathcal{DW}$ model and the schemas sources. [9] proposed to automate the ETL process by *constructing* an OWL ontology linking schemas of relational sources to a target $\mathcal{DW}$ schema. These works consider only *relational* sources. Some works considered *unstructured* sources like [10] who defined an ETL and a design process for analyzing source data stores. This process considers a set of relational and XML data stores integrated into an OWL ontology. [3] considered semantic data provided by the semantic web and annotated by OWL ontologies, from which a DW model is defined and populated. We notice that none of these systems consider *Semantic Databases* that become serious candidates for $\mathcal{DW}$ projects.

III. Background

We present in this sections the background related to the formalization of $\mathcal{DIS}$, Ontologies and Semantic databases.

A. Formalization of Data Integration Systems $\mathcal{DIS}$

$\mathcal{DW}$s systems are defined as Data Integration Systems ($\mathcal{DIS}$) where data sources are duplicated in the same repository. The first $\mathcal{DIS}$ <G,S,M> used *logical* representation of local and global schema, usually the relational model where mappings were defined using operators of the relational algebra. Simple types of mappings can be defined to produce homogeneous sources. It is for example the case of distributed databases where a global schema G is partitioned into horizontal, vertical or mixed fragments. Other systems needed to define more complex mappings using complex queries (views). This restriction to relational schemas is no longer valid in real integration scenarios, where more flexibility is required. Other studies proposed to provide $\mathcal{DIS}$ defined at the conceptual level independently of any logical implementation issue. Some $\mathcal{DIS}$ defined just the global schema by its conceptual model like [11]. Other studies defined fully conceptual $\mathcal{DIS}$ where the global and local schemas are defined by their conceptual models [12]. This conceptual model can be represented by an ontology. Ontologies are actually defined as conceptual models describing a domain and providing reasoning capabilities. Most of these studies used Description logics (DL) formalism (one of its fragments) to define this conceptual layer, because it is able to capture the most popular data *class-based modeling* formalisms presently used in *databases* and *Information system* analysis [13]. Ontology Web Language (OWL), the language standardized by W3C to define ontologies, is also based on DL formalism.

B. Description Logics

We can briefly define DLs as the formalism used to define logics specifically designed to represent structured knowledge and to reason upon. In DL, structured knowledge is described using *concepts* and *roles*. Concepts (like concepts *Student* and *Publication* in the University ontology of figure 1) denote sets of individuals, and roles (like *author* and *takeCourse* roles) denote binary relationships between individuals. Two types of concepts and roles are used: *atomic* and concept *descriptions*, which are defined using other concepts by applying suitable DL constructors. There are many families (fragments) of Description Logic. The most basic family is *AL* (Attributive Language), whose concept descriptions are formed using constructors: C,D $\rightarrow$ A| (atomic concept) $\top$| (universal concept) $\bot$| (bottom

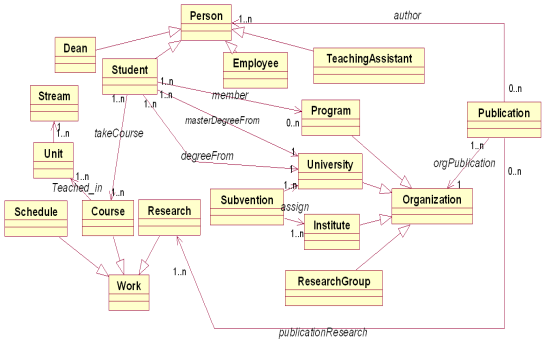

Figure 1. LUBM global schema

concept) $\neg A|$ (atomic negation) $C \sqcap D|$ (intersection) $\forall R.C|$ (value restriction) $\exists R.\top$ for limited existential quantification (A: atomic concept, R and S: atomic roles, C and D: concept descriptions). A knowledge base in DL is composed of two components: the *TBOX* (Terminological Box) stating the *intensional* knowledge and the *ABOX* (Assertion Box) stating the *extensional* knowledge or the instances (Eg. Student(*Student#1*) denotes that *Student#1* is an instance of concept *Student*). Terminological axioms have the form of inclusions: C $\sqsubseteq$ D (R $\sqsubseteq$ S) or equalities: C $\equiv$ D (R $\equiv$ S) (Example. Student $\sqsubseteq$ Person in Fig. 1).

