Cluster Comput (2013) 16:915–931
DOI 10.1007/s10586-013-0266-7

Towards a conceptualization of ETL and physical storage of semantic data warehouses as a service

Nabila Berkani · Ladjel Bellatreche · Selma Khouri

Received: 30 December 2012 / Accepted: 9 April 2013 / Published online: 26 April 2013

Abstract The data warehouse technology has become the incontestable tool for businesses and organizations to make strategic decisions to ensure their competitively. The construction of a data warehouse ($\mathcal{DW}$) passes by selecting relevant information sources, extracting relevant data and loading them into the $\mathcal{DW}$. These processes require a precise expertise from designers related to logical and physical implementations of information sources, which is not usually an easy task. The diversity and heterogeneity of information sources makes the construction process of the $\mathcal{DW}$ complex and time consuming. Domain ontologies have been proposed to reduce heterogeneity between sources, platforms, services, etc. They resolve syntax and semantic conflicts. The phenomenon of adopting domain ontologies by organizations creates a new type of databases, called semantic databases ($\mathcal{SDB}$). As a consequence, they become a candidate for building the semantic $\mathcal{DW}$ ($\mathcal{SDW}$). To handle the diversity of information sources and hide the implementations aspects of sources, proposing a generic framework for constructing $\mathcal{DW}$ becomes a necessity. In this paper, we first proposed an ontology-based approach for designing $\mathcal{SDB}$. Secondly, ETL phases are defined at ontological level to hide the implementation details. Thirdly, a storage service for ontologies and its associated data is given. Finally, our proposal is validated through a case study and a tool.

Keywords Semantic databases · Data warehouse · Ontologies · ETL processes · Service · BPMN

N. Berkani · S. Khouri
National High School for Computer Science (ESI), Algiers, Algeria

N. Berkani
e-mail: n_berkani@esi.dz

S. Khouri
e-mail: s_khouri@esi.dz

L. Bellatreche (✉) · S. Khouri
LIAS/ISAE-ENSMA, Futuroscope 86960, France
e-mail: bellatreche@ensma.fr

S. Khouri
e-mail: selma.khouri@ensma.fr

1 Introduction

The data warehouse technology ($\mathcal{DWT}$) has become the incontestable tool for businesses and organizations to make strategic decisions. It is considered as a pillar of the integration industry widely developed in last two decades [1]. A data warehouse ($\mathcal{DW}$) is defined as a *stepwise* information flow from information sources through *materialized views* towards analyst clients [2]. One of the difficulties of building a $\mathcal{DW}$ application is handling the heterogeneity of information sources. A $\mathcal{DW}$ system may be viewed as a *Data Integration System* ($\mathcal{DIS}$), where parts of data sources are duplicated in the same repository after applying the ETL (extraction, transformation, loading) process. ETL phase aims at designing the mappings and the data transformations required to load data sources into the logical schema of the $\mathcal{DW}$. This loading should take into account the model of the logical schema (Relational OLAP,[1] Multidimensional OLAP, Hybrid OLAP). ETL represents the most important stage of the $\mathcal{DW}$ design. Note that 70 % of the risk and effort in the $\mathcal{DW}$ project is attributed to this stage. The vast amount of data makes ETL extremely time-consuming [3].

[1]On-Line Analytical Processing.

A $\mathcal{DIS}$ is formally defined by the following triplet: $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ [4], where $\mathcal{G}$ represents the global schema providing a unified view of data stored in different heterogeneous sources. Each source $S_i \in \mathcal{S}$ is associated with a local schema. Mappings ($\mathcal{M}$) represent correspondences defined between local and global schemes. In the first generation of $\mathcal{DIS}$, the mapping are performed at the logical/physical level by the means of *relational views*. This is due to two main factors: (i) the large availability of relational databases in organization that aliment the $\mathcal{DW}$ and (ii) the deep knowledge of designers of the relational model. Note that, defining mapping at logical/physical levels forces developers and designers to go inside operational sources which is not an easy task, especially when the number of sources is large which is usually the case of advanced $\mathcal{DW}$ applications. In the meantime, the nature of sources has been evolved, where new types were launched: XML, semantic sources, etc. As a consequence, the construction process of $\mathcal{DW}$ may also evolve and should be generic. In the other hand, the development of the cloud computing pushes and encourages designers and developers to hide implementations and infrastructures details [5, 6]. To be independent from the logical and the physical implementations of each schema, in the past, some research efforts propose to express the mapping on the conceptual level. To do so, the conceptual model of sources and the $\mathcal{DW}$ is expressed by formal languages such as *Description Logics* formalism (DL) or one of its fragments [4, 7].

Parallel to these research efforts, in last two decades, a new era of developing and using *ontologies* arises. An ontology is an explicit specification of a conceptualization [8]. Several ontologies have been developed in various domains like e-commerce, engineering, medicine, semantic web, etc. They have been largely used in data integration systems to reduce heterogeneities between sources by offering mechanisms to resolve conflicts between entities that may be encountered at the schema level and at the data level [9]. Some studies argue that ontologies leverage conceptual models by making them more consensual and enriching them by reasoning mechanisms. The intensive use of ontologies generates a new type of data, called semantic data. In order to facilitate the management of big semantic data, a new database solution has been proposed by academician and industrials, called *semantic databases* ($\mathcal{SDB}$). Both industrial and academic communities proposed $\mathcal{SDB}$ solutions like: Rdfsuite, Ontodb [10], IBM SOR [11] and Oracle [12]. 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) [12]. In a binary representation, classes and properties are stored in different tables [11]. Horizontal representation translates each class as a table having a column for each property of the class [10]. (iii) The *architecture* dedicated to store the $\mathcal{SDB}$. In systems like Oracle [12], use the same architecture of traditional databases with two parts: *data schema part* and the *meta schema part* to store $\mathcal{SDB}$. In systems like IBM Sor [11], the ontology model is separated from its data which gives an architecture with three parts: the ontology model, the data schema and the meta-schema. Systems like Ontodb [10] consider an architecture with four parts, where a new part representing the *meta-schema* is added.

Note that domain ontologies have been widely used in $\mathcal{DIS}$, where they represent the global schema of the $\mathcal{DIS}$ [9, 13]. Some other works proposed then to attach an ontology (usually called local ontology) to each source, and to define mappings between the global and local ontologies [14, 15]. $\mathcal{DIS}$ considering $\mathcal{SDB}$ correspond to this architecture, where each $\mathcal{SDB}$ is a source storing a 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 the sources in order to get a '*free*' ETL process. (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.

The diversity of information sources, ontology formalisms (RDF, DAML, OWL, PLIB), storage layouts dedicated to ontologies and their associated data, the architectures of the target DBMS, requires the development of generic solutions to construct $\mathcal{DW}$. In this paper, we propose generic solutions for a part of the chain of the $\mathcal{DW}$ process that includes ETL by exploiting the presence of ontologies information sources and the physical storage *as a service* of ontologies and its associated instances during the physical phase. It takes into account the diversity of storage models (vertical, horizontal, and hybrid).

The paper is organized as follows. Section 2 presents a rich state of art related to ETL processes and semantic $\mathcal{DW}$. Section 3 presents the background introducing the description logic formalisms to describe domain ontologies. Section 4 describes our generic framework for constructing a semantic warehouse from semantic sources. Section 5 gives a model for ETL processes based on Business Process Model and Notation (BPMN) and a definition of the ten generic operators of the ETL processes (already proposed in the litterature) at the ontological level. Section 6 presents a case study of our proposal to show its feasibility. Section 7 discusses solutions based on the service technology dedicated to ETL and the storage deployment of the target $\mathcal{DW}$. Section 8 concludes the paper.

2 Related work

In this section, we analyze the most important works done on the ETL. We have realized that these studies cover the three phases of the ETL process: *conceptual, logical* and *physical*.

