A Variety-Sensitive ETL Processes

Nabila Berkani[1(✉)] and Ladjel Bellatreche[2]

[1] École nationale Supérieure d'Informatique (ESI), Algiers, Algeria
n_berkani@esi.dz
[2] LIAS/ISAE-ENSMA – Poitiers University, Poitiers, France
bellatreche@ensma.fr

Abstract. Nowadays, small, medium and large companies need advanced data integration techniques supported by tools to analyse data in order to deliver real-time alerts and trigger automated actions, etc. In the context of rapidly technology changing, these techniques have to consider two main issues: **(a)** the variety of the huge amount of data sources (ex. traditional, semantic, and graph databases) and **(b)** the variety of storage platforms, where a data integration system may have several stores, where one hosts a particular type. These issues directly impact the efficiency and the deployment flexibility of ETL (Extract, Transform, Load). In this paper, we consider these issues. Firstly, thanks to Model Driven Engineering, we make generic different types of data sources. This genericity allows overloading the ETL operators. To show the benefit of this genericity, several examples of instantiation are described covering relational, semantic and graph databases. Secondly, a Web-service-driven approach for orchestrating the ETL flows is given. Thirdly, we present a fusion procedure that merges the set of heterogeneous instances and deployed according their favorite stores. Finally, our finding is validated through a proof of concept tool using the LUBM benchmark and YAGO $\mathcal{KB}$ and deployed in Oracle RDF Semantic Graph 12c.

1 Introduction

The past few decades have witnessed a spectacular explosion in the quantity of data sources available in various types and formats. This situation pushes small, medium and large companies to exploit this mine of data in order to achieve a high decision making in science, society, health, etc. This usually passes through the data integration process [11]. Plenty of commercial and open sources data integration solutions and tools exist in the market. When source data are extracted and materialized in an integration system such as a data warehouse, more specific techniques and tools implementing ETL (Extract, Transform, Load) are widely used [20]. Oracle Warehouse Builder, SAP Data Service, Talent Studio for Data Integration, IBM Infosphere Warehouse Edition, etc. are examples of these tools. The maturity of the ETL motivates researchers to make generic its whole workflow [5,26]. An ETL algebra composed of 10 generic operators (*Retrieve, Extract, Convert, Filter, Merge, Join, Union, Aggregate, Delete and Store*) has been proposed [20]. The signature of each operator is personalized

D. Benslimane et al. (Eds.): DEXA 2017, Part II, LNCS 10439, pp. 201–216, 2017.
DOI: 10.1007/978-3-319-64471-4_17

according the type of data sources and the target data warehouse ($\mathcal{TDW}$). By examining deeply the ETL techniques, we figure out that they mainly concentrate on the *traditional types of data* such as relational databases [24] – which has reigned several decades – and recently semantic databases ($\mathcal{SDB}$) [3,14]. In the context of rapidly technology changing, several new types of databases appear (e.g., Graph databases, NoSQL, Time Series, Knowledge bases ($\mathcal{KB}$) such as Yago [21]) and consequently they became candidate for the data integration. This phenomenon is called by the *Variety of sources* [7].

The variety does not only impact data sources, but also the storage of the $\mathcal{TDW}$, where multi-stores are well-adapted to achieve high performance of data accesses. More concretely, we passed from $(n-1)$ scenario, where n heterogeneous sources are integrated into $\mathcal{TDW}$ deployed on one store to $(n-m)$ scenario, where the $\mathcal{TDW}$ may be deployed in several stores, where each one may store a specific type of data [10]. To deal with the variety of sources, we propose the usage of Meta-Driven Engineering (MDE) and then overload the ETL operators to deal with specific type of source. Inspired from the Meta-Object Facility (MOF)[1], we make generic different sources in order to deal with their variety. The MOF describes a generic framework in which the abstract syntax of modelling languages can be defined [15]. It has a hierarchical structure composed of four layers of meta-data corresponding to the different levels of abstraction: the instance layer (M0), the model layer (M1), the meta-model layer (M2) and the meta-meta-model layer (M3). Each layer defines a level to ensure the consistency and the correctness of the instance model syntax and semantics at each level of abstraction.

This generic model can be easily exploited by ETL operators, where each one is overload. An operator overloading (as in C++ language) allows a programmer to define the behaviour of an operator applied to objects of a certain class the same way methods are defined [6]. In our context, each ETL operator will be overload to deal with the diversity of each type of sources.

To offer the designers the possibility to deploy their $\mathcal{TDW}$ on a multi-stores, we exploit the *Store operator* of ETL. It can be associated with a Service Web that orchestrates the ETL flows and distributes the data over the stores according their storage formats.

In this paper, we detail our generic model using MDE. We give examples of its instantiation from three types of data sources: relational, semantic and graph databases. The ETL operators are then overloaded for these types. Thanks to the *Store* operator, the multi-store deployment is guaranteed. Our proposal is implemented and experimented.

The rest of this paper is organized as follows. We give an overview on the evolution of the ETL in Sect. 2. In Sect. 3, we give a formalization of three main classes of databases (relational, semantic and graph) and a motivating example. In Sect. 4, a generalization of ETL elements are given by the means of MDE techniques and the process to overload the ETL operators. In Sect. 5, the deployment methodology of a data warehouse on multi-store system is developed.