C. Ontologies and Semantic databases

$\mathcal{SDB}$ were proposed to give more semantic to their data. Both industrial and academic communities proposed $\mathcal{SDB}$ solutions like: Rdfsuite, Ontodb [2], IBM SOR [14] and Oracle [15]. These $\mathcal{SDB}$ differ according to: (i) the used *ontological formalisms* to define the ontology like RDF, RDFS, OWL, PLIB, FLIGHT, etc. (ii) The *storage schema*of the ontology and of the data model. We distinguish three main *relational* representations: *vertical*, *binary* and *horizontal*. Vertical representation stores data in a unique table of three columns (subject, predicate, object) [15]. In a binary representation, classes and properties are stored in different tables [14]. Horizontal representation translates each class as a table having a column for each property of the class [2]. (iii) The *architecture* dedicated to store the $\mathcal{SDB}$. Systems of *type I* architecture (like Oracle $\mathcal{SDB}$ [15]) use the same architecture of traditional databases with two parts: *data schema part* and the *meta schema part* to store $\mathcal{SDB}$. In systems of *type II* architecture l(ike IBM Sor [14]), the ontology model is separated from its data which gives an architecture with three parts: the ontology model (the TBOX), the data model (the ABOX) and the meta-schema. The ontology model and the data model can be stored in different storage schemas. Systems of *type III* architecture (like Ontodb [2]) consider an architecture with four parts, where a new part representing the *the meta-schema* of the ontology is added.

IV. Generic Framework for Constructing $\mathcal{SDW}$

Given the diversity of $\mathcal{SDB}$, we had to provide a generic integration framework that represents all these different semantic databases. We present in the following the proposed framework based on DL formalism and the ETL algorithm used to integrate $\mathcal{SDB}$ into a $\mathcal{DW}$ system. Our generic framework is composed of a global schema G representing the intentional knowledge or the TBOX (the global ontology), a set of local sources S and mappings M between G and S. The extensional knowledge or instances are stored in local sources.

The global schema $< \underline{G}, S, M >$*:* The global schema is defined by its conceptual structure that we call *Information Model* (IM). IM is defined as follows IM: <C, R, Ref, formalism>

- 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 Integers, Floats, etc).
- Ref : $C \rightarrow$ (Operator, Exp(C,R)). Ref is a *function* defining terminological axioms of a DL TBOX. Operators can be inclusion ($\sqsubseteq$) or equality ($\equiv$). Exp(C,R) is an expression over concepts and roles of IM using constructors of description logics such as union, intersection, restriction, etc. (e.g., Ref(Student)$\rightarrow$($\sqsubseteq$, Person $\sqcap$ $\forall$takesCourse(Person, Course))).
- Formalism is the *formalism* followed by the global ontology model like RDF, OWL, etc.

The local source <G,$\underline{S}$*,M>:* S_i presents each local and is defined as follows: S_i: <IM, I, Pop, SMIM, SMI, Ar>.

- IM: <C, R, Ref, formalism> is the *information model* of the source.
- 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, binary or horizontal).
- SM_I: is the *Storage Model* of the instances part I.
- Ar: is the *architecture* of the source (*type I*, *type II* or *type III* architecture).

The mappings $\mathcal{DIS}$*: <G, S,* $\underline{M}$*>:* The mappings are defined between global and local schemas as follows M:< *MapSchemaG, MapSchemaS, MapElmG, MapElmS, Interpretation, SemanticRelation, Strength, Type*>. This formalization is based on [12] meta-model defined for conceptual mappings.