At the conceptual level, some research efforts have been concentrated on proposing formalism languages for ETL process modeling. We can cite ad-hoc formalisms [16], standard languages using UML [17], model driven architecture (MDA) [18], BPMN [19, 20] and mapping modeling [21–23]. The model proposed in [16] represents ETL processes as a graph $G(N, E)$; where nodes N represent transformations, constraints, attributes as first-class modeling elements, and data stores. An edge between two node n_i and n_j $(i \neq j)$ represents data flows, inter-attribute relations, compositions, or concurrent candidates. Trujillo et al. [17] formalize ETL processes by the means of UML class diagrams, where UML notes can be attached to each ETL operation indicating its functionality. In [19], a methodology for designing ETL processes is proposed. It is centered on the notion of quality objectives. It models the operational processes of the enterprise as well as the processes for generating the end-to-end business views required for operational decision-making. In [20], a BPMN[2]-based conceptual modeling of ETL processes is described. It is based on a classification of ETL objects resulting from a study analyzing the most used commercial and open source ETL tools. In [21] the authors proposed a UML data mapping diagram which considers relations as classes and attributes as proxy classes. Attributes are connected to the relation classes via stereotyped "Contain" relationships. They can be related to each other via stereotyped "Map" relationships. These conceptual approaches focus on the graphical design of the ETL process, whereas the identification of the required mappings and transformations are usually done manually. To make automatic these mapping, Calvanese et al. [21, 22] focus on the construction of a $\mathcal{DW}$, where its global schema is expressed through an enriched Entity-Relationship (ER) model formalized in DLR language.[3] Logical schemes (relations) of sources and the target schema of the $\mathcal{DW}$ are defined as views on this conceptual model. Three types of reconciliation correspondences are defined (namely conversion, matching, and merging correspondences) in order to load data sources into the $\mathcal{DW}$.

[2]Business Process Modeling Notation.

[3]DLR is a subset of Description Logics (DL) formalism.

The logical modeling of ETL processes gets less attention from the $\mathcal{DW}$ community. In [24], ETL workflow is modeled as a graph. The nodes of the graph are activities, record-sets, attributes, and the edges are the relationship between nodes defining ETL transformations. In [25] a formal ETL logical model is given using L_{DL} [26] as a formal language for expressing the operational semantics of ETL activities.

Regarding the studies done at the physical design of ETL, a couple of works have been proposed [27–30]. In [27], the authors present an UML-based physical model of ETL process. This work makes efforts on modeling the logical structure related to the data storage model and the hardware and software configurations supporting ETL. In [28], a set of algorithms was proposed to optimize the physical ETL design. Alkis et al. [29] propose algorithms for optimizing the performance of ETL (efficiency). Other non functional properties (freshness, recoverability, and reliability) of ETL have been proposed in [30]. From the industrial point of view, a plethora of commercial ETL tools exist in the market and the physical design aspects are an entire part of them, since they take into account implementation and execution aspects of ETL. The most prominent tools are Microsoft Integration Services [31], Oracle Warehouse Builder [32], IBM Datastage [33] and Informatica PowerCenter [34]. These tools are based on different paradigms and with different expressivity power.

Another relevant criterion related to ETL works is its automation. Recall that ETL process is costly for organizations. Ontologies may contribute on automating ETL process by the use of their reasoning capabilities and conceptualization. However, ontologies were timidly used in the ETL and researchers do not study their contributions to facilitate the ETL process. Skoutas et al. [35] automate the ETL process by constructing an OWL ontology linking schemes of semi-structured and structured (relational) sources to a target $\mathcal{DW}$ schema. The work in [36] is based on [35]'s proposal. The authors define an ETL process for analyzing data source and define ETL operations to be executed according to a pre-defined multidimensional model. The logical and physical design steps are ignored. [37] consider data provided by the semantic web and annotated by OWL ontologies, from which a $\mathcal{DW}$ model is defined and populated. However, the ETL process in this work is dependent on the storage model used for instances which is the triples. [37] and [36] are two attempts that propose ontological methods combining multidimensional modeling and ETL design.

A comparison including Nebot et al. [37] and Romero et al. [36] works and our proposal is given in Table 1 based on five criteria: (1) the nature of information sources (NIS), (2) the language formalism used to express mapping between source schemes and the warehouse schema (LFM), (3) the level of ETL definition (conceptual, logical and physical) (LED), (4) the degree of automation of ETL process

(AETL) and (5) the chosen storage layout during the deployment (SL).

3 Background

We present in this section the background related to the languages used to describe ontologies, and semantic databases.

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 [38]. Ontology Web Language (OWL), the language standardized by W3C to define ontologies, is also based on DL formalism.

DLare defined 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 Fig. 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 $\mathcal{AL}$ (Attributive Language), whose concept descriptions are formed using constructors: $C, D \rightarrow A|$ (atomic concept) $\top|$ (universal concept) $\bot|$ (bottom 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: $C \equiv D$ ($R \equiv S$) (Eg. Student $\sqsubseteq$ Person in Fig. 1).

Table 1 Comparison of ontology-based approaches of the $\mathcal{DW}$ construction

Criteria	Work		
	Nebot et al. [37]	Romero and al. [36]	Our proposal
NIS	Webdata sources	Webdata sources	Semanc databases
LFM	Description logic	No specified	description logic
LED	C/L/P	C	C/L/P
AETL	semi automatic	semi automatic	automatic
SL	Triple	none	à la carte

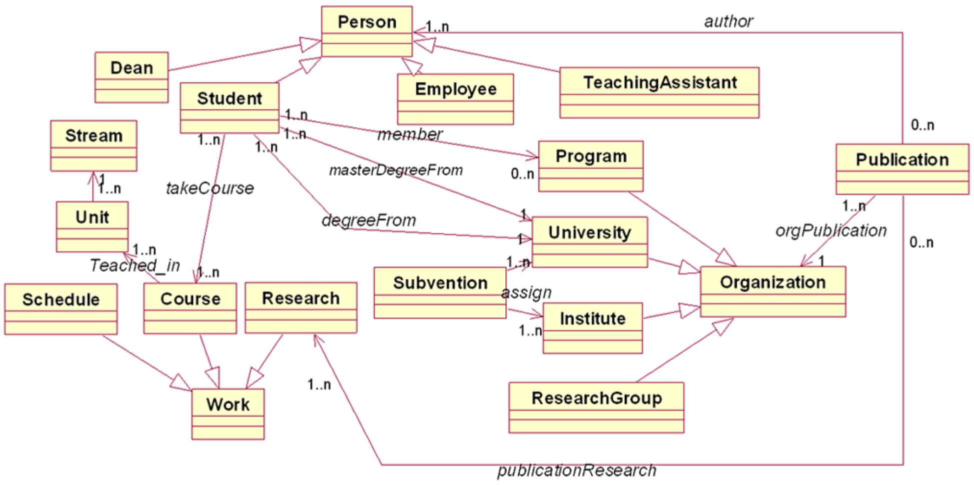
4 Generic framework for constructing $\mathcal{SDW}$

In this section we propose a generic framework based on the DL formalism for a $\mathcal{DW}$ system integrating $\mathcal{SDB}$. Initially, our generic framework is composed of a global schema $\mathcal{G}$ representing the intentional knowledge or the TBOX (the global ontology), a set of local sources S and mappings $\mathcal{M}$ between $\mathcal{G}$ and $\mathcal{S}$. The extensional knowledge or instances are stored in local sources.

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

– 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 re-

Fig. 1 LUBM global schema



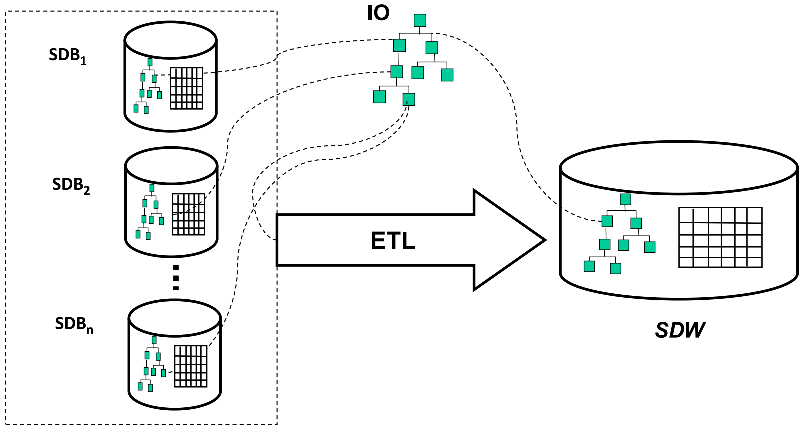


Fig. 2 Transitive closure of the $\mathcal{SDW}$ definition

lationships relating concepts to data-values (like Integers, Floats, etc).