[1] http://www.omg.org/mof/.

A case study is proposed and various experiments are presented. Section 7 concludes the paper.

2 Related Work

In this section, we give an overview of the most important studies on the ETL and the efforts to making them variety-sensitive. The first studies on ETL dealt with sources considering their physical implementations such as: (i) their deployment platforms (centralized, parallel, etc.) and (ii) their storage models (e.g. tables, files). In [23], a set of algorithms was proposed to optimize the physical ETL design. Simitsis et al. [18] have proposed algorithms for optimizing the efficiency and the performance of ETL process. Other non-functional requirements such as freshness, recoverability, and reliability have also been considered [19]. The work of [13] proposes an automated data generation algorithm assuming the existing physical models for ETL to deal with the problem of data growing. In order to hide the physical implementations, several research efforts have been proposed. The first category of these studies attempts to consider the logical level of data sources. In this perspective, [27] proposed an ETL workflow modelled as a graph, where its nodes represent activities, record-sets, attributes, and its edges describe the relationships between nodes that define ETL transformations. In [25], a formal ETL logical model is given using L_{DL} [17] as a formal language for expressing the operational semantics of ETL activities. The second category of these studies considered the conceptual level of sources. Approaches based on ad-hoc formalisms [26], on standard languages using UML [22], model driven architecture (MDA) [8], BPMN [1,28] and mapping modelling [4,12] have been proposed. The third category use ontologies as external resources to facilitate and automate the conceptual design of ETL process. [20] automated the ETL process by constructing an OWL ontology linking schemes of semi-structured and structured (relational) sources to a target data warehouse ($\mathcal{DW}$) schema. Other studies like [14] consider data source provided by the semantic Web and annotated by OWL ontologies. However, the ETL process in this work is dependent on the storage model used for instances which is the triples.

Based on this brief overview, we figure out the effort deployed by the research community in generalizing the ETL processes by going from the physical level to the semantic level of the sources. In [27], a generic model of ETL activities that plays the role of a pivot model has been proposed, but without MDE techniques. [20] has defined an ETL algebra with 10 generic operators. The main drawbacks of these approaches are: they deal with traditional types of sources (relational and XML schemes) and they make an implicit assumption that the data warehouse is deployed on one system usually relational.

3 Background and a Motivating Example

In this section, we give an overview on the most important types of databases: relational, semantic and graph databases adopted by a large number of sources.

Then, a motivating example is considered to illustrate the basic ideas behind our proposal.

3.1 Formalization of Databases

A relational database is defined by set of tables, attributes, instances and constraints.

A semantic database ($\mathcal{SDB}$) is formally defined as follows [3]: $<OM, I, Pop, SL_{OM}, SL_I>$, where:

- OM: $<C, R, Ref, formalism>$ is the ontology model of the $\mathcal{SDB}$; where C and R denote respectively concepts and roles of the model; Ref is a function defining terminological axioms of a DL TBOX (Terminological Box) [2], (e.g., Ref(Student) $\rightarrow$(Person $\cap \forall$ takesCourse(Person, Course))) and $Formalism$ is the formalism followed by the global ontology model like RDF, OWL, etc.);
- I: presents the instances (the ABox) of the $\mathcal{SDB}$;
- Pop: C $\rightarrow 2^I$ is a function that relates each concept to its instances;
- SL_{OM}: is the Storage Layout of the ontology model (vertical, binary or horizontal) [9]; and
- SL_I: is the Storage Layout of the instances I.

A graph database usually used to represent knowledge bases through a graph G whose nodes (V), edges (E) and labels (L_v, L_e) represent respectively classes, instances and data properties, object properties and **DL** constructors. Neo4J[2] is an example of a storage system of graph databases.

3.2 Motivating Example

To explicit the basic ideas behind our proposal, let us consider a scenario, where a governmental organisation wants constructing a data warehouse to analyse the performance of students in universities. To do so, this organisation considers four data sources with a high variety. The particularity of these sources is that they are derived from the benchmark related to the universities (LUMB[3]) and the Yago[4] knowledge base. The details of these sources are given below: S_1 is a MySQL relational databases with the following schema composed of tables and attributes: *Student(name), Course(title), University*, S_2 is a Berkeley XML $\mathcal{DB}$ with a schema composed of elements and attributes: *GraduateStudent(name), GraduateCourse(title), University*, S_3 is an Oracle RDF $\mathcal{SDB}$ composed of classes, properties: *Student(name), Publication, University*, and S_4 is a Neo4j Graph $\mathcal{DB}$ with nodes, edges: *Person, Student(name), Publication, PublicationAuthor, University*.

The obtained warehouse has two stores Semantic Oracle and Mongodb. In this context, the different ETL operators have to be overloaded to deal with

[2] https://neo4j.com/product/.
[3] http://swat.cse.lehigh.edu/projects/lubm/.
[4] www.yago-knowledge.org/.