- *MapSchemaG* and *MapSchemaS*: present respectively the *mappable schema* of the global schema and of the local schema (the information model).
- *MapElmG* and *MapElmS*: present respectively the *mappable element* of the global schema and of the local

source. This element can be a simple concept or an expression over the schema.
- *Interpretation*: presents the *Intentional* interpretation or *Extensional* interpretation of the mapping.
- *SemanticRelation*: presents the type of semantic relationship between MapElmG and MapElmS. Three relationships are possible: *Equivalence*, *Containment* (Sound, Complete) or *Overlap*. Equivalence states that the connected elements represent the same aspect of the real world. Containment states that the element in one schema represents a more specific aspect of the world than the element in the other schema. Overlap states that some objects described by the element in the one schema may also be described by the connected element in the other schema [12].

V. ETL Algorithm

The goal of the ETL process is to populate the target DW schema, represented by an Integrating Ontology (IO). IO schema is defined from the global ontology schema (G) using users' requirements. Three scenarios are possible: (1) $IO = G$: Global schema corresponds exactly to user's requirements; (2) $IO \subseteq G$: IO extracted from global schema using modularity methods; (3) $IO \supseteq G$: Global schema does not fulfil the whole users' requirements. The designer extracts the fragment of the G corresponding to requirements and enriches it with new concepts and properties. The ETL process is applied in order to populate IO with available instances extracted from sources. [16] has defined ten generic conceptual operators typically encountered in an ETL process, which are:

(1) **EXTRACT(S,C)**: extracts, from incoming record-sets, the appropriate portion; (2) **RETRIEVE(S,C)**: retrieves instances associated to the class C from source S; (3) **MERGE(S,I)**: merges instances belonging to the same source; (4) **UNION (C,C')**: unifies instances whose corresponding classes C and C' belong to different sources S and S' respectively; (5) **JOIN (C, C')**: joins instances whose corresponding classes C and C' are related by a property; (6) **STORE(S,C, I)**: loads instances I corresponding to the class C in target data store S, (7) **DD(I)**: detects duplicate values on the incoming record-sets; (8) **FILTER(S,C,C')**: filters incoming record-sets, allowing only records with values of the element specified by C'; (9) **CONVERT(C,C')**: converts incoming record-sets from the format of the element C to the format of the element C'; (10) **AGGREGATE (F, C, C')**: aggregates incoming record-set applying the aggregation function F (COUNT, SUM, AVG, MAX) defined in the target data-store.

Based on these generic conceptual operators, we proposed Algorithm 1 for populating the IO. As explained in the introduction, the integration process is based on the semantics of mappings between global (G) and local source schemas (S_i). According to the integration framework proposed, four mappings semantics are identified: (1) $C_G \equiv C_{S_i}$: *Equivalent mappings* between global classes and source classes (no transformation is needed), instances are extracted from sources, merged, united or joined then loaded in the target data store; (2) $C_G \supset C_{S_i}$: *Containment (Sound) mappings* between G classes and source classes : source instances satisfy all constraints required by the G and more (no transformation needed), instances are extracted from sources, merged, united or joined then loaded to the target data store; (3) $C_G \subset C_{S_i}$: *Containment (Complete) mappings* between G classes and source classes: source instances satisfy only a subset of the constraints required by G classes, some instances need to be transformed (converted, filtered and aggregated) then merged, unified or joined after that they are loaded to the target data store; (4) *Overlap mappings* between G Classes and source classes: in this case we are interested to identify the constraints required by G classes and not applied to the source classes then it can be considered same as the Containment (Complete) mappings. Algorithm 1 depicts these four scenarios.

VI. Case study

In this section, a case study using Lehigh University BenchMark (LUBM) [17] is conducted, based on an industrial $\mathcal{SDB}$: ***Oracle***. It illustrates the instantiation of the generic integration framework described above and the application of the ETL algorithm proposed.

