Ontology Based Data Integration Over Document and Column Family Oriented NOSQL stores

Olivier Curé[1], Myriam Lamolle[2], Chan Le Duc[2]

[1] Université Paris-Est, LIGM, Marne-la-Vallée, France
ocure@univ-mlv.fr
[2] LIASD Université Paris 8 - IUT de Montreuil
{myriam.lamolle, chan.leduc}@iut.univ-paris8.fr

Abstract. The World Wide Web infrastructure together with its more than 2 billion users enables to store information at a rate that has never been achieved before. This is mainly due to the will of storing almost all end-user interactions performed on some web applications. In order to reply to scalability and availability constraints, many web companies involved in this process recently started to design their own data management systems. Many of them are referred to as NOSQL databases, standing for 'Not only SQL'. With their wide adoption emerges new needs and data integration is one of them. In this paper, we consider that an ontology-based representation of the information stored in a set of NOSQL sources is highly needed. The main motivation of this approach is the ability to reason on elements of the ontology and to retrieve information in an efficient and distributed manner. Our contributions are the following: (1) we analyze a set of schemaless NOSQL databases to generate local ontologies, (2) we generate a global ontology based on the discovery of correspondences between the local ontologies and finally (3) we propose a query translation solution from SPARQL to query languages of the sources. We are currently implementing our data integration solution on two popular NOSQL databases: MongoDB as a document database and Cassandra as a column family store.

1 Introduction

The distributed architecture of the World Wide Web and its more than 2 billion users, in 2011, enables to store vast amount of information from end-user interactions. The volumes of data retrieved this way are so large that it motivated the design and implementation of new data models and management systems able to tackle issues such as scalability, high availability and partition tolerance. In fact, the Web helped us to understand that the until now prevalent relational model does not fit all the data management issues [24].

These new data stores are regrouped under the NOSQL label (but coSQL [17] is another recently proposed name). This acronym stands for 'Not Only SQL' and generally identifies data stores based on the Distributed Hash Table (DHT)

model which provides a hash table access semantics. That is in order to access and modify a data object, a client is required to provide the key for this object and a management system will lookup the object using an equality match to the required attribute key. The First successful NOSQL databases were developed by Web companies like Google (with Big Table [6]) and Amazon (Dynamo [9]). An important number of open source projects followed more or less inspired by these two systems, e.g. MongoDB[3], Cassandra[4] which respectively correspond to the document and column family categories. Nowadays, NOSQL systems are used in all kinds of application domains (e.g. social networks, science, finance) and are present in cloud computing environments. Hence, we consider that the Web of Data can not miss the opportunity to address and integrate technologies and datasets emerging from this ecosystem.

In this paper, we propose a data integration framework where the target schema is represented as a semantic web ontology and the sources correspond to NOSQL databases. The main difficulty in integrating these data sources concerns their schemalessness and lack of a common declarative query language.

Concerning the schemalessness, although this provides for a form of flexibility in term of data modeling, this makes the generation of correspondences between a global and local schemata more involved. Thus a first contribution of our work consists in generating a local schema for each integrated source using an inductive approach. This approach uses non-standard description logic (DL [2]) reasoning services like Most Specific Concept (MSC) and Least Concept Subsumer (LCS) in order to generate a concept for a group of similar individuals and to define hierarchies for these concepts. Our second contribution enables the specification of a global ontology based on the local ontologies generated for each data source. This global ontology results from the correspondences discovered between concept definitions present in each local ontology.

Concerning the lack of a common declarative query language, we propose a Bridge Query Language (BQL) that supports a translation from SPARQL queries expressed over the global ontology to the possibly different query languages accepted at the sources. In general, document and column family databases do not provide for a declarative query language like SQL. They rather propose a procedural approach based on the use of specific APIs, for instance for the Java language. Hence our last contribution is to present the main steps involved in this transformation and to provide a sketch of the BQL language.

This paper is organized as follows. In Section 2, we present related works in ontology based data integration. Section 3 provides some background knowledge on NOSQL databases, non-standard DL reasoning services and some alignment methods. Section 4 details our contributions in the design of our ontology-based data integration system and thus provides for an overview of this system's architecture. In Section 5, we present the query processing solution adopted for our system. Section 6 concludes the paper and gives perspectives on future works.

[3] http://www.mongodb.org/
[4] http://cassandra.apache.org/

2 Related work

To the best of our knowledge, this paper is a first attempt to integrate data stored in NOSQL systems into an ontology based framework. Hence, in this section, we focus on the broader subject of ontology-based data integration and concentrate on solutions addressing the relational model. Most of the work dedicated to bridging the gap between ontologies and relational databases concentrated on defining mapping languages, query answering and its relationship with reasoning over the ontology.