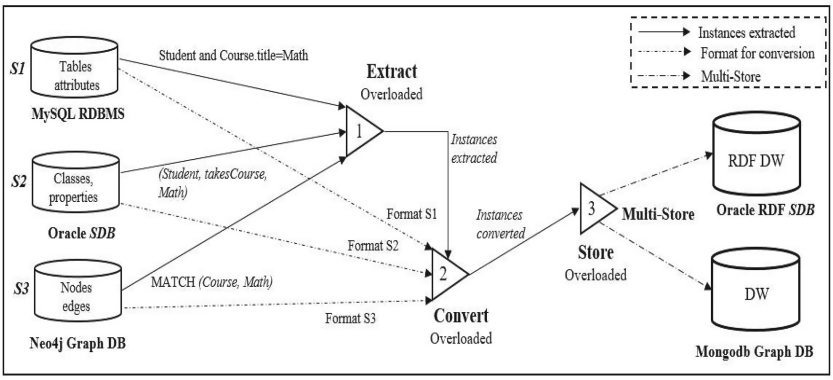


Fig. 1. An example of ETL operator overloading

this variety. Figure 1 describes the whole architecture of the ETL process, where *Extract* and *Convert* operators are overloaded. As we see, they have the same name, but different signatures. Based on the format of each store, the Store operator is also overloaded.

4 Generalisation of ETL Elements

Before discussing our proposal in overloading ETL operators, we first formalize the ETL process and its operators. An ETL process is defined as 5-tuples as follows: $<InputSet, OutputSet, Operator, Function, ETLResul>$, where:

InputSet: represents a finite set of input elements describing data sources. Each source has its own format and storage layout. To make generic the representation of data sources, we propose to generalize them using MOF initiatives. The obtained meta-model is composed of conceptual entities and their attributes. In addition, links between entities are also represented via associations. We also represent several semantically restrictions, such as primary and foreign keys. Figure 2, part (a), illustrates the fragment of our meta-model. Table 1 is an instantiation of relational, semantic and graph databases sources.

OutputSet: is a finite set of intermediate or target elements. The output of the ETL process can be either the intermediate output (sub process) or the final output (ETL process). The final output corresponds to the target data stores, where the schema of each store can be seen as an instance of our meta-model (part (a) of Fig. 2).

Operator: is a set of operators commonly encountered during the ETL process in [20]. By analysing these operators, we propose to decompose them into four categories: (1) loading class, (2) branching class, (3) merging class and (4) activity class.

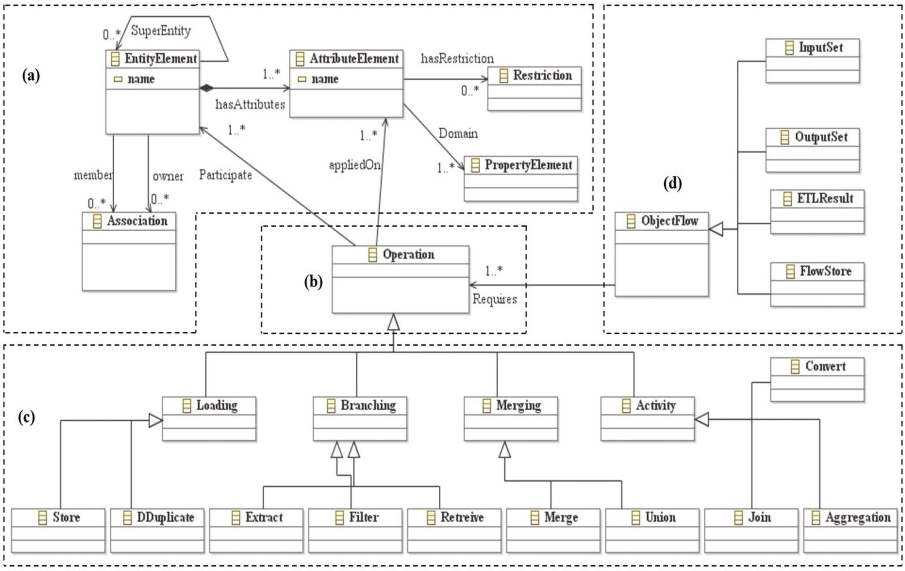


Fig. 2. Excerpt of ETL meta models

Table 1. Sample of InputSet and OutputSet databases.

Elements	Databases		
	Relational	Ontological	Graph
Entity	Table	Class	Node
Association	Table	Object_property	Edge
Attribute	Column	Data_property	Node
Property	Domain of values	Data type	Domain of values
Restriction	Primary key	SameAs	Node

- **Branching Class:** delivers multiple output-sets which can be further classified in Filter operations based on conditions or Extract and Retrieve operations that handle with the appropriate portion of selected data.
- **Merging Class:** fuses multiple data incoming from data sources. We identify two possible operations: (i) Merge operation applied when the data belong to attributes related to entities of the same source; (ii) Union operation applied when data belong to entities incoming from different data sources
- **Activity Class:** represents points in the process where work is performed. It corresponds to all operations of join conversion and aggregation. Join operations is applied when data belong to different entities. Conversion operation is applied on data having different format in order to unify it and adapt it to the target data stores. The aggregation operation is done depending on the schema of the target data stores applying needed functions (count, sum, avg, max, min).

– **Loading Class:** represents the point of data quality by the detection of duplicated data and cleaning them before their loading in the target data store.