Oracle has incorporated support for languages RDF and OWL in its system to enable its customers to benefit from a management platform for semantic data. Oracle has defined two subclasses of DLs: *OWLSIF* and a richer fragment *OWLPrime*. We use *OWLPrime* fragment which offers the following constructors[1]: *rdfs:domain, rdfs:range, rdfs:subClassOf, rdfs:subPropertyOf, owl:equivalentClass, owl:equivalentProperty, owl:sameAs, owl:inverseOf, owl: TransitiveProperty, owl:SymmetricProperty, owl:FunctionalProperty, owl: InverseFunctionalProperty*. Note that *OWLPrime* limits the expressive power of DL formalism in order to ensure decidable query answering.

The scenario adopted consists on the creation of 4 Oracle $\mathcal{SDB}$s considered as sources (S_1, S_2, S_3 and S_4) and populated locally using Lehigh University Benchmark (LUBM) ontology, as illustrated in Fig. 2. We consider a scenario where a director organism (for example the education ministry) imposes the same vocabulary for all universities. Each university refers to the same global schema (LUBM schema), extracts its local schema from this global schema using simple and complex mappings, and populates this schema locally (using LUBM instances). Each source stores its schema and instances in an Oracle $\mathcal{SDB}$. Assume that the education ministry needs to perform some analysis studies to take relevant decisions. These $\mathcal{SDB}$s need thus to be

[1] http://www.w3.org/2007/OWL/wiki/OracleOwlPrime

```
begin
    Input:
        ODW: Ontology of 𝒟𝒲 (Schema) and Si: Local Source (𝒮𝒟ℬ)
    Output: The integrating ontology ODW (schema + instances)
    for Each C : Class of ontology ODW do
        I_ODW = ϕ
        for Each source Si do
            if Cs ≡ C  /* instances in Si satisfy all constraints imposed by ODW*/ then
                C'= IdentifyClasse (Si, C) /*identify class from Si*/
            end
            else
                if Cs ⊂ C  /*Instances in Si satisfy all constraints imposed by ODW, plus additional ones*/
                then
                    C'= IdentifyClasse (Si, C) /*identify class from Si*/
                end
                else
                    if Cs⊃ C Or Overlap mappings /* Instances satisfy only a subset of constraints imposed by
                    ODW*/ then
                        if format(C) ≠ format(Cs) then
                            Cconv= CONVERT (C, Cs) /*identify the constraint of format conversion from the
                            source to the target ODW*/
                        end
                        if C represent aggregation constraint then
                            Caggr= AGGREGATE (F, C, Cs) /*identify the constraint of aggregation defined by F*/
                        end
                        if C represents filter constraint then
                            Cfilt= FILTER (Si, C, Cs) /*identify the filter constraint defined in the target ODW*/
                        end
                        C'= ClasseTransformed (Si, C, Cconv, Caggr, Cfilt) /* Associate to the class C' the constraint
                        of conversion, aggregation or filtering defined by Cconv, Caggr and Cfilt*/
                    end
                end
            end
            I_si= RETRIEVE (Si, C') /*Retreive instances of C' and applying contraints of conversion,
            aggregation or filtering if necessary*/
            if more than one instance are identified in the same source then
                I_ODW= MERGE (I_ODW, I_si) /*Merge instances of Si*/
            end
            if classes have the same super class then
                I_ODW= UNION (I_ODW, I_si) /*Unites instances incoming from different sources*/
            end
            else
                if classes are related by same property then
                    I_ODW= JOIN (I_ODW, I_si) /* Join incoming instances*/
                end
            end
            if Source contain instances more than needed then
                I_ODW= EXTRACT (I_ODW, I_si) /* Extract appropriate portion of instances*/
            end
        end
        STORE( ODW,C, DD(I_ODW)) /*Detects duplicate values of instances and load them in ODW*/
    end
end
```

Algorithm 1: Algorithm for populating target datastore using conceptual ETL operators

integrated in a $\mathcal{DW}$ system following an ETL process. First, we start our case study by presenting the process of creation of the $\mathcal{SDB}$ in Oracle. We then instantiate the integration framework presented above using these sources. Finally, we apply the integration algorithm to integrate these sources in a $\mathcal{DW}$ schema.