MASTRO [5] is the reference implementation for the Ontology-Based Data Access (OBDA) approach. In OBDA, ontologies, expressed in Description Logics, represent the conceptual layer of the data stored in relational databases. It allows for both sound and complete conjunctive query answering over an ontology by retrieving data from a relational databases. Most of the nice properties of MASTRO come from the computational characteristics of DL-Lite which motivated the creation of OWL2QL, an OWL2 fragment. Nevertheless, MASTRO requires that both the global ontology and relational schemata are known in order to define semantic mappings.

In [10], the SHER system is presented as a system for scalable conjunctive query answering over $\mathcal{SHIQ}$ ontologies where the ABox is stored in a relational database management system. A main contribution of this work is to implement an ABox *summarization* technique which improves the computational performances of query answering.

In the Maponto tool [1], the authors propose a solution that enables to define complex mappings from simple correspondences. This approach expects end-users or an external software to provide mappings and then uses them to generate new ones. Maponto is being provided with a set of relational databases and an existing ontology.

Systems like MARSON [14] and RONTO [19] discover simple mappings by classifying the relations of a database schema and validate the mapping consistency that have been generated. Like Maponto, these systems require that the target ontology is provided.

In comparison with these systems, our approach deals with the absence of a schema at the sources and of global ontology. Moreover, while all systems based on a relational model benefit from the availability of SQL, the existence of a common query language for the sources can not be assumed in the context of NOSQL databases.

3 Background

In this section, we present background knowledge concerning the two NOSQL databases we are focusing on in this paper, namely document and column-oriented stores. This is motivated by their ability to provide an efficient solution to the scalability issue by enabling to scale out quickly. For both of these

approaches, we model a similar use case dealing with the submission and reviewing process of scientific conferences. Concerning ontology related operations, we present non-standard reasoning services encountered in DL, i.e. MSC, LCS and GCS, and provide information on methods used to align expressive ontologies.

3.1 Document oriented databases

Document oriented databases correspond to an extension of the well-known key-value concept where in this case the value consists of a structured document. A document contains hierarchically organized data similar to XML and JSON. This permits to represent one-to-one as well as one-to-many relationships in a single document. Therefore a complex document can be retrieved or stored without using joins. Since document oriented databases are aware of stored data, it enables to define document field indexes as well as to propose advanced query features. The most popular document oriented databases are MongoDB (10gen) and CouchDB (Apache).

Example 1 This document database (denoted `docDB`) stores data in 2 collections, namely `Person` and `Document`. In the `Person` collection, documents are identified by the email address of the person and contains information regarding the last name, first name, url, university, person type (i.e. either a user, author, conference member or reviewer) and possibly a list of reviewed document identifiers. The documents in the `Document` collection are identified by a 'doc' prefix followed by a unique numerical value. For each document, the system stores the title, email of the different authors (corresponding to keys in the `Person` collection), the abstract and full content of the paper. Finally, a list of reviews is stored for each document. Fig. 1 presents a graphical representation of a document for each collection. In this database, the reviews of a paper are stored within the paper document. This is easily structured in a document store which generally supports the nesting of documents. Similarly, the documents a person needs to review are stored in `Person` documents, i.e. in `writeReview`.

3.2 Column-family databases

Column family stores correspond to persistent, sparse, distributed multilevel hash maps. In column family stores, arbitrary keys (rows) are applied to arbitrary key value pairs (columns). These columns can be extended with further arbitrary key value pairs. Afterwards, these key value pair lists can be organized into column families and keyspaces. Finally, column-family stores can appear in a very similar shape to relational databases on the surface. The most popular systems are HBase and Cassandra. All of them are influenced by Google's Bigtable.

Example 2 Considering the kind of queries one can ask on this column family (denoted `colDB`), the structure consists of 3 columns families: `Person`, `Paper` and `Review`. The set of information stored in these column families is the

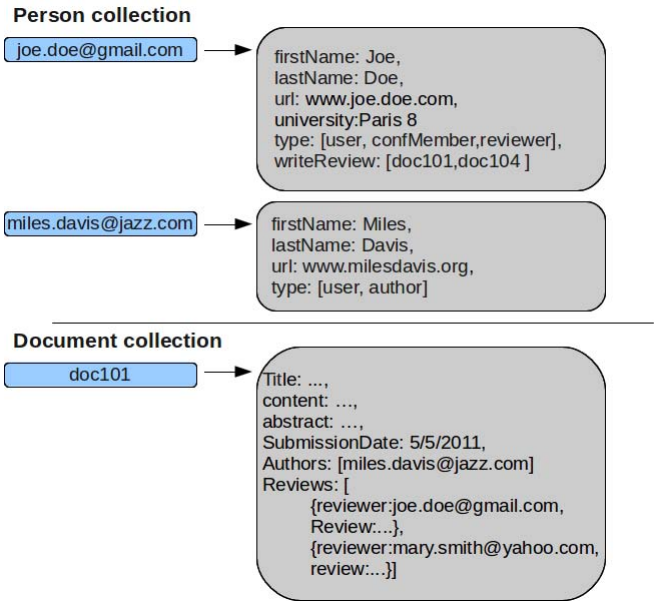

Fig. 1. Extract of the document oriented database

same as in Example 1. The row key for the `Person`, `Paper` and `Review` are respectively the email address of the person and system generated identifiers for papers and reviews. All other information entries are stored in columns with some of them being multi-valued. Fig. 2 provides a graphical representation of an extract of `colDB`. The `Paper` and `Review` column families have several columns in common (`abstract`, `content` and `submissionDate`). But while the `authors` column stores the list of authors of a paper, the `author` column of reviewer stores the identifier of its reviewer.