Based on this, we propose a meta models of these operations (part (c) of Fig. 2). The generic formalization of each operator is given by:

- $Retrieve(S, E, A, R)$: retrieves data D of attributes A related to entities E from Source S;
- $Extract(S, E, A, R, CS)$: enables the data D extraction of A related to entities E from source S satisfying constraint CS;
- $Merge(S, E_1, E_2, A_1, A_2, D_1, D_2)$: merges data D_1 and D_2 belonging to the source S;
- $Union(S_1, S_2, A_1, A_2, D_1, D_2)$: unifies data D_1 and D_2 belonging to different sources S_1 and S_2 respectively;
- $Filter(S, E, A, D, CS)$: filters incoming data D, allowing only values satisfying constraints CS;
- $Join(S, E_1, E_2, A_1, A_2, D_1, D_2)$: joins data D_1 and D_2 having common attributes A_1 and A_2;
- $Convert(S, E, A, D, F_S, F_T)$: converts incoming data D from the format F_S of source S to the format of the target data store F_T;
- $Aggregate(S, E, A, D, F)$: aggregates incoming data D applying the aggregation function F (count, sum, avg, max) defined in the target data-store.
- $DD(D)$: detects and deletes duplicate values on the incoming data D;
- $Store(T, E, A, D)$: loads data D of attributes A related to entities E in the target data store T.

Function: is a function over a subset of Input-Set applied in order to generate data satisfying restrictions defined by any ETL operator.

ETLResult: is a set of output elements representing the flow.

4.1 Overloading Operators

In this section, we show the mechanism to overload ETL operators by considering semantic and graph databases.

In the case of a semantic database, the signature of overload operators is as follows:

- $Retrieve(S_i, C, I)$: retrieves instances I related to classes C from Source S_i;
- $Extract(S_i, C, I, CS)$: extracts instances I related to classes C from source S satisfying constraint CS;
- $Merge(S_i, C_1, C_2, I_1, I_2)$: merges instances I_1 and I_2 related to the classes C_1 and C_2 respectively and belonging to the same source S_i;
- $Union(S_1, S_2, C_1, C_2, I_1, I_2)$: unifies instances I_1 and I_2 related to C_1 and C_2 respectively and belonging to different sources S_1 and S_2 respectively;
- $Filter(S_i, C, I, CS)$: filters incoming instances I related to C, allowing only the values satisfying constraints CS;
- $Join(S_i, C_1, C_2, I_1, I_2)$: joins instances I_1 and I_2 related to C_1 and C_2 respectively and having common object properties;
- $Convert(S_i, C, I, F_S, F_{TDW})$: converts incoming instances I from the format F_S of source S_i to the format of the target $\mathcal{TDW}$ (F_{TDW});
- $Aggregate(S, C, I, F)$: aggregates incoming instances I related to C applying the aggregation function F (count, sum, avg, max) defined in the $\mathcal{TDW}$.
- $DD(I)$: detects and deletes duplicate values on the incoming instances I;
- $Store(TDW, I)$: loads instances I in target $\mathcal{TDW}$.

In the case of a graph database, the signature of overload operators is as follows:

- $Retrieve(G, V_j, L_j)$: retrieves a node V_j having an edge labeled by L_j of G.
- $Extract(G, V_j, CS)$: extracts, from G, the node V_j satisfying CS.
- $Convert(G, G_T, V_i, V_T)$: converts the format of the node V_i to the format of the target node V_T. The conversion operation is applied at instance level.
- $Filter(G, V_i, CS)$: applied on V_i node, allowing only instances satisfying CS.
- $Merge(G, V_i, V_j)$: merge instances denoted by nodes V_i, V_j in same graph G.
- $Union(G, G_T, V_i, V_j, E_j)$: links nodes that belongs to different sources. It adds in the target graph G_T, both nodes V_i and V_j and link them by an edge E_j.
- $Join(G, G_T, V_i, V_j, E_j)$: joins instances whose corresponding nodes are $V_i \in G$ and $V_j \in G_T$. They are linked by an *object property* defined by an edge E_j.
- $Store(G_T, V_j)$: loads instances denoted by nodes V_j to the target graph G_T.
- $DD(G_T, CS)$: sorts the graph G_T based on CS and detects duplication.
- $Aggregate(G_T, V_j, Op)$: aggregates instances represented by the nodes V_j.

Some primitives need to be added to manage the ETL operations required to build the ETLgraph such as:

- $AddNode(G_T, V_j, E_j, L_j)$: adds node V_j, edge E_j, label L_j required to G_T.
- $UpdateNode(G_T, V_j, E_j, L_j)$: updates node V_j, edge E_j, label L_j in G_T.
- $RenameNode(G_T, V_j, E_j, L_j)$: renames node V_j, edge E_j, label L_j in G_T.
- $DeleteNode(G_T, V_j, E_j, L_j)$: deletes node V_j, edge E_j, label L_j from G_T.
- $SortGraph(G_T, V_j, CS)$: sorts nodes of G_T based on some criteria CS to improve search performance.

Our goal is to facilitate, manage and optimize the design of the ETL process during the initial design, deployment phase and during the continuous evolution of the $\mathcal{TDW}$. For that, we enrich the existing ETL operators with *split*, *context* and *Link* operators elevating the clean-up and deployment of ETL process at the conceptual level.