- *Ref* : $C \to$ (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., $Ref(Student) \to (\sqsubseteq, Person \sqcap \forall takesCourse(Person, Course))$).
- Formalism is the *formalism* followed by the global ontology model like RDF, OWL, etc.

This definition allows representing any domain ontology.

The information sources $\langle G, \underline{S}, M\rangle$ Each local source $S_i \in \mathcal{S}$ representing a semantic database is defined as follows: $S_i : \langle IM_i, I_i, Pop_i, SM_{IM_i}, SM_{I_i}, Ar_i\rangle$.

- $IM_i : \langle C_i, R_i, Ref_i, formalism_i\rangle$ is the *information model* of the source.
- I_i: presents the *instances* or data of the source.
- $Pop_i : C_i \to 2^{I_i}$ is a *function* that relates each concept to its instances.
- SM_{IM_i}: is the *Storage Model* of the information model (vertical, binary or horizontal).
- SM_{I_i}: is the *Storage Model* of the instances part I_i.
- Ar_i: is the *architecture* model of the source.

The mappings $\mathcal{DIS}$: $\langle G, S, \underline{M}\rangle$ The mappings are defined between global and local schemes as follows $M : \langle MapSchemaG, MapSchemaS, MapElmG, MapElmS, Interpretation, SemanticRelation\rangle$. This formalization is based on [39] 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 [39].

The $\mathcal{SDW}$ is defined as a transitive closure as follows (Fig. 2): $\mathcal{SDW}$: $\langle IM_{DW}, I_{DW}, Pop_{DW}, SM_{IM_{DW}}, SM_{I_{DW}}, Ar_{DW}\rangle$.

5 ETL process

The goal of the ETL process is to populate the target $\mathcal{DW}$ 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 fulfill 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. [35] 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.

5.1 BPMN for ETL process

The conceptual design of the ETL process still remains a complex and error-prone activity, due to its complexity and the lack of standards. Existing conceptual approaches and their tools have focused on providing graphical solutions, based on different formalisms (ad hoc, UML, MDA, BPMN, etc.). Consequently, there is a need to have a generic and a standardized method. On the other hand, ETL process can be considered as a particular type of business process applied in $\mathcal{DW}$ environment. Several authors [19, 20] argue that the ETL process can be seen as a workflow process. BPMN (Business Process Model and Notation) has emerged as a standard notation for designing business processes. Consequently, BPMN can be customized for designing ETL processes. BPMN 2.0 is a process modeling language whose semantics are supported by a standard proposed by the OMG.[4] BPMN is not owned by vendors. In addition, the OMG does not provide any official methodology for BPMN design. Consequently, BPMN is highly adopted by the Business Process Management community. There are several improvements between BPMN 1.2 and BPMN 2.0 specifications. The most significant one is the BPMN meta-model and the mechanism allowing his extension, remaining BPMN-compliant. BPMN 2.0 meta-model is available through two syntaxes: graphical notation and textual using XML Schema Definition (XSD) proposed by W3C. BPMN 2.0 meta-model consists of more than one hundred elements giving rise to many combinations of concepts and semantics. In this paper, we consider a fragment of BPMN as core elements that we consider sufficient to describe ETL processes. Figure 3 depicts a relevant fragment of the BPMN meta-model.

[4]http://www.omg.org/cgi-bin/doc?dtc/10-06-02.

The list of meta-classes and their descriptions are presented as follows:

- *BaseElement*: the abstract super class for all *BPMN meta-classes*.
- *Flow objects*: inherits the attributes and model associations of *BaseElement*. It consists of *activities*, *events*, *gateways*, *input* and *output* specifications.
- *Activity*: is a process step that can be atomic (*Tasks*) or decomposable (*Sub-Processes*).
- *Task*: is an atomic activity within a process flow. A task is used when the work in the process cannot be broken down to a finer level of detail.
- *Event*: is something that happens during the course of a process. It affects the flow of the process and usually has a cause or an impact.
- *DataInput*: represents the declaration of entries of a given process.
- *DataOutput*: represents the declaration of outputs of a given process.
- *InputSet*: is a collection of DataInput elements.
- *OutputSet*: is a collection of DataOutput elements.
- *Gateway*: used to control how the flow interacts (converge or diverge) within a process.
- *Inclusive*: used to create alternative and parallel choices within a process flow.

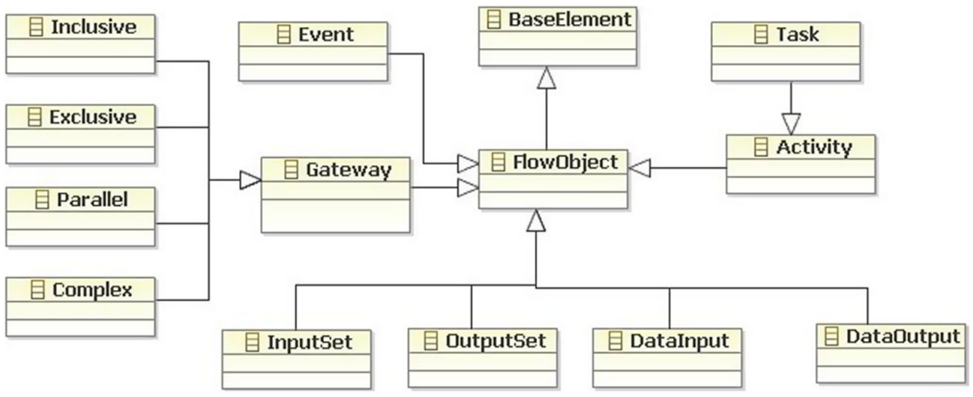


Fig. 3 BPMN for ETL Process

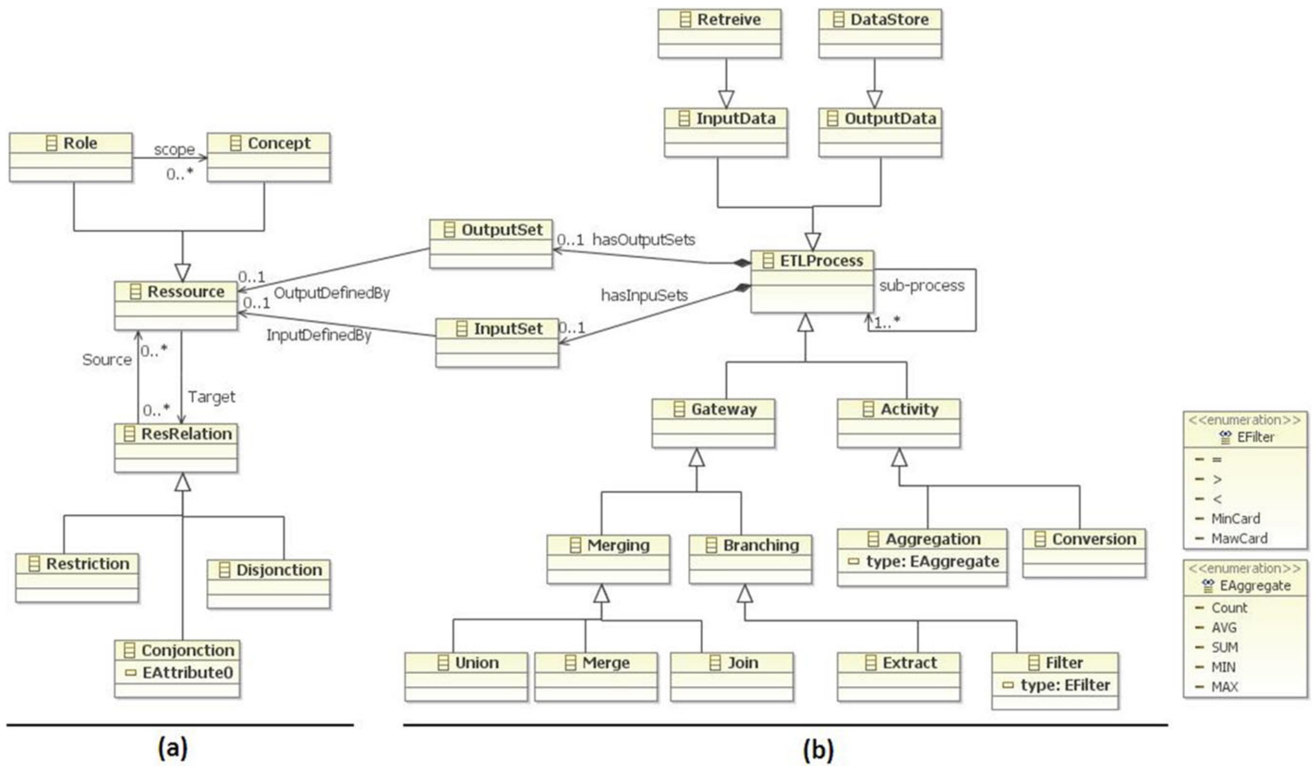


Fig. 4 Connection of ETL Model to Ontology Model

– *Exclusive*: used to create alternative choices within a process.
– *Parallel*: used to synchronize (combine) parallel flows and to create parallel flows.
– *Complex*: used to model complex synchronization behavior.