3.3 Non-standard Reasoning Services

We present in this section the basic versions of *Least Common Subsumer* (LCS) [18] , *Most Specific Concept* (MSC) [2] and *Good Common Subsumer* (GCS) [4]. The *MSC* of an individual consists in defining the least concept description that the individual is an instance of.

Definition 1 Given concept terms $C_1, ..., C_n$, the *MSC* of an individual a is a concept term C iff
- $C \sqsubseteq C_i$, for $1 \leqslant i \leqslant n$;
- C is the most specific concept term with this property, i.e., if D is a concept term such that $C_i \sqsubseteq D$ for $1 \leqslant i \leqslant n$, then $C \sqsubseteq D$.

And, the *LCS* of a set of concepts is the least concept that subsumes all of them, i.e., there is no sub-concept of this *LCS* that subsumes the set of concepts too.

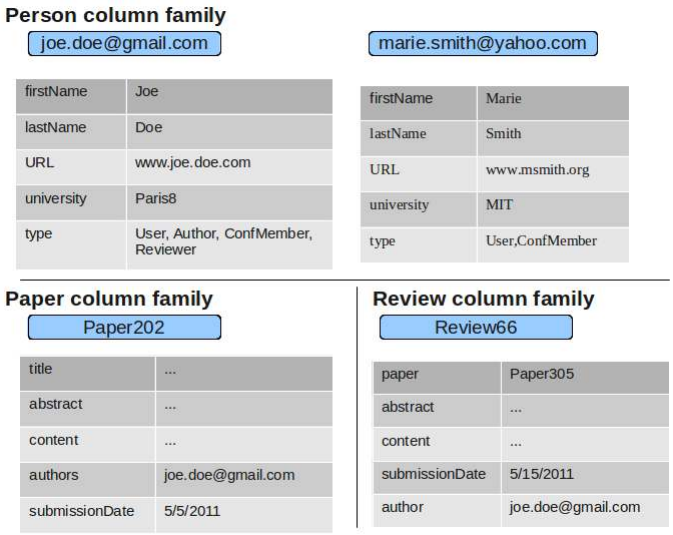

Fig. 2. Extract of the column family database

Definition 2 Given concept terms $C_1, ..., C_n$, the LCS of $C_1, ..., C_n$ is a concept term C such that

- $C_i \sqsubseteq C$ for $1 \leqslant i \leqslant n$;
- C is the least concept term with this property, i.e., if D is a concept term such that $C_i \sqsubseteq D$ for $1 \leqslant i \leqslant n$, then $C \sqsubseteq D$.

But, the *LCS* is very hard to process in practice. So, Baader [4] proposes an algorithm named *Good Common Subsumer (GCS)* to compute an approximation of *LCS* by determining the smallest conjunction of (negated) concept names subsuming the conjunction of the top level concept names of each considered concept. By computing the *MSC* and *LCS* of these individuals, more complex concept descriptions can be added to the ontology [3].

3.4 Alignment methods

The heterogeneity between ontologies must be reduced in order to facilitate interoperability of applications based on these ontologies. For this purpose, semantic correspondences between different entities belonging to two different ontologies are required to be established. This is the goal of ontology alignment as presented in [12]. An alignment consists of a set of correspondences between pairs of ontology entities. Two entities of each pair are connected by a semantic relation (e.g. equivalence, subsumption, incompatibility, etc.). Moreover, a similarity measure can be associated to each correspondence to specify its trust. Then, a set of correspondences (i.e. alignment) can be used to merge ontologies, migrate data or translate queries from one to another ontology.

In the literature, there are several alignment methods that can be categorized according to techniques employed to produce alignments. The most early methods are based on the comparison of linguistic expressions [11]. Another aligner presented in [8] has taken into account annotations of entities defined in ontologies. More recently, the methods introduced in [15], [22], [16] have exploited ontological structures related to concepts in question. These methods, namely simple alignment methods, are the most prevalent at present. They detect simple correspondences between atomic entities (or simple concepts) (e.g. $Human \sqsubseteq Person, Female \sqsubseteq Person$). As a result, some kinds of semantic heterogeneity in different ontologies can be solved by using these classical alignment methods.