- $Split(G, G_i, G_j, CS)$: splits G into two sub-graphs G_i and G_j based on CS.
- $Link(G_T, V_i, V_j, CS)$: links two nodes V_i and V_j using the rule CS.
- $Context(G, G_T, CS)$: extracts from the graph G a sub-graph G_T that satisfies the context defined by restrictions CS using axioms.

5 Deployment on a Multistores System

In this section, we propose a methodology to satisfy the $n-m$ scenario discussed in the Introduction. To do so, we have to consider three issues: consolidation of schemas, fusion of instances, and deployment.

5.1 Consolidation ETL Algorithm

Algorithm 1 describes in details the overloading of ETL operators in the context of semantic and graph data sources. It is based on mappings defined between data sources schemes and global schema. We used mappings described in [3].

5.2 Fusion Procedure

In this section, we propose a fusion method to merge different input data sources representations based on the target model chosen by the designer. Our solution is based on the *Graph Property* model presented above [16]. The property graph is common because modellers can express other types of models or graphs by

Algorithm 1. Overloading ETL Process Algorithm

Input: $\mathcal{IO}$ or Contextual $\mathcal{KB}$, S_i: Local sources $\mathcal{SDB}$
Output: $\mathcal{TDW}$ (schema + instances)

```
1: V_Si := ∅; ETL_G:= Graph(Tbox(kb)); V_kb := GetNodes(kb);
2: if Input is IO then
3:    Input_cond := (C : Class of ontology IO);
4: else if Input is KB then
5:    Input_cond := (V_i ∈ V_kb ∧ (V_i isClass));
6: end if
7: for each Input_cond do
8:    for Each S_i  do
9:       if Equivalent or complete mappings (N_Si,N_KB) ∨ (C_Si, C_IO) then
10:         if Input is IO then
11:            C := IdentifyClass(C_TDW, C_i);
12:         else if Input is KB then
13:            V_i:= IdentifyNode(ETL_G, V_i);
14:            E_i:= IdentifyEdge(ETL_G, E_i);
15:         end if
16:      else if sound or overlap mappings (N_Si,N_KB) ∨ (C_Si, C_IO) then
17:         if Input is IO then
18:            Const := ExtractConstraint(C_TDW, C_i);
19:         else if Input is KB then
20:            Const := ExtractNeighbor(ETL_Graph, V_i);
21:         end if
22:      end if
23:      if (Input is IO) then
24:         if (Const isDataTypeProperty)  then
25:            I:= Convert(C_j, I, const);
26:            I:= Filter(C_j, I, Const);
27:         else if (Const isObjectProperty) then
28:            I:= Join(C_j, C_i, I, Const);
29:         else if (Const isAxiom) then
30:            I:= Aggregate(C_j, C_i, I, Const);
31:         end if
32:         I:= MERGE(C_Si, I); I:= UNION (C_Si, C_IO, I);
33:         STORE(IO, C_i, DD(I));
34:      else if (Input is KB) then
35:         if (Const isDataTypeProperty) then
36:            I:= Convert(V_j, I, const);
37:            I:= Filter(V_j, I, Const);
38:         else if (Const isObjectProperty) then
39:            I:= Join(V_j, V_i, I, Const);
40:         else if (Const isAxiom) then
41:            I:= Aggregate(V_j, V_i, I, Const);
42:         end if
43:         I:= MERGE(N_Si, I); I:= UNION (N_Si, N_kb, I);
44:         for Each I  do
45:            ETL_G := addEdge(ETL_G, N_i, edge, I);
46:            ETL_G := addNode(ETL_G, N_i, edge, I);
47:         end for
48:         ETL_G := Filter(ETL_G, DD(N_i), Null-values);
49:         STORE(ETL_G);
50:      end if
51:   end for
52: end for
```

adding or abandoning particular elements. To do so, we propose to use the primitives proposed previously. They enable designers to adds, deletes and renames graph elements in order to manage the ETL flow generated and adapt it to the target storage layout chosen. An example of *addnode* primitive is done as follows:

```
- Sparql query language :
construct ?V
where {GRAPH  :?G {?V rdf:type name-space:Class}}
- Using Cypher query language for Neo4j graph database:
MERGE (<node-name>:<label-name>
{<Property1-name>:<Pro<rty1-Value> ..... <Propertyn-name>:<Propertyn-Value>}
```

On the basis of items presented previously, we have identified three particular cases:

Deployment of $\mathcal{KB}$ on $\mathcal{SDB}$: the RDF graph allowing the representation of $\mathcal{KB}$ deployed on $\mathcal{SDB}$ can be obtained by restricting labels of the nodes and edges to Uniform Resource Identifiers (URIs) and not allowing node/edge attributes;

Deployment of $\mathcal{KB}$ on graph database: using graph property having directed, labelled, attributed nodes and edges will allow a deployment of $\mathcal{KB}$ on a graph system;

Deployment of traditional data on graph: starting from a property graph, we generate a standard semantic graph by discarding the nodes/edges attributes. Having a semantic graph, we consider the nodes as attributes/data of traditional data, labels nodes are either *attributes or data*, edges as relationships between data and attributes of traditional data, labels edges can be either *has_data or has_attributes*.