Note that the BPMN meta-model focuses only on the structure of concepts and relationships, without taking into account either their representations (textual or graphical) or their semantics. The meta-model can be instantiated by assigning the required semantic for each concept and relationship. It can be extended with new non-standard elements to satisfy specific needs without contradicting the semantics of any BPMN element used. In addition, constraints defining the role and semantic of concepts are not explicitly included in a set of rules in BPMN 2.0 specification. They are not expressed by a formal language.

To make the corresponding between ETL and business processes, we propose to extend BPMN in such way that ETL operations will be described at the ontological level. We enrich BPMN with the generic conceptual ETL operators typically encountered in an ETL process (Extract, Retrieve, Merge, Union, Join, Filter, Conversion, Aggregation and Store). The model is presented in Fig. 4.

The instantiated ETL model contains the following classes:

– *ETL Process Class*: is the central class which provides the mechanisms to manage the instances extracted from the source until their loading into the $\mathcal{DW}$. The ETL process class is composed of several sub ETL process, defining different ETL operations, that corresponds to each requirement. It means that ETL processes sharing the same requirement are represented as a SubETLProcess.
– *InputSet Class*: represent the entry parameter of the ETL process which is the source schema (semantic databases defined by mean of concepts and roles) and the target $\mathcal{DW}$ schema (the ontology is defined by the means of concepts and roles).
– *OutputSet*: represent the output of the ETL process. It can be either the intermediate output (sub process) or the final output (ETL process). The final output corresponds to the $\mathcal{DW}$ schema, defined by mean of concepts and roles.
– *Data Input*: represents the retrieved instances of sources.
– *Data Output*: represents the loaded data into the target $\mathcal{DW}$.
– *Gateway*: controls the flow of both diverging (Branching class) and converging (Merging) data Flow.
– *Branching Class*: delivers multiple output-sets which can be further classified in Filter operations (filters instances based on conditions) or Extract operations (the appropriate portion of instances is selected).
– *Merging Class*: combines multiple instances into a single one. We identify three possible operations: (i) Merge op-

eration applied when the instances belong to classes of the same source; (ii) Union operation applied when instances belong to classes that have the same super class; and (iii) Join operations applied when instances belong to classes that are related by same property (object property).
– *Activity:* represents points in the process where work is performed. It corresponds to all operations of conversion and aggregation.
– *Aggregation*: aggregates incoming instances applying the aggregation function F (count, sum, avg, max, min).
– *Conversion*: converts incoming instances from the format specified in the corresponding source to the format of the target $\mathcal{DW}$.

Recall that a given domain ontology is described by concepts and properties, both are used to define the schemes of data sources and the $\mathcal{DW}$. Thereby, a connection between the instantiated ETL model and the ontology model is feasible. Figure 4 illustrates this connection. A connection between core elements describing the input/output flow of an ETL process (InputSet, OutputSet) and the resources (concepts and roles) of the ontology is established. This connection allows designers to choose the most relevant ontological concepts to derive ETL transformations at ontological level, without wondering about the implementation details.

5.2 ETL on ontological level

Based on these generic conceptual operators, we proposed an algorithm for populating the integrated ontology *IO*. According to our framework, four semantic mappings are identified:

1. $C_G \equiv C_{S_i}$: *Equivalent mappings* between global classes and the classes of source S_i (no transformation is needed), instances are extracted from sources, merged, united or joined then loaded in the target warehouse;
2. $C_G \supset C_{S_i}$: *Containment (Sound) mappings* between $\mathcal{G}$ classes and the classes of source S_i: 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 the classes of $\mathcal{G}$ and the source S_i: source instances satisfy only a subset of the constraints required by $\mathcal{G}$'s classes, some instances need to be transformed (converted, filtered and aggregated) then merged, unified or joined after that they are loaded to the target warehouse;
4. *Overlap mappings* between $\mathcal{G}$ Classes and the classes of source S_i: in this case we are interested to identify the constraints required by $\mathcal{G}$ 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.

The consequence to connect the ontology model to the ETL model allows us to augmenting/extending the initial definition of the $\mathcal{DW}$ defined as: $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$. It becomes as: $\langle \mathcal{G}, \mathcal{S}, \mathcal{M}, ETL \rangle$

6 Case study

In this section, a case study using *Lehigh University BenchMark* (LUBM) [40] 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:[5] *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. 5. 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 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.

6.1 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

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

```
begin
    Input: (1) $IO$: The integrating Ontology $IO$ (the schema only), (2) $S_i$: Local source ($\mathcal{SDB}$)
    Output: The integrating ontology $IO$ (schema + instances)
    for Each C : Class of ontology IO do
        $I_{IO} = \phi$
        for Each source $S_i$ do
            if C exists in $S_i$ /*Equivalent mappings, instances in $S_i$ satisfy all constraints imposed by IO*/ then
                $C'$ = IdentifyClass ($S_i$, $C$) /*identify class from source*/
            end
            else if $Cs \subset C$ /*Instances in $S_i$ satisfy all constraints imposed by IO, plus additional ones*/ then
                $C'$ = IdentifyClass ($S_i$, $C$) /*identify class from source*/
            end
            else if $Cs \supset C$ or Overlap mappings /*Instances satisfy only a subset of constraints imposed by IO*/
            then
                if format of C is different from format of $C'$ then
                    $C'$ = CONVERT($C$, $C'$) /*Convert instances from format of sources to that of $IO$*/
                end
                if C represent aggregation constraint defined by $F$ then
                    $C'$ = AGGREGATE($F$, $C$, $C'$) /*Aggregates instances using function $F$*/
                    else if $C$ represents filter constraint then
                        $C'$ = FILTER($S_i$, $C$, $C'$) /*Filter instances depending on $C$ of $IO$*/
                    end
                end
                $C'$ = ClassFiltred($S_i$, $C$) /*Select the class filtered*/
            end
        end
        $I_{S_i}$ = RETRIEVE($S_i$, $C'$) /*Retrieve instances associated to $C'$ in $S_i$*/
        if more than one instance are identified in the same source then
            $I_{IO}$ = MERGE($I_{IO}$, $I_{S_i}$) /*Merge instances associated to $C$*/
        end
        if classes have the same super class then
            $I_{IO}$ = UNION($I_{IO}$, $I_{S_i}$) /*Unites instances incoming from other sources*/ else if classes are related
            by same property then
                $I_{IO}$ = JOIN($I_{IO}$, $I_{S_i}$) /*Join incoming instances*/
            end
        end
        if Source contain instances more than needed then
            $I_{IO}$ = EXTRACT($I_{IO}$, $I_{S_i}$) /*Extract appropriate portion of instances*/
        end
    end
    STORE($IO$, $C$, $DD(I_{IO})$) /*Detects duplicate values of instances and store them in $IO$*/
end
```

Algorithm 1: Algorithm for populating the $\mathcal{DW}$ using conceptual ETL operators

(ii) sources are defined by a complex mapping from the global ontology (using DL constructors).