However, simple correspondences are not sufficient to express relationships that represent correspondences between complex concepts since (i) it may be difficult to discover simple correspondences (or they do not exist) in certain cases, or (ii) simple correspondences do not allow for expressing accurately relationships between entities.

A second important issue is that generating a complex alignment has a certain impact during the consistency checking of the system. Indeed, a reasoner such as Pellet [23] or FaCT++ [25], running on a system consisting of two ontologies O_1 and O_2 and a simple alignment A_s, may reply that the system is not consistent. But, this same reasoner, with the same ontologies O_1 and O_2, and with a complex alignment A_c can deduce that the system is consistent.

Consequently, new works follow the way of complex alignment solutions such as [20]. But, currently, they address the alignment of simple concept with a complex concept, at best.

4 Architecture overview

In this section, we present the main components of our system and highlight on the approaches used at each steps of the data integration processing. These steps, depicted in Fig. 3, consist of the (1) creation of an ontology associated to each data sources, (2) aligning these ontologies and (3) creating a global ontology given these correspondences.

Finally, we present a query language enabling to retrieve information stored in the sources from a query expressed over the global ontology.

4.1 Source ontology generation

As explained earlier, NOSQL databases are generally schemaless. Although this provides flexibility for information storage, it makes the generation of associated ontologies more involved. In fact, one can only use containers, i.e. collections and column families in respectively document and column family databases, of key/value pairs as well as key labels to deduce a schema. Our approach considers that each container defines a DL concept and that each key label corresponds to a DL property that can either be a data type or object one and whose domain is the DL concept corresponding to its container.

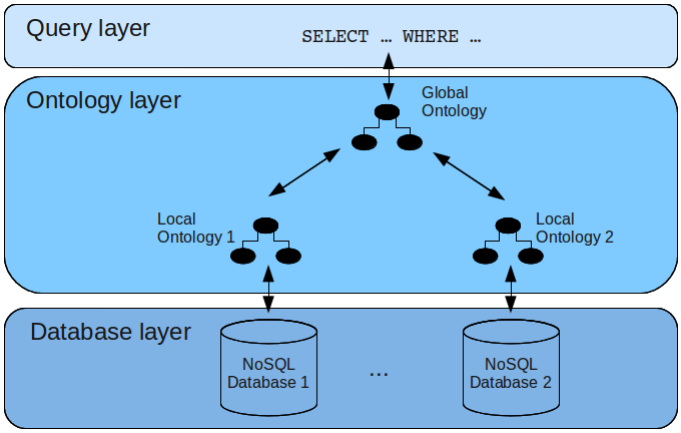

Fig. 3. Basic architecture of our data integration system

Example 3 Consider Example 1 (resp. Example 2), the following concepts are automatically generated: `Person` and `Document` (resp. `Person`, `Paper` and `Review`). Concerning DL properties, a `firstName` DL datatype property will be created in the cases of both Examples 1 and 2 with a domain corresponding to the `Person` DL concept. Additionally, a `writeReview` DL object property is created with a domain and range corresponding to respectively the `Person` DL and `Paper` concepts. This is due to the fact that the values of the `writeReview` (doc101 and doc 104 in the case of the document identified by joe.doe@gmail.com) served as identifier of other documents. The same approach applies for `authors`, `author` and `paper` in Example 2.

A modeling pattern frequently encountered in key/value stores supports the discovery of complementary DL concepts and some subsumption relationships. This pattern, henceforth denoted *type definition*, consists of a key whose range of values is finite and which do not correspond to container identifiers, i.e. they do not serve as foreign keys. We assume that each of these values specifies a DL concept. For instance, this is the case of the `type` key in respectively Examples 1 and 2. Its set of possible values is {User, Author, ConfMember and Reviewer}, each of them corresponding to a DL concept. These concepts can be organized into a hierarchy of concepts using methods of Formal Concept Analysis (FCA) [13]. In a recent paper [7], we have emphasized on an FCA methodology for ontology mediation. Some features of this method are to create concepts that are not in the source ontologies, to label the new concepts, and to optimize the resulting ontology by eliminating redundant or irrelevant concepts. This approach easily applies to the discover of DL concepts and their subsumption relationships in the context of a *type definition* pattern. That is, tuples of the key of the pattern (`type` in our example) correspond to objects in the FCA terminology and their values provide FCA attributes. Then a Galois connection lattice can easily be computed using the methods proposed in [7]. The nodes of this lat-

tice correspond to DL concepts and arrows between them specify subsumption relationships.

Example 4 We consider the document database of Fig.1. The document identified by key 'joe.doe@gmail.com' has several `type` values (User, ConfMember, Author and Reviewer) while the document identified by 'miles.davis@jazz.com' is only characterized by the User value. Using the information coming from different documents, one can discover the following DL concept subsumptions: $Author \sqsubseteq User$
$Reviewer \sqsubseteq User$
$ConfMember \sqsubseteq User$

The method we have presented so far can be applied recursively to embedded structures where the nested container is reified into an object.