5.3 Deployment of ETL Process

Storage deployment models can follow different representations according to specific requirements. A $\mathcal{TDW}$ can be deployed using horizontal, vertical, hybrid models, NoSQL, etc. [9]. In our case, we choose to deploy the $\mathcal{TDW}$ into vertical representation using Oracle DBMS which offers a storage model to represent instances and graphs using Oracle RDF Semantic Graph. We translated the $\mathcal{TDW}$ schema into vertical relational model, then generated an N-Triple file, load it into a staging table using Oracle's SQL*Loader utility. We applied the ETL Algorithm to populate the target schema.

6 Experimental Study

In order to illustrate the feasibility of our approach, we use our motivating example (cf. Section 3). We choose Oracle semantic database system to implement the sources and the warehouse. Oracle 12c release 2 delivers *RDF Semantic Graph features* as part of Oracle Spatial and Graph. With native support for RDF and OWL standards for representing semantic data, with SPARQL for query

language. Oracle has defined two subclasses of DLs: OWLSIF and a richer fragment OWLPrime. Note that *OWLPrime* limits the expressive power of DL formalism in order to ensure decidable query answering. The proposed Algorithm 1 was implemented using the overload of ETL operators in order to integrates the created sources into the DW taking in account their heterogeneity. Note that generic ETL operators defined in the previous section are expressed on the conceptual level. Therefore, each operation has to be 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. Here an example of $\mathcal{KB}$ aggregation ETL operator translation to SPARQL:

```
PREFIX yago: http://yago-knowledge.org/resource/yago.owl#
AGGREGATE: Aggregates incoming record-set.
Select (Count(?Instance) AS ?count) Where {
GRAPH  :?G {?Instance rdf:type yago:Class}}
Group By ?Instance.
```

The proposed tool is implemented in Java language and uses JENA API to access ontologies and a $\mathcal{KB}$. Each generic ETL operator is implemented as a Web Service Restful using Java overload polymorphism implementation. The restful web service is implemented is such way to consider the overload resolution. Each ETL operator is overloaded by determining the most appropriate definition to use. It compares the argument type used to call the appropriate service restfull with the parameter types specified in the definitions. This will allow managing the different representations of input data (instances and graph). The proposed ETL algorithm consists then in orchestrating the Web services.

Each Web service that accesses the persistent storage is implemented using Data Access Object (DAO) Design patterns[5]. DAO implements the access mechanism required to handle the different input representations. 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. 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. The application implements an orchestration of web services in early binding. Indeed, each web service is implemented in such way that parameters and variables are detected and checked at compile time. Figure 3 describes the whole architecture of the ETL and MultiStore Services.

A demonstration video summarizing the different services offered by our proposal is available at: https://youtu.be/zbtl1qMvPOU.

6.1 Evaluation Study

In this section, we present the performance of our approach through a set of experiments considering an Ontology and large $\mathcal{KB}$. Four criteria are used to

[5] http://www.oracle.com/technetwork/java/dataaccessobject-138824.html.

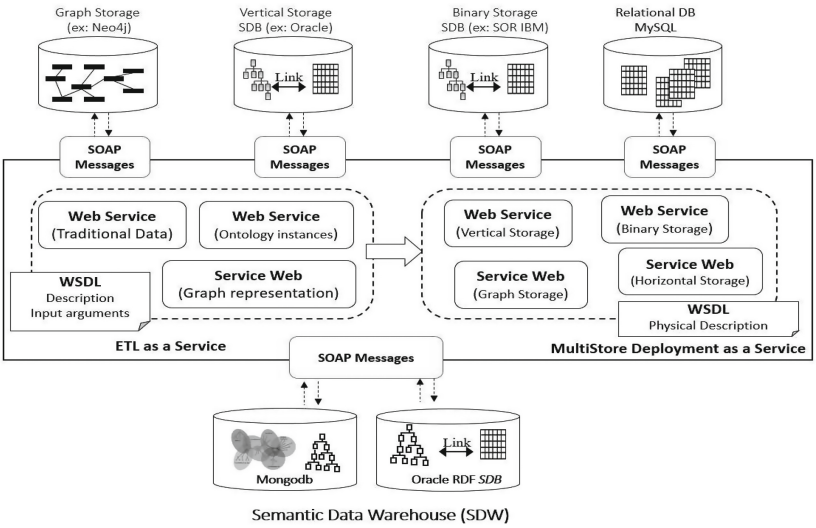


Fig. 3. A general architecture of the ETL and MultiStore Services.

evaluate our proposal: (i) complexity of the proposed ETL algorithm, (ii) evaluation time per ETL operators before and after overloading, (iii) scalability of the ETL process, (iv) inference performance.

Environment of our experiments. Our experiments are based on LUBM ontology and YAGO $\mathcal{KB}$ (version 3.0.2). The architecture of the YAGO system is based on themes. Each theme is a set of facts. A fact is the equivalent of an RDF triple (s,p,o). YAGO has defined the context relation between individuals [21], which we used to extract the set of themes related to our context study which is university domain. The resulting contextual YAGO $\mathcal{KB}$ contains around 5, 9×10^6 triples. Note that five (5) sets of triples were generated using LUBM benchmark and Yago knowledge base.