6.1.1 *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. Figure 6 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 fragmen-

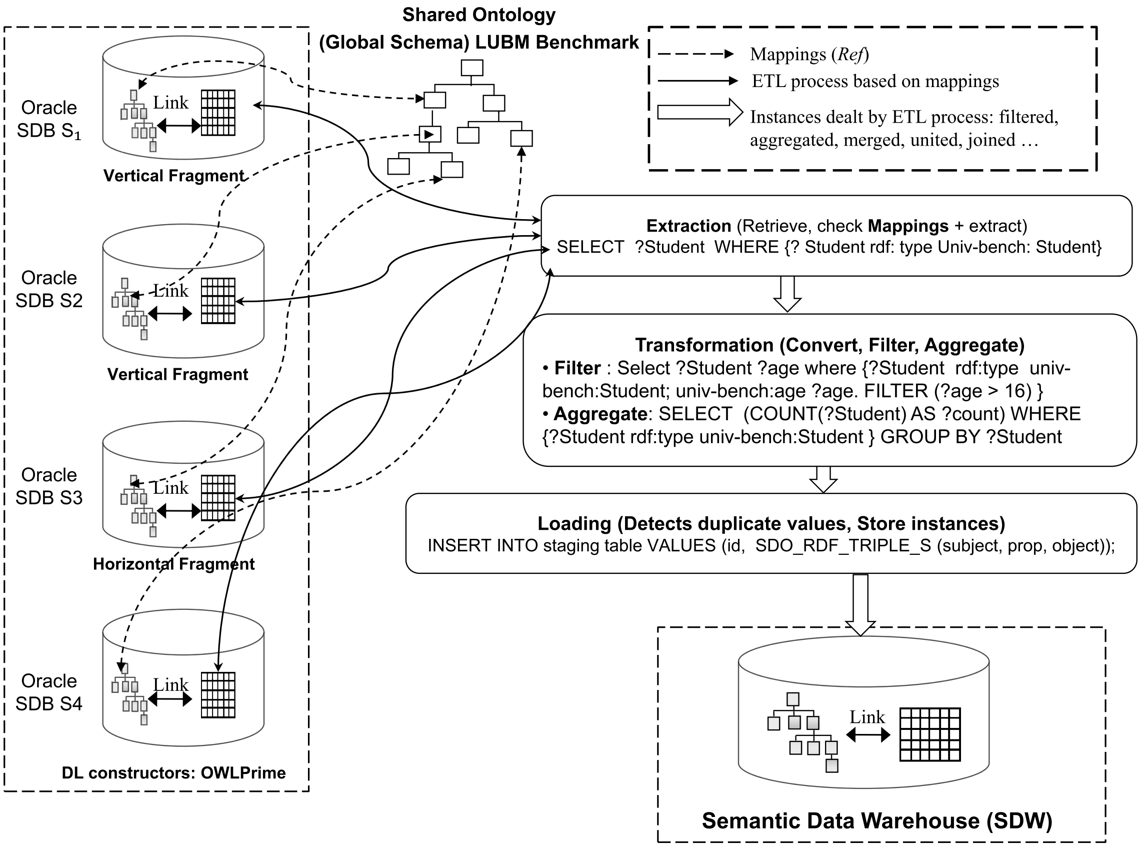
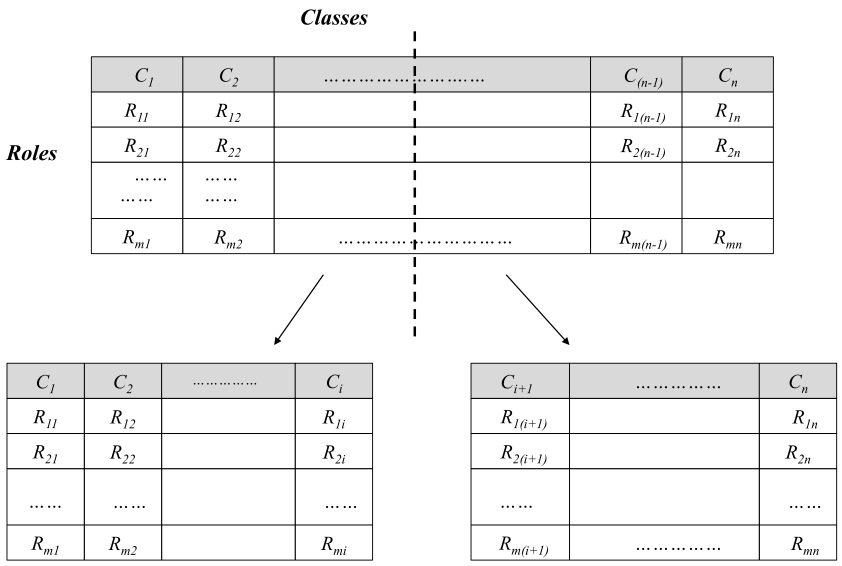

Shared Ontology
(Global Schema) LUBM Benchmark
Mappings (Ref)
ETL process based on mappings
Instances dealt by ETL process: filtered, aggregated, merged, united, joined …
Oracle SDB S1
Link
Vertical Fragment
Oracle SDB S2
Link
Vertical Fragment
Oracle SDB S3
Link
Horizontal Fragment
Oracle SDB S4
Link
DL constructors: OWLPrime
Extraction (Retrieve, check Mappings + extract)
SELECT ?Student WHERE {? Student rdf: type Univ-bench: Student}
Transformation (Convert, Filter, Aggregate)
• Filter : Select ?Student ?age where {?Student rdf:type univ-bench:Student; univ-bench:age ?age. FILTER (?age > 16) }
• Aggregate: SELECT (COUNT(?Student) AS ?count) WHERE {?Student rdf:type univ-bench:Student } GROUP BY ?Student
Loading (Detects duplicate values, Store instances)
INSERT INTO staging table VALUES (id, SDO_RDF_TRIPLE_S (subject, prop, object));
Link
Semantic Data Warehouse (SDW)

Fig. 5 General architecture of the approach: Integrating Oracle $\mathcal{SDB}$ using LUBM Benchmark

Fig. 6 Vertical ontological fragmentation based on projection of classes



tation as follows: $\Pi_{(C_i,C_j,\dots,C_m)}(C)$. The ontological projection is quite similar to the relational projection.

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 $\sigma_{(rest^{r_i},\dots,rest^{r_m})}(C_j)$, where $rest^{r_i}$ and C_j represent respectively, a restriction on the role r_i and the ontology class C_j. Figure 7 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.

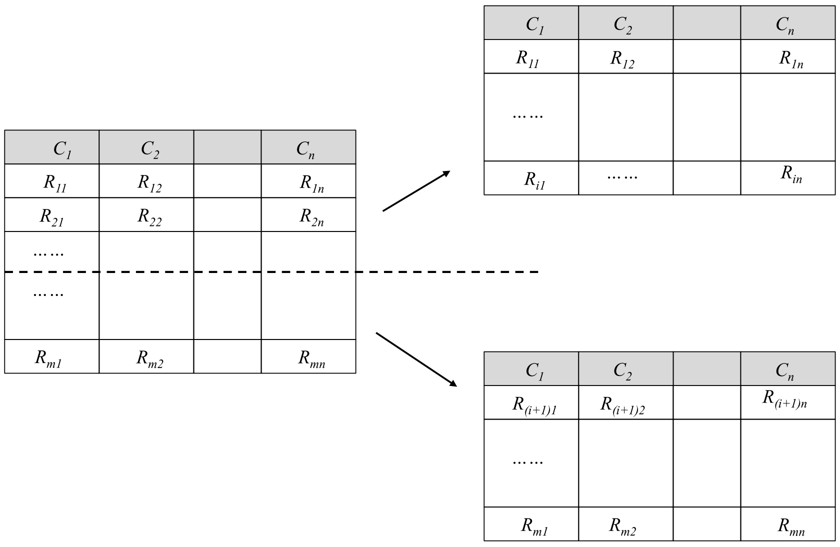


Fig. 7 Horizontal ontological fragmentation based on selection of roles

6.1.2 *Complex mappings*

Note that 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)

6.2 Instantiation of the generic integration framework

The generic integration framework $\langle G, S, M\rangle$ can be instantiated for our case study as follows:

The global schema G: The global schema is represented by LUBM ontology,[6] and is formalized by its Information Model as follows:

IM_{Oracle}: $\langle$Classes C, Properties P(Datatype Property and Object Property), Ref: (Operator, Expressions), OWLPrime$\rangle$.