At this point in the local ontology generation process, we have created an ontology that is no more expressive than RDFS. We consider that using induction over the instances of the source database, we can enrich the ontology and leverage its expressiveness to a fragment of OWL2, namely OWL2EL. This is performed using the approach proposed in [21] to compute the GCS wrt to local ontology computed. This method exploits the TBox of the ontology and precomputes the conjunction of concept names using FCA. One issue in this precomputation is to handle a possibly very large set of FCA objects.

Ganter's *attribute exploration* interactive algorithm [13] is an efficient approach for computing an appropriate representation of a concept lattice that at certain stages asks contextualized questions to a domain expert. Instead of relying on this interactive process, we propose other solutions that may be used to select a subset of relevant objects. In some cases, the set of objects may be of a reasonable size, (e.g. fitting into main memory) and a complete analysis is possible. Nevertheless, in many situations, due to the size of individual data, a complete analysis is not realistic and some heuristics need to be proposed. The first naive approach one can think of is to randomly access a set of individuals. Apart from the hazardous results this approach could provide, it is not just doable in hash table context where the key of the container needs to be known a priori.

A simple heuristic consists in considering that the most frequently accessed individuals are the most representative of the ontology to generate. In order to discover this set, one can take advantage of the data store architecture, generally distributed over several servers and supervised by several tools such as load balancers. Using logs generated by these tools enables to identify a subset of the individuals that are the most frequently accessed in the application.

Finally an incremental schema generation approach can be implemented. That is each time a tuple is inserted or modified, the system checks if some labels are being introduced or deleted into the schema. This approach imposes that each update operation goes through this process.

At the end of this step, using an inductive approach, we have created a schema for each NOSQL source. The goal of this schema is twofold: it enables

the creation of DL ontology which can be serialized into an OWL2 fragment (namely OWL2EL) and supports the definition between ontology entities (i.e. DL concepts and properties) with elements of the NOSQL source (i.e. documents, column families, columns and keys). Hence, the arrows linking the database and ontology layers of Fig. 3 have been generated. The task of the next section is to generate a global ontology via the discovery of alignments between local ontologies.

4.2 Discovering Alignments between ontologies and global ontology building

We now propose a new solution to detect both simple and complex correspondences. To do this, we follow several steps. The first step consists in enriching the two ontologies to be aligned using the IDDL reasoner [26]. This reasoner allows to add subsumption relations which are implicit in ontologies (see Fig.4).

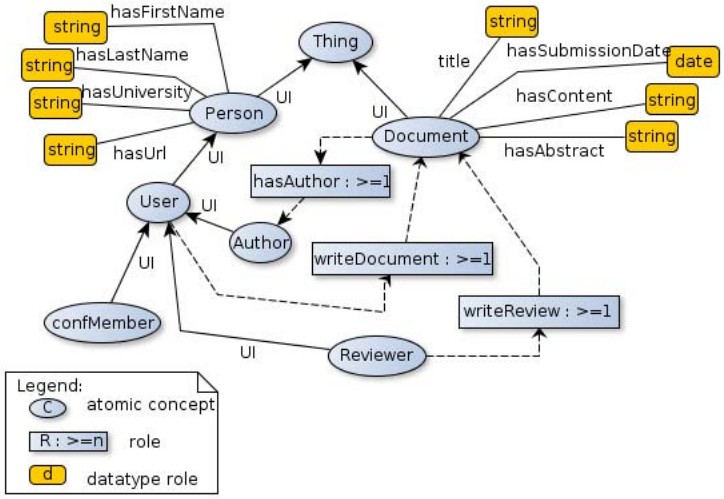

Fig. 4. Extract of review document family DB graph

The second step detects the simple correspondences using three classical alignment processes. We use the three conventional aligners OLA[5] [15], AROMA[6][8] et WN[7]. Each of them is based on a particular approach. The first aligner is a basic aligner, which uses the linguistic resource *WordNet*, the second is based on a structural approach and the third on the annotations associated to entities.

[5] OWL-Lite Alignment
[6] Association Rule Ontology Matching Approach
[7] basic aligner of API alignment named JWNLAlignment

Note that the last two chosen aligners are considered by OAEI[8] among the best alignment systems.

The last step detects the complex correspondences. Our idea is inspired from simple alignment methods which are based on graphs [15],[22]. Since, from a finite vocabulary, an infinite number of formulas can be processed, it is impossible to know what are the relevant formulas to align. A possible solution is to try to capture the semantic of OWL and to represent the constructors (for example, subsumption, disjunction, restriction of cardinality) by a graph formalism. Then, from the two graphs representing the ontologies to be aligned, it must search relevant subgraphs which can be aligned taking into account their structures and using a terminological similarity measure.

Proposition 1 Our first proposition allows correspondences between two complex concepts (i.e. formulas) to be detected.