(1) *Deployment of Data Sources and $\mathcal{TDW}$*: We have created five Oracle *SDBs* using generated data-sets and deployed the $\mathcal{TDW}$ schema using Oracle *SDB*. We chose N-Triple format (.nt) to load instances using Oracle SQL*Loader.
(2) *Oracle Database Tuning: $\mathcal{TDW}$* schema was optimized using Btree indexing triples and sparql query hints. Some PL/SQL APIs are also invoked after each load of significant amount of data. The API SEM_PERF.GATHER_STATS Collects stats for sources models and SEM_APIS.ANALYZE_MODEL for $\mathcal{TDW}$ model in the semantic network graph. The memory SGA and PGA are also increased to 2GB.
(3) *Inference Engine:* Oracle has incorporated a reasoner engine defined based on *TrOWL* and *Pellet* reasoners. Oracle provides full support for native inference in the database for RDFS, RDFS++, OWLPRIME, OWL2RL, etc. It uses forward chaining to do the inference. It compiles entailment rules

directly to SQL and uses Oracle's native cost-based SQL optimizer to choose an efficient execution plan for each rule. The following is an example of user defined rules applied, they are saved as records in tables. $Rule_1$: co-author rule: $authorOf(?A1, ?P) \wedge authorOf(?A2, ?P) \rightarrow coAuthor(?A1, ?A2)$.

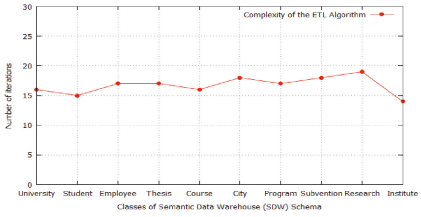
(4) *Hardware:* Our evaluations were performed on a laptop computer (HP EliteBook 840 G2) with an Intel(R) CoreTM i5-5200U CPU 2.20 GHZ and 8 GB of RAM and a 500 GB hard disk. We use Windows10 64bits. Cytoscape[6] is used for visualization.

Obtained results. We evaluate our proposal based on the following criteria:

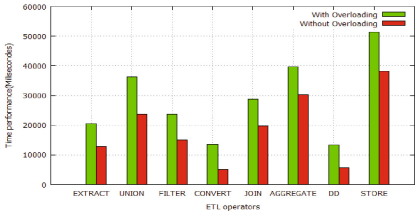
Criterion 1: ETL Algorithm Complexity. The algorithm is implemented based on semantic ontologies (classes and properties) and graph theory, where nodes represents concepts and instances, edges for roles and labels for definitions. We examine the number of iterations of our algorithm to generate ETL process as flow or graph. In this case, we are interesting on the time complexity. The algorithm is based on concepts searches (Tbox for intentional mappings i.e. mappings only between classes and properties and not between instances). The time complexity is $O(n)$, where n represents the number of involved classes or nodes. Figure 4a shows the number of iterations by classes. It indicates a polynomial time. This finding shows the feasibility and efficiency of our approach.

Criterion 2: Evaluation Time Per ETL Operator Before and After Overloading. We run the ETL Algorithm for both scenarios (without overload for ontology and $\mathcal{KB}$, and with overload for both) to populate the target schema of semantic $\mathcal{TDW}$. We measure the time spent to run each ETL operator. Figure 4b shows the results obtained. Our approach improves the performance time spent by overloaded ETL operator in an 18%. This is due to one call of the functions related to ETL operators done by the compiler, instead of multiple calls in a case without overload.

Criterion 3: Scalability of the Proposed Solution. The ETL Algorithm populates the target schema of semantic $\mathcal{TDW}$ using an overload of ETL operators.



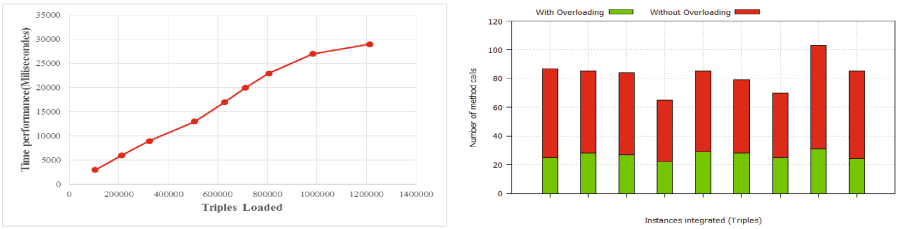
(a) Complexity of the proposed ETL algorithm



(b) Evaluation time per ETL operator before and after overloading.

Fig. 4. Complexity and evaluation time of the ETL process

[6] http://www.cytoscape.org/.



(a) Scalability of the proposed solution. (b) Number of method calls

Fig. 5. Scalability and number of calls

We measure the time spent to integrate data sources having different sizes. Note that time spent to load all instances is equal 3, 2 min. Figure 5a illustrates the results obtained where for each triple size loaded using overload approach, corresponding time performance is shown in milliseconds. The result remains reasonable w.r.t. the size of the stored instances. This is proof the scalability of our approach.

Criterion 4: Number of Method Calls. We consider a set of $\mathcal{SDB}$ participating in the $\mathcal{TDW}$. We run the ETL Algorithm from two different perspectives: first taking in account the overload of ETL process, second without considering it. Figure 5b shows the number of methods calls with and without overloading of ETL operators for each $\mathcal{SDB}$ integrated. It clearly demonstrates that number of invocation method without the overload is much higher comparing to the number of method calls using the overload of ETL operators.