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: $\langle IM_{Oracle}$, Individuals (triples), Pop is given in tables RDF_link$ and RDF_values$, Vertical, Vertical, type I$\rangle$. 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,*

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

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: $\langle IM_{Oracle}$ of each source, IM_{Oracle} of the global schema, Expression over G, Class of a source S, Intentional interpretation, (*Equivalent, Containment or Overlap* (owl:SubClassOf and owl:equivalentClass in OWLPrime))$\rangle$.

6.3 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.

  `PREFIX univ-bench1:`
  http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl1#

  `PREFIX univ-bench2:`
  http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl2#

  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.

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));

Figure 5 summarizes the steps of our proposal.

7 ETL as service

In the traditional $\mathcal{DW}$, the storage model followed one-2-one rule, where each warehouse table is stored following one storage layout. This is equivalent to an implicit assumption that the $\mathcal{DW}$ model will be usually deployed using the same logical and physical representations (the relational model). Consequently, the potential storage deployment model of the $\mathcal{DW}$ is always known in advance and frozen. With the evolution of the storage layouts, this hypothesis will not make sense. Storage deployment models can follow different representations according to specific requirements. A $\mathcal{DW}$ can be deployed using horizontal, vertical, hybrid models, NoSQL, etc.

To overcome this problem and provide more flexible and adequate storage deployment of $\mathcal{DW}$, we propose a service allowing designers the possibility to choose her/his favorite representation and storage layout. This service leverages the one-2-one rule to one-2-many rule. To achieve this goal, the integration of $\mathcal{SDB}$ into the $\mathcal{DW}$ by the means of an ETL process is implemented as a services, shown in Fig. 8: ETL as a Service (ETLaaS) and Physical storage as a Service (PDaaS).

The proposed tool is implemented in Java language and uses OWL API to access ontologies. The tool takes as inputs a set of requirements and a set of $\mathcal{SDB}$ that participates in the construction of the $\mathcal{DW}$. These sources reference a shared ontology formalized in OWL. The first step (conceptual design) is supported by a model-to-model transformation process. The warehouse ontology (DWO) is extracted as a module using ProSé plug-in available within protégé editor. Fact++ reasoner is invoked to classify the DWO class's taxonomy and to check its consistency.

The storage deployment of the $\mathcal{DW}$ is done according to the target platform. The diversity of storage models (vertical, horizontal, and hybrid) is handled by our proposed tool, where the suitable web service is invoked in order to translate the logical schema according to the physical model of the target DBMS. The ETL process is implemented in our tool such that technical details are hidden to the user. Each generic ETL operator is implemented as a Web Service. The proposed ETL algorithm consists thus in composing these

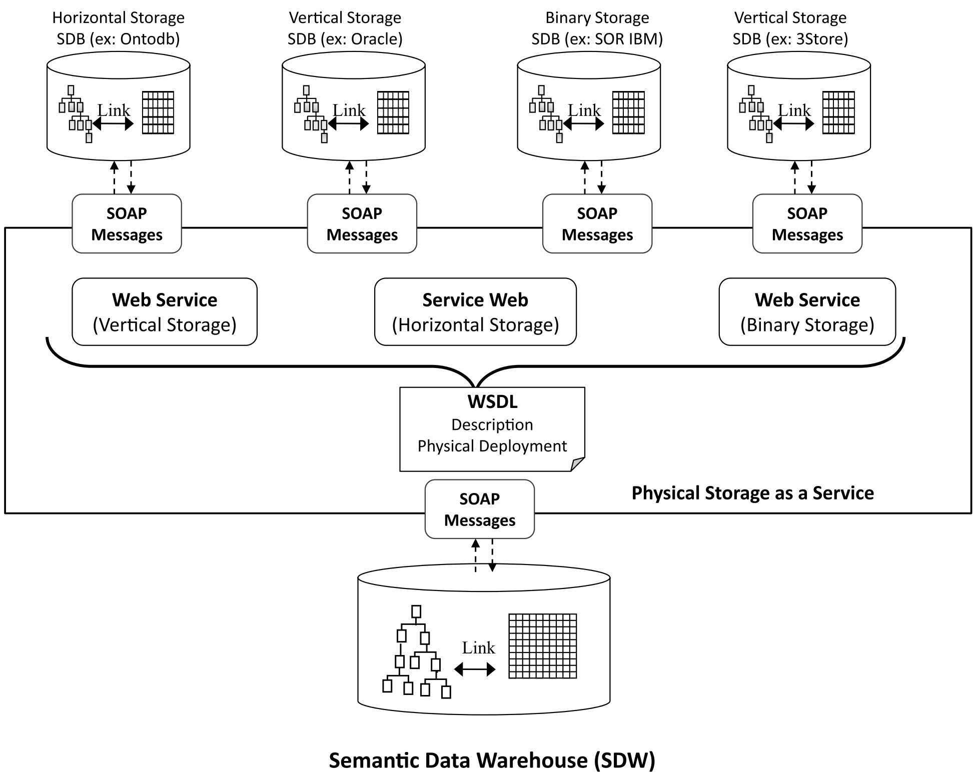
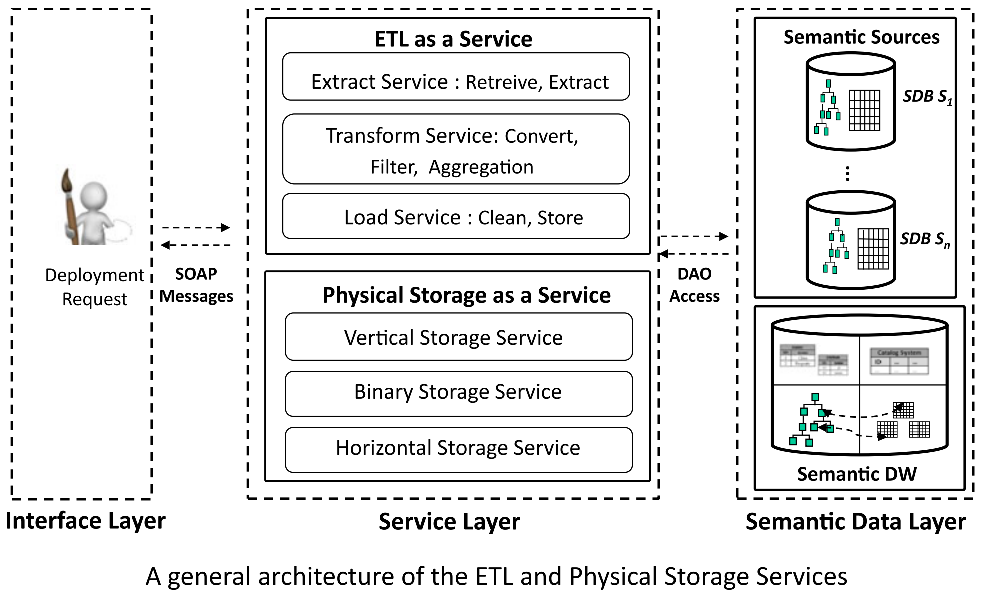

Horizontal Storage SDB (ex: Ontodb)
Vertical Storage SDB (ex: Oracle)
Binary Storage SDB (ex: SOR IBM)
Vertical Storage SDB (ex: 3Store)
Link
SOAP Messages
Web Service (Vertical Storage)
Service Web (Horizontal Storage)
Web Service (Binary Storage)
WSDL Description Physical Deployment
Physical Storage as a Service
Semantic Data Warehouse (SDW)

Fig. 8 ETL as Service

Fig. 9 A general architecture of the ETL and Physical Storage Services


ETL as a Service
Extract Service : Retreive, Extract
Transform Service: Convert, Filter, Aggregation
Load Service : Clean, Store
Physical Storage as a Service
Vertical Storage Service
Binary Storage Service
Horizontal Storage Service
Semantic Sources
SDB S1
SDB Sn
Semantic DW
Deployment Request
SOAP Messages
DAO Access
Interface Layer
Service Layer
Semantic Data Layer
A general architecture of the ETL and Physical Storage Services

Web Services. Based on the existing mappings between the schemes of $\mathcal{SDB}$ and the target $\mathcal{DW}$ schema, the implemented web services allows an automatic extraction of the appropriate data from the SDB sources, their transformation (filtering, conversion and aggregation) and the computation of the new values in order to obey to the structure of the $\mathcal{DW}$ classes. Then, data are loaded to the appropriate classes of the $\mathcal{DW}$ model. Each web service that accesses the persistent storages is implemented using Data Access Object (DAO) Design patterns [41]. DAO implements the access mechanism required to handle the $\mathcal{SDB}$. The DAO solution abstracts and encapsulates all access to persistent storage, and hides all implementation details from business components and interface clients. The DAO pattern provides flexible and transparent accesses to different storage layout. Based on the architecture of the $\mathcal{SDB}$ and the target $\mathcal{DW}$, the right object DAO is selected.