For example, $Paper \sqcap \exists write.Author \sqcap \exists accept.Reviewer \sqsubseteq Paper \sqcap \exists write.Author$ can be deduced. The detection of this kind of matching is made by exploiting the structure of graphs, which expresses the semantics of the OWL-DL ontologies. Each graph consists of a set of subgraphs, which represent a formula. So, for all pairs of concepts (C_1, C_2) belonging to the ontologies (O_1, O_2), it is necessary to check whether their respective subgraphs can be aligned.

A subgraph of a given concept C consists of all concepts directly linked to C by simple edges (subsumptions or disjunctions) or properties.

To align two subgraphs, one of the following cases must be checked:

1. The first subgraph SG_1 subsumes the second subgraph SG_2 (i.e. $SG_1 \sqsupseteq SG_2$). In this case, a relation of subsumption is generated;
2. The second subgraph SG_2 subsumes the first subgraph SG_1 (i.e. $SG_1 \sqsubseteq SG_2$). In this case, a relation of subsumption is generated, in the opposite way of case 1;
3. The two subgraphs are equivalent. In other words, SG_1 subsumes SG_2 and SG_2 subsumes SG_1. In this case, a relation of equivalence is generated (i.e. $SG_1 \equiv SG_2$).

A subgraph SG_1 subsumes a subgraph SG_2 if the following conditions hold:

- All direct subclasses of SG_1 are similar to direct subclasses of SG_2,
- All disjoint subclasses of SG_1 are similar to disjoint subclasses of SG_2,
- All direct super classes of SG_1 or their generalization are similar to direct super classes of SG_2 or their generalization,
- All direct properties of SG_1 are similar to direct properties of SG_2,
- All properties cardinalities of SG_1 are equivalent or subsumed by properties cardinalities of SG_2,
- All domains or co-domains of these properties of SG_1 are similar to domains or co-domains or their generalizations in SG_2.

[8] Ontology Alignment Evaluation Initiative

Proposition 2 This proposition allow us to detect correspondences between a simple concept and a formula (e.g. $SubmittedPaper \sqsubseteq \exists submit.Author$). It is inspired by the research work presented in [20] with some simplification and generalization. We search correspondences between simple concepts and formulas based on syntactic similarities between concepts and properties. It is necessary to use a purely syntactic similarity measure to compare concepts labels to properties labels. Moreover, a concept to align with a formula having a property similar syntactically must be a concept specializing a concept already aligned to a concept source or target of this property (or one of its super concept).

To generate the complex correspondences detected by these two propositions, we used the language EDOAL[9], which extends the alignment format proposed by INRIA. This language can express complex structures between entities of different ontologies.

Example .

Given two ontologies O_1 and O_2 built from NOSQL databases to be aligned concerning a conference domain. OWL semantics of these ontologies are represented by graphs (cf. Fig. 4 of O_1 from example 1) .

The following simple correspondences are detected during the first step:

$O_1 : Document \equiv O_2 : Document$
$O_1 : Person \equiv O_2 : Person$
$O_1 : Reviewer \equiv O_2 : Referee$
$O_1 : Review \equiv O_2 : Review$
$O_1 : Conference \equiv O_2 : Conference$
$O_1 : Submit \equiv O_2 : Submit$
$O_1 : WriteReview \equiv O_2 : WriteReview$
$O_1 : ConfMember \equiv O_2 : ConfMember$

The second step consisting in traversing the relevant subgraphs detects complex correspondences such as these presented in Fig. 5 and 6. To do this, the neighborhood of the graphs nodes are considered. For example, the generated correspondence from the two subgraphs in Fig. 5 having respectively the nodes $O_1 : Paper$ and $O_2 : Published$ as starting point is:

$O_1 : Paper \sqcap \geqslant 1\ O_1 : hasAuthor.O_1 : contactPerson \sqcap \geqslant 1\ O_1 : Submit.O_1 : contactPerson \sqsupseteq O_2 : Published \sqcap \geqslant 1\ O_2 : Submit \geqslant .O_2 : Author \sqcap \geqslant O_2 : isAuthorOf.O_2 : Author \sqcap O_2 : AcceptedBy.O_2 : ComitteMember$

In the same way, the generated correspondence from the two subgraphs of Fig. 6 having respectively the nodes O_1 *: Reviewer* and O_2 *: Referee* as starting point is:

$O_1 : Reviewer \sqcap \geqslant 1\ O_1 : WriteReview.O_1 : Review \sqsubseteq O_2 : Referee \sqcap \exists\ WriteReview.O_2 : Review$