7 Conclusion

In this paper, we deal with the variety of data sources and diversity of deployment platforms when constructing a data warehouse. Thanks a Model Driven Engineering techniques, we make generic all elements of the ETL processes. Meta models are proposed for each element. This genericity contributes in overloading all ETL operators in order to reduce their development costs (prototyping) and consequently their performance. Examples of instantiation of three major classes of databases (relational, semantic and graph) are given. Our efforts of genericity facilitates the multi-store deployment. Finally, an evaluation of our proposal to study the effect of overloading on the performance of different operators is also given. A tool available at Youtube is also given. Currently, we are working the scalability of our proposal using considering a large set of dynamic data sources.

References

1. Akkaoui, Z., Mazón, J.-N., Vaisman, A., Zimányi, E.: BPMN-based conceptual modeling of ETL processes. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2012. LNCS, vol. 7448, pp. 1–14. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32584-7_1
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
3. Berkani, N., Bellatreche, L., Khouri, S.: Towards a conceptualization of ETL and physical storage of semantic data warehouses as a service. Cluster Comput. **16**(4), 915–931 (2013)
4. Calvanese, D., Lenzerini, M., Nardi, D.: Description logics for conceptual data modeling. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems, pp. 229–263. Springer, Boston (1998). doi:10.1007/978-1-4615-5643-5_8
5. Casati, F., Castellanos, M., Dayal, U., Salazar, N.: A generic solution for warehousing business process data. In: VLDB, pp. 1128–1137 (2007)
6. Craig, I.: The Interpretation of Object-Oriented Programming Languages. Springer, London (2002). doi:10.1007/978-1-4471-0199-4
7. Dong, X.L., Srivastava, D.: Big data integration. PVLDB **6**(11), 118 (2013)
8. Mazón, J.-N., Trujillo, J.: An MDA approach for the development of data warehouses. In: JISBD, p. 208 (2009)
9. Jean, S., Bellatreche, L., Ordonez, C., Fokou, G., Baron, M.: OntoDBench: interactively benchmarking ontology storage in a database. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) ER 2013. LNCS, vol. 8217, pp. 499–503. Springer, Heidelberg (2013). doi:10.1007/978-3-642-41924-9_44
10. Kolev, B., Valduriez, P., Bondiombouy, C., Jiménez-Peris, R., Pau, R., Pereira, J.: CloudMdsQL: querying heterogeneous cloud data stores with a common language. Distrib. Parallel Databases **34**(4), 463–503 (2016)
11. Lenzerini, M.: Data integration: a theoretical perspective. In: ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 233–246 (2002)
12. Luján-Mora, S., Vassiliadis, P., Trujillo, J.: Data mapping diagrams for data warehouse design with UML. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 191–204. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30464-7_16
13. Nakuçi, E., Theodorou, V., Jovanovic, P., Abelló, A.: Bijoux: data generator for evaluating ETL process quality. In: ACM DOLAP, pp. 23–32 (2014)
14. Nebot, V., Berlanga, R.: Building data warehouses with semantic web data. Decis. Support Syst. **52**(4), 853–868 (2012)
15. Raventós, R., Olivé, A.: An object-oriented operation-based approach to translation between MOF metaschemas. Data Knowl. Eng. **67**(3), 444–462 (2008)
16. Rodriguez, M.A., Neubauer, P.: Constructions from dots and lines. CoRR, abs/1006.2361 (2010)
17. Shmueli, O., Tsur, S.: Logical diagnosis of LDL programs. New Gener. Comput. **9**(3/4), 277–304 (1991)
18. Simitsis, A., Vassiliadis, P., Sellis, T.-K.: Optimizing ETL processes in data warehouses. In: ICDE, pp. 564–575 (2005)
19. Simitsis, A., Wilkinson, K., Dayal, U., Castellanos, M.: Optimizing ETL workflows for fault-tolerance. In: ICDE, pp. 385–396 (2010)

20. Skoutas, D., Simitsis, A.: Ontology-based conceptual design of ETL processes for both structured and semi-structured data. Int. J. Semant. Web Inf. Syst. **3**(4), 1–24 (2007)
21. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: WWW, pp. 697–706 (2007)
22. Trujillo, J., Luján-Mora, S.: A UML based approach for modeling ETL processes in data warehouses. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 307–320. Springer, Heidelberg (2003). doi:10.1007/978-3-540-39648-2_25
23. Tziovara, P., Vassiliadis, P., Simitsis, A.: Deciding the physical implementation of ETL workflows. In: DOLAP, pp. 49–56 (2007)
24. Vassiliadis, P.: A survey of extract-transform-load technology. IJDWM **5**(3), 1–27 (2009)
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. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Conceptual modeling for ETL processes. In: DOLAP, pp. 14–21 (2002)
27. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Modeling ETL activities as graphs. In: DMDW, pp. 52–61 (2002)
28. Wilkinson, K., Simitsis, A., Castellanos, M., Dayal, U.: Leveraging business process models for ETL design. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 15–30. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16373-9_2