In order to obtain a generic implementation of the ETL process, we implemented our solution following service oriented architecture (SOA). SOA offers the loose coupling of the web services defined bellow, and interaction among them. It allows the integration of new web services without affecting the existing one. This provides the flexibility of the physical deployments of $\mathcal{DW}$. A demonstration video summarizing the different services offered by our proposal is available at: http://www.lias-lab.fr/forge/ETL/video.html.

7.1 Experimental study

In this section, we conduct experiments to evaluate the performance of our system by considering large data sets. Figure 9 describes the general architecture of the semantic $\mathcal{DW}$ deployment system. Two main services are distinguished: the *ETL as a service* and the *physical storage as a service*. Note that our system offers an automatic deployment service based on the chosen storage layout and the architecture of the target DBMS. During the experimental phase, we identified two criteria to be evaluated: the complexity of the proposed algorithm and the scalability in terms of instances of sources.

Data sets and environment Six sets of ontologies with respectively 1, 3, 6, 9, 12 and 15 universities are generated. The number of instances in each set is shown in Table 2. This generation is guided by the Data Generator tool (UBA) provided by the benchmark LUBM. This tool allows creating a domain ontology "University". Each university consists of 15 to 25 departments. In each department, different categories of Professors, Students, GraduateStudents, Courses, etc. can be found. The UBA generates OWL data over the LUBM ontology in the unit of a university. These data are repeatable and customizable, allowing us to specify the seed for random number generation. The generated data guarantee the inclusion of correct cases (e.g. respects the hierarchy of classes and relationship between the generated instances). However, we have added some contradictory cases to test their influence on the semantic data integration.

Our evaluations were performed on a laptop computer (HP) with an Intel (R) Core^{TM} i5-3320M CPU 2.60 GHZ and 4 GB of RAM and a 500 GB hard disk. We used Windows XP Professional OS and Java SDK 1.7. We use Oracle 11g release 2 DBMS.

SDB source creation We have created six Oracle $\mathcal{SDB}$ sources using simple and complex mappings. Oracle 11g offers only the data loading under N-Triple format (.nt). To

Table 2 Data Sets generated

Ontological Data Set	OWL Instances	N-Triple Instances
1 University	70 193	84 231
3 Universities	210 579	259 012
6 Universities	421 158	505 389
9 Universities	631 737	770 719
12 Universities	842 316	1 027 625
15 Universities	1 052 895	1 295 060

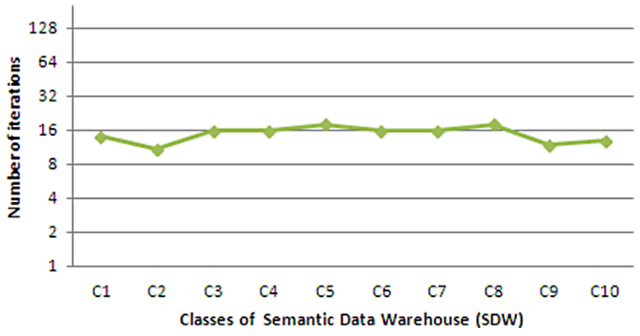


Fig. 10 Complexity of the proposed algorithm

meet this requirement, we used the Jena API which provides a converter named rdfcat. It enables the transformation of the generated OWL files to N-Triple format. The number of instances obtained for each ontological set is depicted in Table 2 column N-Triple Instances. These N-triple instances are loaded into the six $\mathcal{SDB}$ sources using Oracle SQL Loader.

Obtained results We consider the following scenario: we construct a semantic warehouse from the six $\mathcal{SDB}$ and the obtained warehouse is deployed. First of all, we examine the number of iterations of our algorithm to populate each $\mathcal{DW}$ ontological class (C_i). Figure 10 shows the obtained results. They show that the search space is not exponential regarding the number of semantic $\mathcal{DW}$ classes. Note that our algorithm is based on intentional mappings (concepts) and not on extensional mappings (number of instances). Indeed, although the average number of iterations per source is 15, in the worst case, our algorithm computes no more than 125 iterations. These findings verify the feasibility and efficiency of our approach in real-world cases.

Secondly, we evaluate the scalability and performance issues regarding the population of the constructed $\mathcal{DW}$. Figure 11 illustrates the results of the integration of the six $\mathcal{SDB}$. For these sources, we measure the time spent to integrate instances into the $\mathcal{DW}$. Notice axis x measures the number of instances shown in thousands.The time performance remains reasonable w.r.t. the size of the stored instances. This means that our method scales; the time spent to integrate the six $\mathcal{SDB}$ does not exceed 4 minutes which represents good performance.

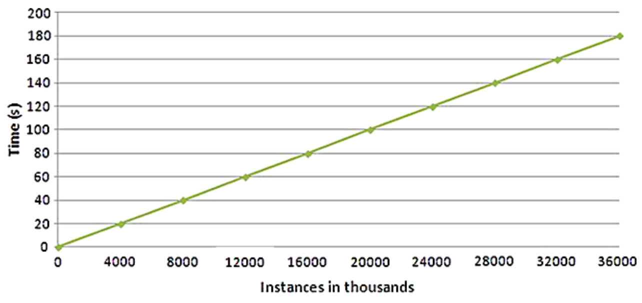


Fig. 11 The impact of the size of the stored instances during the population of the $\mathcal{SDB}$

8 Conclusion

In this paper, we have shown the evolution of information sources and diversity of storage models of the target warehouse. Inspired by Cloud computing technology, we propose a generic framework to construct a semantic $\mathcal{DW}$ from $\mathcal{SDB}$. Different kinds of mappings between source schemes and a global ontology schema of the warehouse 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. This framework contributes on defining ETL process at ontological level. To ensure this genericity, we proposed to use BPMN to model ETL process including the ten operators defined in the state of art. This model is then connected to the ontology model, which is a part of the warehouse components. As a consequence, all ETL operations are defined at ontological level which gives a high abstraction of designers and developers and hide the implementation aspects. Another contribution of this paper is the proposition of service for storing the ontology and its associated instances into the warehouse repository. This is due to the diversity of storage layouts and architectures of the target DMBS. 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 $\mathcal{DW}$ schema populated with instances loaded from $\mathcal{SDB}$. An operational tool is also developed supporting our proposal.