[9] http ://alignapi.gforge.inria.fr/edoal.html

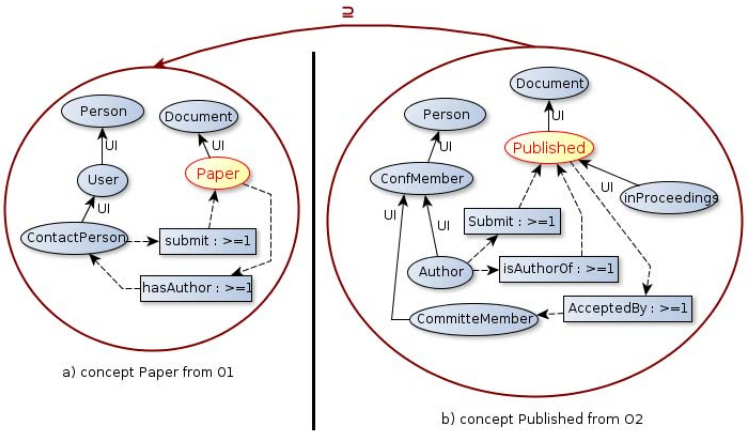

Fig. 5. Aligned subgraphs of concepts $O_1 : Paper$ and $O_2 : Published$

Taking into account the ontologies representing NOSQL data sources and their alignments, a global ontology GO is built. $GO = (O, A)$ represents networked ontologies $0 = \{0_1, ..., O_n\}$ through a set of alignments $A = \{A_1, ..., A_m\}$ where A_i is the set of correspondences between 0_k and $0_l (k \neq l)$.

5 Query processing

In this section, we present the query processing solution adopted in our ontology-based data integration system. The approach consists in two consecutive translation operations.

The first one transforms end-user written SPARQL queries expressed over the global ontology into a set of queries specified in the Bridge Query Language (BQL). This translation uses the correspondences discovered during the local and global ontology generation steps and occurrences of a set of RDF/S properties (e.g. `rdf:type`, `rdfs:subClassOf`) in SPARQL queries. Given these correspondences, a BQL query is generated over the local ontology of a NoSQL source. Due to space limitations, we do not provide a thorough presentation of BQL but rather sketch its main features. BQL is a high-level declarative query language and has low-level, procedural programming flavor that enables to retrieve information from data repositories. In fact, a BQL program is similar to specifying a query execution plan that can easily be translated into fully procedural programs satisfying a given API and programming language. Following a nested data model, a BQL program specifies a sequence of steps that each define a single high level data operation. Like a relational algebra, each step is specified via a relation definition which can serve as the input to another step. A main construct of BQL is a `foreach .. in` operation which permits to iterate other a defined relation and perform some associated operations. These operations

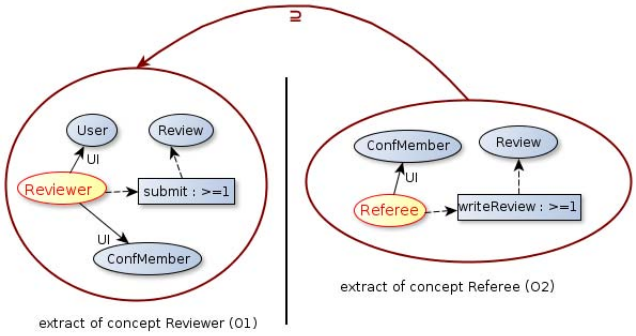

Fig. 6. Aligned subgraphs of concepts $O_1 : Reviewer$ and $O_2 : Referee$

generally consist in retrieving information from the database. This is specified using a `get` operation defined over a given database and container. It contains 2 parameters: a set of filters expressed over source keys with standard comparator (e.g. =, <, <=, ≠) and a set of attributes to retrieve from the resultset. In Example 5, we highlight a SPARQL to BQL transformation given our conference document database.

Example 5 Consider a query that retrieves titles of reviews written by a person with last name Doe. This corresponds to the following SPARQL query which is simplified for readability reasons:
`SELECT ?t WHERE {?p rdf:type Person. ?p hasLastName 'Doe'. ?p writeReview ?r. ?r hasTitle ?title.}`
The presence of a `rdf:type` property in the SPARQL query provides some information about which source database and container we can create a BQL query for. The query specifies that the ?p variable must of type `Person` which is mapped to the `person` container of the document NoSQL database. This query addresses a list of reviews hence an iteration needs to be performed over the `writeReview` attribute of the `Person` container. This first step of the BQL query is written as the following:
`temp(paper) = docDB.Person.get({lastName='Doe'},{writeReview})`
Intuitively, the `temp` relation stores the list of review identifiers written by the person whose last name is 'Doe'. The final result of the query is provided by the `ans` relation:
`ans(title) = foreach paper in temp : docDB.Paper.get({Key=paper}, {title})`. That is for each identifier in the `temp` relation, find documents in the `Paper` collection of the docDB database and retrieve its title.

The second translation corresponds to generating a program in a given programming language (e.g. Java) from the different relations of a BQL query. Given the procedural flavor of BQL, this translation is relatively straight forward but one set of rules needs to be defined for each language and each NoSQL database.