A. Creating $\mathcal{SDB}$ from LUBM Ontology

Two types of sources participating in the construction of the target data warehouse are considered: (i) sources are defined by a simple mapping from the global ontology and (ii) sources are defined by a complex mapping from the global ontology (using DL constructors).

1) Simple mappings: In this mapping, we consider three scenarios: *vertical*, *horizontal* and *mixed* fragmentation.

Definition 1: **A vertical fragment over an ontology** is defined as the projection on a set of classes of the global ontology. Each class inherits all her roles. Fig. 3 depicts a vertical ontological fragmentation. Formally, following the integration framework, we define the global ontology by its information model: *Ontology*= $\{C$*: Set of Classes,* R*: roles,* Ref*, Formalism*$\}$, and we define the vertical fragmentation as follows: $\Pi_{(C_i,C_j,\cdots,C_m)}(C)$. The ontological projection is quite similar to the relational projection.

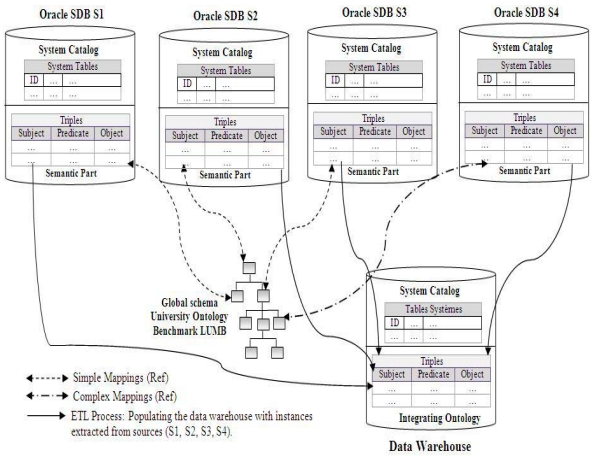

Figure 2. Integrating Oracle $\mathcal{SDB}$ using LUBM Benchmark.

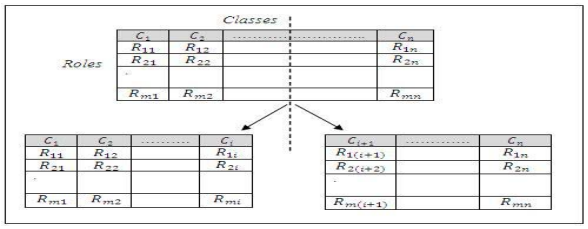

Figure 3. Vertical ontological fragmentation based on projection of classes.

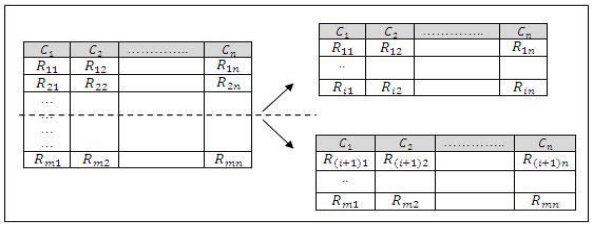

Figure 4. Horizontal ontological fragmentation based on selection of roles.

Definition 2: **A horizontal fragmentation over ontology** is defined as a restriction of a set of roles for each class of the ontology. More formally, it is defined as $\sigma_{(rest^{r_i},\cdots,rest^{r_m})}(C_j)$, where $rest^{r_i}$ and C_j represent respectivly, a restriction on the role r_i and the ontology class C_j. Fig. 4 depicts a horizontal ontological fragmentation. The ontological restriction saves the global schema of the ontology.

Definition 3: **A mixed fragmentation over ontology** is defined as the process of applying simultaneously horizontal and vertical fragmentation on the ontology.