References

1. Halevy, A.Y., Ashish, N., Bitton, D., Carey, M.J., Draper, D., Pollock, J., Rosenthal, A., Sikka, V.: Enterprise information integration: successes, challenges and controversies. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 778–787 (2005)

2. Jarke, M., Jeusfeld, M.A., Quix, C., Vassiliadis, P.: Architecture and quality in data warehouses: an extended repository approach. Inf. Syst. **24**(3), 229–253 (1999)
3. Liu, X., Thomsen, C., Pedersen, T.B.: Mapreduce-based dimensional ETL made easy. J. Proc. VLDB Endow. **5**(12), 1882–1885 (2012)
4. Calvanese, D., Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Data integration in data warehousing. Int. J. Coop. Inf. Syst. **10**(3), 237–271 (2001)
5. Agrawal, D., Das, S., El Abbadi, A.: Big data and cloud computing: new wine or just new bottles? J. Proc. VLDB Endow. **3**(2), 1647–1648 (2010)
6. Agrawal, D., El Abbadi, A., Wang, S.: Secure and privacy-preserving data services in the cloud: a data centric view. J. Proc. VLDB Endow. **5**(12), 2028–2029 (2012)
7. Haase, P., Motik, B.: A mapping system for the integration of owl-dl ontologies. In: IHIS, pp. 9–16 (2005)
8. Gruber, T.R.: A translation approach to portable ontology specifications. In: Knowledge Acquisition, vol. 5, pp. 199–220 (1993)
9. Bellatreche, L., Nguyen Xuan, D., Pierra, G., Dehainsala, H.: Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases. Comput. Ind. **57**(8–9), 711–724 (2006)
10. Fankam, C.: 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)
11. Lu, J., Ma, L., Zhang, L., Brunner, J.S., Wang, C., Pan, Y., Yu, Y.: Sor: a practical system for ontology storage, reasoning and search. In: VLDB, pp. 1402–1405 (2007)
12. Wu, Z., Eadon, G., Das, S., Chong, E., Kolovski, V., Annamalai, M., Srinivasan, J.: Implementing an inference engine for rdfs/owl constructs and user-defined rules in oracle. In: ICDE, pp. 1239–1248 (2008)
13. Beneventano, D., Bergamaschi, S., Castano, S., Corni, A., Guidetti, R., Malvezzi, G., Melchiori, M., Vincini, M.: Information integration: the momis project demonstration. In: VLDB Journal, pp. 611–614 (2000)
14. Mena, E., Illarramendi, A., Kashyap, V., Sheth, A.P.: Observer: an approach for query processing in global information systems based on interoperation across pre-existing ontologies. Distrib. Parallel Databases **8**(2), 223–271 (2000)
15. Wache, H., et al.: Ontology-based integration of information—a survey of existing approaches. In: OIS, pp. 108–117 (2001)
16. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Conceptual modeling for ETL processes. In: DOLAP, pp. 14–21 (2002)
17. Trujillo, J., Luján-Mora, S.: A uml based approach for modeling ETL processes in data warehouses. In: ER, pp. 307–320 (2003)
18. Mazón, J.-N., Trujillo, J.: An mda approach for the development of data warehouses. In: JISBD, p. 208 (2009)
19. Wilkinson, K., Simitsis, A., Castellanos, M., Dayal, U.: Leveraging business process models for ETL design. In: ER, pp. 15–30 (2010)
20. Akkaoui, Z., Mazón, J., Vaisman, A., Zimányi, A.: Bpmn-based conceptual modeling of ETL processes. In: DaWaK, pp. 1–14 (2012)
21. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: A principled approach to data integration and reconciliation in data warehousing. In: DMDW, p. 16 (1999)
22. Calvanese, D., Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Data integration in data warehousing. Int. J. Coop. Inf. Syst. **10**(3), 237–271 (2001)
23. Luján-Mora, S., Vassiliadis, P., Trujillo, J.: Data mapping diagrams for data warehouse design with uml. In: ER, pp. 191–204 (2004)
24. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Modeling ETL activities as graphs. In: DMDW, pp. 52–61 (2002)
25. Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., Skiadopoulos, S.: A generic and customizable framework for the design of ETL scenarios. Inf. Syst. **30**(7), 492–525 (2005)
26. Shmueli, O., Tsur, S.: Logical diagnosis of ldl programs. New Gener. Comput. **9**(3/4), 277–304 (1991)
27. Luján-Mora, S., Trujillo, J.: Physical modeling of data warehouses using uml component and deployment diagrams: design and implementation issues. J. Database Manag. **17**(2), 12–42 (2006)
28. Tziovara, P., Vassiliadis, P., Simitsis, A.: Deciding the physical implementation of ETL workflows. In: DOLAP, pp. 49–56 (2007)
29. Simitsis, A., Vassiliadis, P., Sellis, T.-K.: Optimizing ETL processes in data warehouses. In: ICDE, pp. 564–575 (2005)
30. Simitsis, A., Wilkinson, K., Dayal, U., Castellanos, M.: Optimizing ETL workflows for fault-tolerance. In: ICDE, pp. 385–396 (2010)
31. Microsoft: Sql server integration services (2008). Available online: http://technet.microsoft.com/fr-fr/library/ms141026.aspx
32. Oracle: Oracle warehouse builder 11g release 2.1 (2009). Available online: http://www.oracle.com/technetwork/developer-tools/warehouse/documentation/library/index.html
33. IBM: IBM infosphere datastage (2008). Available online: http://www-01.ibm.com/software/data/infosphere/datastage/
34. Informatica: Informatica powercenter (2008). Available online: http://www.informatica.com/us/products/enterprise-data-integration/powercenter/
35. Skoutas, D., Simitsis, A.: Ontology-based conceptual design of ETL processes for both structured and semi-structured data. Int. J. Semantic Web Inf. Syst. **3**(4), 1–24 (2007)
36. Romero, O., Simitsis, A., Abelló, A.: Gem: requirement-driven generation of ETL and multidimensional conceptual designs. In: DaWaK, pp. 80–95 (2011)
37. Nebot, V., Berlanga, R.: Building data warehouses with semantic web data. Decis. Support Syst. **52**(4), 853–868 (2012)
38. Calvanese, D., Lenzerini, M., Nardi, D.: Description logics for conceptual data modeling. In: Logics for Databases and Information Systems, pp. 229–263 (1998)
39. Brockmans, S., Haase, P., Serafini, L., Stuckenschmidt, H.: Formal and conceptual comparison of ontology mapping languages. In: Modular Ontologies, pp. 267–291 (2009)
40. Guo, Y., Pan, Z., Heflin, J.: Lubm: a benchmark for owl knowledge base systems. J. Web Semant. **3**(2–3), 158–182 (2005)
41. Mayr, C., Zdun, U., Dustdar, S.: Model-driven integration and management of data access objects in process-driven soas. In: ServiceWave, pp. 62–73 (2008)

Nabila Berkani received her Engineer degree in Computer Science from National High School for Computer Science (ESI), Algiers, Algeria in 2005. After that, she spent seven years working as Project Manager at ATM Mobilis Telecom Operator, Algiers, Algeria. She received her Master degree in Computer science from the same school in April 2013. She is a Ph.D. student at National High School for Computer Science (ESI), Algiers. Her current research interests include Data Integration Systems, Data Warehouse Design, Ontology-based Modeling.

Ladjel Bellatreche is a Professor at National Engineering School for Mechanics and Aerotechnics (ISAE-ENSMA), Poitiers, France, where he joined it as a faculty member since Sept 2010. He leads the Data and Model Engineering Team of Laboratory of Computer Science and Automatic Control for Systems (LIAS). Prior to that, he spent eight years as Assistant and then Associate Professor at Poitiers University, France. He was a Visiting Professor of the Québec en Outaouais, Canada, a Visiting Researcher at Department of Computer Science, Purdue University, USA and Department of Computer Science of Hong Kong University of Science and Technology, China. He is also involved in Research Postgraduate Programs in Computer Science of several Universities and Schools in Africa. His research interests include data integration systems, data warehousing, physical design of VLDB, ontologies, personalization and recommendation. Prof. Ladjel Bellatreche has been actively involved in the research community by serving as reviewer for technical journals (IEEE TKDE, DKE, Distributed and Parallel Database Journal, etc.) and Editorial Board Member, International Journal of Reasoning-based Intelligent Systems, Inderscience, subject area editor of the *Scalable Computing Journal, Springer* and as an organizer/co-organizer of numerous international and National Conferences and Workshops (DAWAK, DASFAA, MEDI, WISE, EDA, JFO). Some recent conferences in which he is playing or has played major roles include DAWAK, MEDI, WISE Workshops. In addition, he served as a program committee member for over twenty international conferences and Workshops.

Selma Khouri received her Engineer degree in Computer Science from National High School for Computer Science (ESI), Algiers, Algeria in June 2007, and the Master degree in Computer science in September 2009. She is currently a lecturer at National High School for Computer Science, Algiers, Algeria. She is a Ph.D. student in a joint Ph.D. between Laboratory of Communication in Computer Science Systems (LCSI) at National High School for Computer Science (ESI), Algiers, Algeria and Laboratory of Computer Science and Automatic Control for Systems (LIAS) at National Engineering School for Mechanics and Aerotechnics (ISAE-ENSMA), Poitiers, France. Her current research interests include Data Warehouse Design, Data Integration Systems, Ontology-based Modelling and Requirements Engineering.