Ontologies as a Solution for Simultaneously Integrating and Reconciliating Data Sources

Abdelghani Bakhtouchi*, Ladjel Bellatreche†, Stéphane Jean† and Yamine Ait-Ameur‡
*National High School for Computer Science (ESI), Algiers, Algeria
bakhtouchi@esi.dz
†LISI/ENSMA and University of Poitiers, France
bellatreche@ensma.fr, jean@ensma.fr
‡IRIT/ ENSHEEIHT, Toulouse, France
yamine@enseeiht.fr

***Abstract*—With the increasing needs for the world wide enterprises to integrate, share and visualize data from various heterogeneous, autonomous and distributed sources data and Web data covering a given domain, the development of integration and reconciliation solutions becomes a challenging issue. The existing studies on data integration and reconciliation of results have been developed in an isolated way and did not consider the strong integration between these two processes. On one hand, ontologies were largely used for building automatic integration systems due to their ability to reduce schematic and semantic heterogeneities that may exist among sources. On the other hand, reconciliation of results is performed either by considering that all sources use the same identifier for an instance or by means of statistical methods that identify affinities between concepts. These reconciliation solutions are not usually suitable for real-world sensitive-applications where exact results are required and where each source may use a different identifier for the same concept. In this paper, we propose a methodology that simultaneously integrate source data and reconciliate their instances based on ontologies enriched with functional dependencies ($\mathcal{FD}$) in a mediation architecture. The presence of $\mathcal{FD}$ gives more autonomy to sources when choosing their primary keys and facilitates the result reconciliation. This methodology is experimented using the Lehigh University Benchmark (LUBM) dataset to show its scalability and the quality of the reconciliation result phase.**

***Index Terms*—Data integration, data reconciliation, ontology.**

I. Introduction

The integration of heterogeneous sources is a well known problem. It has as inputs a set of *distributed*, *heterogeneous*, *autonomous* sources, where each one has its own schema and population and produces as output a unified description of source schemes via an integrated schema that can be used to access data sources. This problem raises two challenging issues: *source heterogeneity* and *data reconciliation*.

Source heterogeneity must be handled to ensure an automatic data integration. Since ontologies are developed in more and more domains to explicit the semantics of a domain, a large number of research studies use them to deal with the semantic problems raised by data integration (e.g., *COIN* [1] or *Picsel* [2]). Thus these approaches deal with sources that contain both data and ontologies that describe its semantics. When these sources use a database as back-end storage, it is called an *Ontology-Based DataBase* ($\mathcal{OBDB}$) (e.g., *Sesame* [3] or *Jena* [4]). Most integration systems do not consider the storage system used to store data and ontologies. However, integration services should also be developed for $\mathcal{OBDB}$.

Data reconciliation consists in correcting the results returned by an integration system. Indeed, when a query is executed on an integration system, results can be redundant since several sources may have the same answer. Throughout the literature, two main trends arise. (1) Some works assume the existence of a common *single identifier* for each concept of the sources participating in the integration process. This assumption relaxes the data reconciliation problem, but it violates sources autonomy, since this common identifier is imposed for various sources. (2) Other research efforts propose to use entity reconciliation methods performed either by *entity matching* or *data fusion* [5], [6]. These approaches have shown their efficiency in linguistic and information retrieval applications [7]. However, they may suffer in the context of sensitive applications such as engineering, banking, healthcare, travel, etc., where exact solutions are required by the end users.

Source heterogeneity and data reconciliation are usually treated separately by existing integration systems. In this paper, we propose a complete integration methodology, called *MIRSOFT*, that handle these issues in a mediator architecture. A preliminary work has been proposed in [8], where we showed that the data reconciliation and integration may be combined in a mediator architecture. This methodology is mainly motivated by a conjunction of three main factors: (1) the conceptual continuity offered by ontologies to generate conceptual models and to ease the resolution of data heterogeneity, (2) the recent definition of functional dependencies ($\mathcal{FD}$) on ontological concepts [9] that can be an interesting solution to the data reconciliation problem and (3) the spectacular development of $\mathcal{OBDB}$ sources that may need to be integrated. To the best of our knowledge, MIRSOFT is the only system that considers simultaneously heterogeneity and data reconciliation problems by integrating $\mathcal{OBDB}$ and reconciliating data through $\mathcal{FD}$ defined on ontologies. Experimentations on the well known LUBM benchmark defined for $\mathcal{OBDB}$ are provided to show the scalability and the quality of the reconciliation result phase of MIRSOFT.

The paper is structured as follows. Section 2 compares main existing data integration systems based on five criteria. Section

3 introduces a formal definition of ontology and $\mathcal{OBDB}$. Section 4 presents $\mathcal{FD}$ for ontologies and shows their interests for the data reconciliation problem. Section 5 describes in details our integration system with its main components. Section 6 presents the experimental studies we have performed to validate our system. Finally section 7 concludes the paper.