So far, we have implemented rules for the Java language for both the MongoDB and Cassandra stores. In the future, we aim to define such rules for more NoSQL stores and programming languages (e.g. Python, Ruby).

6 Conclusion

This paper tackles the problem of integrating data stores in two of the most popular NOSQL database categories, i.e. document and column family oriented stores, in a Semantic Web context. It is well recognized that scalability is a main issue for these systems. The most involved aspect of this integration concerns the fact that these databases are schemaless and generally lack a common declarative query language. Addressing this first issue, we emphasized that using existing techniques like FCA together with non-standard DL inferences like GCS, we could compute an ontology from the structure and instances of each databases source. Using a novel alignment ontology method, we highlighted that these ontologies can be linked to create a global ontology over which SPARQL queries are expressed. Finally, a bridge query language supports a translation approach to generate procedural queries, using specific APIs for each database source, from SPARQL queries. We have already implemented this translation for the Java language for both the MongoDB and Cassandra NOSQL databases and we are currently working on query optimisation. Recently, several propositions for a common NOSQL declarative query language are emerging (e.g. CQL for Cassandra, unQL for CouchDB). Studying these specifications is on our list of future works. Nevertheless, we consider that our data integration framework is not complete until we incorporate another category of NOSQL stores: graph databases.

References

1. Y. An, A. Borgida, and J. Mylopoulos. Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In *OTM Conferences (2)*, pages 1152–1169, 2005.
2. F. Baader, D. Calvanese, D. L. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, Applications.* Cambridge University Press, Cambridge, UK, 2003.
3. F. Baader, R. Ksters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. pages 96–101. Morgan Kaufmann, 1999.
4. F. Baader, B. Sertkaya, and A. yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. In *Journal of Applied Logic*, pages 400–412. Springer, 2004.
5. D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The mastro system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.
6. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data (awarded best paper!). In *OSDI*, pages 205–218, 2006.

7. O. Curé and R. Jeansoulin. An fca-based solution for ontology mediation. *JCSE*, 3(2):90–108, 2009.
8. J. David, F. Guillet, and H. Briand. Association rule ontology matching approach. *International Journal Semantic Web Information Systems*, 2:27–49, 2007.
9. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, pages 205–220, 2007.
10. J. Dolby, A. Fokoue, A. Kalyanpur, E. Schonberg, and K. Srinivas. Scalable highly expressive reasoner (sher). *J. Web Sem.*, 7(4):357–361, 2009.
11. J. Euzenat. An api for ontology alignment. *In 3rd conference on international semantic web conference (ISWC)*, pages 698–712, 2004.
12. J. Euzenat and S. Pavel. *Ontology matching*. Springer Verlag, Heidelberg (DE), 2007.
13. B. Ganter and R. Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.
14. W. Hu and Y. Qu. Discovering simple mappings between relational database schemas and ontologies. In *ISWC*, pages 225–238, 2007.
15. J.-F. Kengue Djoufak, J. Euzenat, and P. Valtchev. Alignement d'ontologies dirigé par la structure. In Y. A. Ameur, editor, *Conférence Francophones sur les Architectures Logicielles, CAL 2008*, volume RNTI-L-2 of *RNTI*, pages 155–. Cépaduès-Éditions, 2008.
16. T. Lê Bach. *Construction d'un Web sémantique multi-points de vue*. Thèse de doctorat, École des Mines de Paris à Sophia-Antipolis, 2006.
17. E. Meijer and G. M. Bierman. A co-relational model of data for large shared data banks. *Commun. ACM*, 54(4):49–58, 2011.
18. R. Möller, V. Haarslev, and B. Neumann. Semantics-based information retrieval. In *Int. Conf. on Information Technology and Knowledge Systems*, pages 48–61, 1998.
19. P. Papapanagiotou, P. Katsiouli, V. Tsetsos, C. Anagnostopoulos, and S. Hadjiefthymiades. Ronto: Relational to ontology schema matching. *AIS SIGSEMIS Bulletin*, 3(4):32–36, 2006.
20. D. Ritze, C. Meilicke, O. Šváb Zamazal, and H. Stuckenschmidt. A pattern-based ontology matching approach for detecting complex correspondences. *Proceedings of the ISWC 2009 Workshop on Ontology Matching*, 2009.
21. B. Sertkaya. A survey on how description logic ontologies benefit from formal concept analysis. *CoRR*, abs/1107.2822, 2011.
22. P. Shvaiko, J. Euzenat, F. Giunchiglia, and B. He, editors. *SODA: an OWL-DL based ontology matching system*, volume 304 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
23. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: a pratical owl-dl reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.
24. M. Stonebraker and U. Çetintemel. "one size fits all": An idea whose time has come and gone (abstract). In *ICDE*, pages 2–11, 2005.
25. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.
26. A. Zimmermann and C. Le Duc. Reasoning with a network of aligned ontologies. *In Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems (ICWRRS)*, pages 43–57, 2008.