2) Complex mappings: Note that a a concept of a local source may be defined from atomic concepts and roles of the global ontology. This definition is performed by the means of DL constructors. The most used constructors in DLs are constructors of AL fragment, enriched by: *negation* of arbitrary concepts ($\neg C$), *union of concepts* ($C \cup D$) and *number restriction constructor* ($\geq .nR$). Formally, a local ontology with complex mappings may be defined as follows: $Ontology_{CM}$= $\{C'$= $Ref(C)$, R', Ref', Formalism$\}$. Note that *Ref* denotes external references between the local and global ontology. For example: Student$_{LO}$ (LO for local ontology) is defined using concepts and roles of the global ontology as follows: Student$_{LO}$ = Person $\cap$ $\forall$takesCourse (Person, Course). Ref' denotes internal references between classes and roles of the local ontology. Note that the consistency of the resulting local ontology must *be checked by the designer*.

Based on the simple and complex mappings; four Oracle $\mathcal{SDB}$s sources are created: S_1, S_2, S_3 and S_4. The first three sources were created using simple mappings and represent respectively: *vertical*, *horizontal* and *mixed fragments over LUBM ontology*.

1) S_1: contains three classes: *person (Age, EmailAdress, Telephone, Title), Student and GraduateStudent.*
2) S_2: (i.e. it contains all its classes). A projection on the class person is done on two properties: *Age* and *EmailAdress*. The other classes are the same classes of LUBM classes.
3) S_3: is a mixed fragment over LUBM ontology containing three classes: *Person, Student* and *GraduateStudent*. A projection is done on two properties of the class *Person: Age and EmailAdress*.
4) S_4: is defined as a fragment of LUBM ontology using complex mappings. It contains three classes: *Person*, *Student* and *Employee* defined as follows:
   - $S_4.C_1$: Person, Ref(Person) = (Student $\cup$ Employee) $\cap$ $\forall$ member (Person, Organization)
   - $S_4.C_2$: Student, Ref (Student) = Student $\cap$ $\forall$takesCourse (Person, Course)
   - $S_4.C_3$: Employee, Ref (Employee) = Person $\cap$ $\forall$WorksFor (Person, Organization)

B. Instantiation of the generic integration framework

The generic integration framework <G, S, M> can be instantiated for our case study as follows:

The global schema G: The global schema is represented by LUBM ontology[2], and is formalized by its Information Model as follows:

IM$_{Oracle}$: <Classes C, Properties P (Datatype Property and Object Property), Ref, OWLPrime>. Ref is a function that gives the expression (or definitions) of classes and properties using operators available in *OWLPrime* (rdfs:subClassOf, owl:equivalentClass, rdfs:subPropertyOf, owl:equivalentProperty). *Expression* is an expression over classes and properties using *OWLPrime* constructors described above.

The local sources S***:*** Each local source S_i is instantiated for Oracle $\mathcal{SDB}$ as follow:

S_i: < IM_{Oracle}, Individuals (triples), Pop is given in tables RDF_link$ and RDF_values$, Vertical, Vertical, type I>. Vertical storage is a relational schema composed of one table of triples (subject, predicate, object). For example: *(Student, type, Class)* for the ontology storage and *(Student#1, type, Student)* and *(Student#1, takeCourse, Course#1)* for the instances storage.

The mappings: Mapping assertions between global and local schema are instantiated for Oracle as follows:

Mapping M:< IM_{Oracle} of each source, IM_{Oracle} of the global schema, Expression over G, Class of a source S, Intensional interpretation, (*Equivalent, Containment or Overlap* (owl:SubClassOf and owl:equivalentClass in OWLPrime))>.

C. ETL Process

Note that generic ETL operators defined in the previous section are expressed on the conceptual level. Therefore, each operation has to translated according the logical level of the target DBMS (Oracle). Oracle offers two ways for querying semantic data: SQL and SPARQL. We choose SPARQL to express this translation as follows:

- The namespace of University Ontology of benchmark LUBM:
  `PREFIX univ-bench: http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#`
- EXTRACT: *Select ?Instance# Where {?Instance# rdf:type nameSpace:Class. ?Instance NameSpace:DataProperty value_condition}*
  Example 1: Extract students those age = 15 years.
  Select ?student Where {?student rdf:type univ-bench:Student . ?student univ-bench:age 15}
- RETRIEVE: *Select ?Instances# Where {?Instances# rdf:type Namespace:Class}*
  Example 2: Retrieve the instances of the *Student* class.
  Select ?InstanceStudent Where {?InstanceStudent rdf:type univ-bench:Student}
- MERGE: *Select ?instance Where {{?instance rdf:type namespace:Class1} Union {?instance rdf:type namespace:Class2}}*
  Example 3: Merge instances of classes *Employee* and *Student* belonging to the same source:
  Select ? instance Where {{? instance rdf:type univ-bench:Student} Union {? instance rdf:type univ-bench:Employee}}
- UNION: *Select ?instance Where {{?instance rdf:type namespace1:Class1} Union {?instance rdf:type namespace2:Class2} }*
  Example 4: Unify instances of classes *Student* and *Person* belonging respectively to *univ-bench1* and univ-bench2 ontologies. Note that namespace enables to distinguish between different sources. *Select ? instance Where {{? instance rdf:type univ-bench1:Student} Union {? instance rdf:type univ-bench2:Person}}*
- JOIN: *Select ?instance1 ?instance2 Where {? instance1 rdf:type namespace:Class1 . ? instance2 rdf:type namespace:Class2 . ? instance1 namespace:P ? instance2}*
  Example 5: Join the classes *Student* and *Course* that are related by the object property *takesCourse*:
  Select ?instanceStudent ? instanceCourse Where {? instanceStudent rdf:type univ-bench:Student . ? instanceCourse rdf:type univ-bench:Course . ? instanceStudent univ-bench:takesCourse ? instanceCourse}
- DD: detects duplicate values on the incoming record-sets. *Select Distinct ?instance Where {{?instance rdf:type namespace:Class1} Union {?instance rdf:type namespace:Class2} }}*
  Example 6: Detect and remove duplicate values on the incoming instances associated to the classes Student and Person :
  Select Distinct ?instance Where ? instance rdf:type univ-bench:Student Union ? instance rdf:type univ-bench:Person
- FILTER: filters incoming record-sets, allowing only records with specific values of a data property P.
  Select ?instance ?P where {?Instance rdf:type namespace:Class ; namespace:P ?P . FILTER (?P > value_condition)}
  Example 7: Filter incoming student instances allowing only those with age is greater than 16 years:
  Select ?instanceStudent ?age where {? instanceStudent rdf:type univ-bench:Student ; univ-bench:age ?age . FILTER (?age > 16) }
- AGGREGATE: Aggregates incoming record-set applying aggregation function (F= COUNT, SUM, AVG, MAX).
  Select (Count(?Instance) AS ?count) Where {?Instance rdf:type namespace:Class} Group By ?Instance

[2]http://www.lehigh.edu/ zhp2/2004/0401/univ-bench.owl

Example 8: Select number of student :
Select (count(?Student) AS ?count) Where {?Student rdf:type univ-bench:Student } Group By ?Student

- STORE: loads instances corresponding to a class in the target data store. The following statement shows a SPARQL query selecting all triples from a source then inserts them into a staging table of oracle SDB using a SQL query:
Select ?subject ?prop ?object Where { ?subject ?prop ?object } Insert Into Staging_table Values (id, SDO_RDF_TRIPLE_S (subject, prop, object));

D. Results of integration

The results of our integration process is a data warehouse whose schema corresponds to the classes and properties of the integrated ontology IO, populated by instances selected from Oracle $\mathcal{SDB}$s. We considered the case where the global schema fits exactly with the users' requirements ($IO = G$). We can store this IO (model and instances) in an Oracle $\mathcal{SDB}$ to simulate a semantic $\mathcal{DW}$. A multidimensional organization of the obtained $\mathcal{DW}$ needs to be performed. It can be easily performed using our previous work [18]. We proposed a generic integration approach (framework and ETL algorithm) defined at the conceptual level that subsumes existing $\mathcal{SDB}$. The only effort that has to be made is to instantiate the framework, and to translate the ETL operators according to the query language of the target DBMS. The rest of the integration process is fully automatic.