II. Related Work

A lot of integration systems have been proposed in the literature. To distinguish clearly our approach from other propositions we propose a classification based on five orthogonal criteria of existing integration systems.

_ *C1: Data representation*. This criterion specifies whether data of local sources are duplicated in a warehouse (noted M) or if they are kept in local sources and then accessed through a mediator (noted V). The ware house approach eliminates several problems of integration, mainly the excessively long server response times or the sources unavailability. However, a main disadvantage of this approach is that answers to queries can frequently be built from outdated data. As a consequence, few systems use this approach (examples are Ajax [10] or Potter's Wheel [11]). Other systems use the virtual architecture, where a software called a mediator supports a virtual database, translates queries into source queries, synthesizes results and returns answers to a user query.

_ *C2: Mapping sense between global and local schemas*. In *Global-as-View* (GaV) systems, the global schema is expressed as a view over data sources. This approach facilitates the query reformulation by reducing it to a simple execution of views in traditional databases. However, changes in source schema or adding a new data source requires a designer to revise the global schema and the mappings between the global schema and source schemas. Examples of integration systems that follow a GaV approach are Multibase [12], Hermes [13] or HumMer [10].

The reverse approach is *Local-as-View* (LaV). In this approach, the designer creates a global schema independently of source schemas. Then, for a new source schema, the designer has only to give a source description that describes source relations as views of the global schema. However, evaluating a query in this approach requires to rewrite it in terms of the data sources and rewriting queries using views is a difficult problem in databases. LaV is used by systems such as Carnot [14] or Picsel2 [2].

Finally, an approach called *Generalized Local-as-View* (GLaV) generalizes both the GaV and the LaV paradigms. In this approach, the designer associates a general query over the source relations to a general query over the global relations. GLaV mappings are more expressive, and are well suited to represent complex relationships in distributed environments. Fusionplex [15] uses this approach.

_ *C3: Mapping automation*. This criterion specifies whether the mapping between the global schema and local schemas is done in a manual (M), semi-automatic (SA), or fully automatic way (A). Manual mappings are found in integration systems that focus mainly on global query processing (e.g., TSIMMIS [16] or Informix [17]). They provide algorithms for identifying relevant sources and decomposing a global query into sub queries for the involved sources but the construction of the mediators and the wrappers must be done manually.

To make the data integration process (partially) automatic, explicit representation of data meaning is necessary. Thus most of the recent integration approaches use ontologies (e.g., Picsel2 [2]). When ontologies and ontology mappings are defined at integration time, the process always request a human supervision and thus they are are only partially automatic. To enable automatic integration, the semantic mapping shall be defined during the database design. Then a shared ontology must exist and moreover, each local source shall contain ontological data that refers to the shared ontology. It means that each local ontology extract a sub-ontology of the shared ontology. Some systems have already been proposed on that direction such as Picsel2 [2], and COIN [1]. But to remain automatic, these systems do not allow individual data source to add new concepts and properties.

_ *C4: Data reconciliation method*. Evaluating a query on an integration system requires first to identify the relevant sources and then to reconciliate the results. Two trends emerge for this last task. Some systems assume that different entities of sources representing the same concept have a global common identifier (CI) (e.g. TSIMMIS [16] and Infomix [17]). This identifier is used for the reconciliation of the results through relational operations. Other systems use statistical methods to identify similar instances (SI) (e.g. HumMer [10] and Ajax [18]). Instances of a query result are compared pair wisely using a similarity measure. A tuple pair is classified based on this measure as *sure-duplicate* or *non-duplicate*.

_ *C5: Data fusion capabilities*. After the data reconciliation phase, conflicts that may appear between results must be handled. Different fusion strategies exist that can be categorized in the following groups [5].

Conflict-Resolution Systems (R) (e.g., Hermes [13] and Fusionplex [15]) perform conflict resolution by implementing deciding and mediating strategies based on metadata or instances.

Conflict-Avoiding Systems (A) (e.g., SIMS [19] and ConQuer [20]) handles data conflicts by conflict avoidance. Conflict are avoided implementing metadata and instance-based strategies.

Conflict-Ignoring Systems (I) (e.g., Pegasus [21] and Nimble [22]) do not perform neither resolution nor avoiding methods. They simply handle conflicts by ignorance.

Table I summarizes our analysis of existing integration systems based on our five criteria. Most of these systems focus either on the integration process or on the data reconciliation and fusion processes. The originality of our approach is to propose a complete system that both handle source heterogeneity through ontologies and reconciliation of query results using $\mathcal{FD}$ on these ontologies.