VII. Conclusion

In this paper, we have presented a generic approach for analyzing semantic data provided by $\mathcal{SDB}$s using a DW system. The approach is based on a generic conceptual integration framework that handles $\mathcal{SDB}$s diversity and an ETL algorithm based on generic conceptual operators for populating the target DW schema. Different kinds of mappings between source schemas and a global ontology schema are defined, allowing the creation of data sources: (i) Simple mappings by selecting vertical, horizontal or mixed fragments of the global ontology; and (ii) complex mappings by selecting views of the global ontology. A case study is conducted using LUBM benchmark schema as a global ontology, and using its instances to populate four Oracle $\mathcal{SDB}$s. The application of the ETL algorithm generates the target DW schema populated with instances loaded from $\mathcal{SDB}$s.

Currently, we are developing a tool supporting our generic methodology for designing semantic data warehouses.

References

[1] A. Kiryakov, D. Ognyanov, and D. Manov, "Owlim - a pragmatic semantic repository for owl." *WISE Workshops*, pp. 182–192, 2005.

[2] C. Fankam, "Ontodb2 : un systme flexible et efficient de base de donnes base ontologique pour le web smantique et les donnes techniques," Poitiers University, Ph.D. Thesis, 2009.

[3] V. Nebot and R. Berlanga, "Building data warehouses with semantic web data," *Decision Support Systems*, vol. 52, no. 4, pp. 853–868, 2012.

[4] D. Calvanese, G. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, "Data integration in data warehousing," *Int. J. Cooperative Inf. Syst.*, vol. 10, no. 3, pp. 237–271, 2001.

[5] P. Haase and B. Motik, "A mapping system for the integration of owl-dl ontologies," in *IHIS*, 2005, pp. 9–16.

[6] H. Wache and et al., "Ontology-based integration of information - a survey of existing approaches," in *OIS*, 2001, pp. 108–117.

[7] A. Salguero, F. Araque, and C. Delgado, "Spatio-temporal ontology based model for data warehousing," *WSEAS Int. Conf. on Telecommunications and Informatics (TELE-INFO)*, pp. 125–130, 2008.

[8] T. Critchlow, M. Ganesh, and M. R., "Automatic generation of warehouse mediators using an ontology engine," in *KRDB*, 1998, pp. 1–8.

[9] D. Skoutas and A. Simitsis, "Designing etl processes using semantic web technologies." 2006, pp. 67–74.

[10] O. Romero, A. Simitsis, and A. Abelló, "Gem: Requirement-driven generation of etl and multidimensional conceptual designs," in *DaWaK*, 2011, pp. 80–95.

[11] D. Calvanese, G. Giacomo, D. Lembo, R. Lenzerini, Rosati, and M. Ruzzi, "Using owl in data integration," in *Semantic Web Information Management*, 2009, pp. 397–424.

[12] S. Brockmans, P. Haase, L. Serafini, and H. Stuckenschmidt, "Formal and conceptual comparison of ontology mapping languages," in *Modular Ontologies*, 2009, pp. 267–291.

[13] D. Calvanese, M. Lenzerini, and D. Nardi, "Description logics for conceptual data modeling," in *Logics for Databases and Information Systems*, 1998, pp. 229–263.

[14] 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*, pp. 1402–1405, 2007.

[15] Z. Wu, G. Eadon, S. Das, E. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan, "Implementing an inference engine for rdfs/owl constructs and user-defined rules in oracle," in *ICDE*, 2008, pp. 1239–1248.

[16] D. Skoutas and A. Simitsis, "Ontology-based conceptual design of etl processes for both structured and semi-structured data," *Int. J. Semantic Web Inf. Syst.*, vol. 3, no. 4, pp. 1–24, 2007.

[17] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Journal of Web Semantics*, pp. 158–182, 2005.

[18] S. Khouri and L. Bellatreche, "A methodology and tool for conceptual designing a data warehouse from ontology-based sources," *DOLAP10*, pp. 19–24, 2010.