III. Preliminaries

In this section, we formally defined the concepts needed to understand our proposal.

TABLE I
CLASSIFICATION OF MAIN EXISTING DATA INTEGRATION SYSTEMS

System	C1	C2	C3	C4	C5
Multibase [12]	V	GaV	M	CI	R
Hermes [13]	V	GaV	M	CI	R
Fusionplex [15]	V	GLaV	M	CI	R
HumMer [10]	V	GaV	SA	SI	R
Ajax [18]	M	n/a	M	SI	R
TSIMMIS [16]	V	GaV	M	CI	A
SIMS/Ariadne [19]	V	LaV	M	SI	A
Infomix [17]	V	GaV	M	CI	A
ConQuer [20]	V	n/a	M	CI	A
Pegasus [21]	V	GaV	M	SI	I
Nimble [22]	V	unknown	unknown	SI	I
Carnot [14]	V	Lav	M	CI	I
Picsel2 [2]	V	Lav	A	CI	n/a
Potter's Wheel [11]	M	n/a	n/a	n/a	I
Our approach	V	GaV	M	$\mathcal{FD}$	R

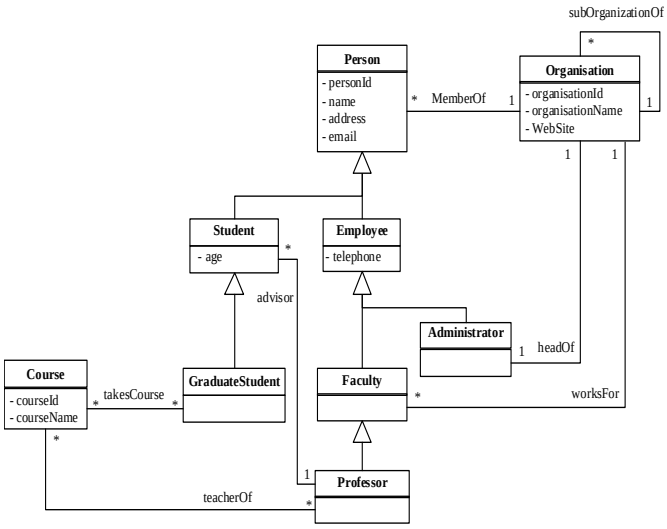

Figure 1. UML diagram representation of a part of LUBM ontology

A. Ontologies

An ontology is usually defined as *an explicit specification of a conceptualization* [23]. Even if different ontology models exist, all ontologies are composed of classes and properties. Thus we consider, in this work, ontologies that may be defined formally as the quadruplet $\mathcal{O} :< \mathcal{C}, \mathcal{P}, \mathcal{S}ub, \mathcal{A}pplic >$ [24]:

- $\mathcal{C}$ is the set of ontology classes.
- $\mathcal{P}$ is the set of properties used to describe the instances of the $\mathcal{C}$ classes.
- $\mathcal{S}ub$ is the subsumption relationship defined as Sub: $\mathcal{C} \rightarrow 2^{\mathcal{C}}$. It associates to a given class its direct subclasses.
- $\mathcal{A}pplic$ is a function defined as $\mathcal{A}pplic : \mathcal{C} \rightarrow 2^{\mathcal{P}}$. It associates to each class the properties that are applicable for each instance of this class and that may be used to describe its instances.

Example. Let's consider the subset of the ontology presented in Figure 1 composed of the *Course*, *Student* and *GraduateStudent* classes. It can be presented as $O < C, P, Sub, Applic >$ with:

$\mathcal{C} = \{Student, GraduateStudent, Course\}$
$\mathcal{P} = \{personId, name, address, email, age, takesCourse, courseId, courseName\}$
$\mathcal{S}ub(Student) = \{GraduateStudent\}$
$\mathcal{S}ub(GraduateStudent) = \phi$
$\mathcal{S}ub(Course) = \phi$
$\mathcal{A}pplic(Student) = \{personId, name, address, email, age\}$
$\mathcal{A}pplic(GraduateStudent) =$
$\{personId, name, address, email, age, takesCourse\}$
$\mathcal{A}pplic(Course) = \{courseId, courseName\}$

B. Ontology-Based DataBases ($\mathcal{OBDB}$)

An $\mathcal{OBDB}$ is a database that store ontologies and its instances. Thus an $\mathcal{OBDB}$ must be defined by an ontology, its instances and the database schema of its instances. As a consequence we consider that an $\mathcal{OBDB}$ is a quadruplet $< \mathcal{O}, \mathcal{I}, \mathcal{S}ch, \mathcal{P}op >$, where:

- $\mathcal{O}$ is an ontology $\mathcal{O} :< \mathcal{C}, \mathcal{P}, \mathcal{S}ub, \mathcal{A}pplic >$;
- $\mathcal{I}$ is the set of instances of the database;
- $Pop : \mathcal{C} \rightarrow 2^{\mathcal{I}}$ associates to each class its instances;
- $\mathcal{S}ch$: $C \rightarrow 2^{P}$ associates to each ontology class c of $\mathcal{C}$ the properties, which are really used to describe the instances of the class c. For each class c, $\mathcal{S}ch(c)$ must satisfy: $\mathcal{S}ch(c) \subseteq \mathcal{A}pplic(c)$.

Our proposition is also based on the notion of functional dependencies presented in next section.

IV. FUNCTIONAL DEPENDENCIES FOR ONTOLOGIES

Traditional ontology formalisms do not support $\mathcal{FD}$ in their definition. Yet, $\mathcal{FD}$ have been used heavily in databases (e.g., normalization theory or query rewriting). As a consequence, a couple of studies have been recently conducted on $\mathcal{FD}$ in the context of ontologies [25], [26], [9], [27]. Inspired by these works we propose the following formal definition for $\mathcal{FD}$.

A. Formal Model for $\mathcal{FD}$

A $\mathcal{FD}$ is composed of the following elements:

1) a *left part* (LP) representing a set of properties,
2) a *right part* (RP) representing a sole property,
3) a *root class* (R). This class is the domain class of the properties of the left part and the right part.

This definition can also be expressed as an implication: $fd\ R : LP \rightarrow RP$.

Among the different functional dependencies proposed in the literature we have identified three types that are helpful for data reconciliation : *classic*, *key* and *basic*.

1) *Classic* $\mathcal{FD}$ ($fd\ R : LP \rightarrow RP$) indicates that values of the properties of the left part determine a unique value of the property of the right part.
   Example. $fd : Person : email \rightarrow name$.
   Each $email$ value of a person determines a unique $name$ value. Formally this $\mathcal{FD}$ is written: $(\{email\}, name) \in \mathcal{FD}\,(Person)$.

2) *Key* $\mathcal{FD}$ ($fd\ R : LP \rightarrow$) indicates that values of the properties of the left part determine a unique instance of the class R. Thus, a *Key* $\mathcal{FD}$ does not contain a right part, but implicitly its right part are the functional properties, denoted by $FP(R)$, of the class.
Definition 1: $(R : LP \rightarrow)$ is a *Key* $\mathcal{FD}$ if and only if LP determines all the *functional properties* $FP(R)$ of the class R. The left parts of all key $\mathcal{FD}$ of a class R, denoted $CK(R)$, are called candidate keys of R.
Example. $fd_k : Person : personId \rightarrow$
Each $personId$ value of a person determines a unique instance of the class $Person$. Formally this $\mathcal{FD}$ is written: $(\{personId\}, \phi) \in \mathcal{FD}\,(Person)$.
3) *Basic* $\mathcal{FD}$ ($fd\ R : \rightarrow RP$) indicates that each instance of R determines a unique instance of the range class of RP. A basic $\mathcal{FD}$ does not contain a left part, implicitly its left part is one of the candidate keys $CK(R)$ of R.
Definition 2: $fd\ R :\rightarrow RP$ is a Basic $\mathcal{FD}$ if and only if RP is a *functional object property* or RP^{-1} is an *inverse functional object property*. In other words RP determines all candidate keys of its range class $CK(\rho(RP))$, where $\rho(p)$ is the range class of the property p.
Example. $fd_b : Person : \rightarrow memberOf$
Each instance of $Person$ determines a unique instance of the class $Organisation$. Formally this $\mathcal{FD}$ is written $(\phi, memberOf) \in \mathcal{FD}\,(Person)$.

B. Extended Formal Model of Ontologies

The formal definition of ontologies (see section III-A) can be extended with $\mathcal{FD}$ as follows:
$\mathcal{O} :< \mathcal{C}, \mathcal{P}, \mathcal{S}ub, \mathcal{A}pplic, \mathcal{FD} >$, where $\mathcal{FD}$ is a binary relationship $\mathcal{FD}$: $\mathcal{C} \rightarrow (2^{\mathcal{P}}, \mathcal{P})$ which associates to each class c of $\mathcal{C}$, the set of the functional dependencies (LP, RP), where the class c is the root ($fd\ c : LP \rightarrow RP$).

Example. The ontology presented in Example III-A can be extended as $O < C, P, Sub, Applic, FD >$ with:
$\mathcal{FD}(Student) = \{(\{personId\}, \phi), (\{email\}, \phi)\}$
$\mathcal{FD}(GraduateStudent) =$
$\{(\{personId\}, \phi), (\{email\}, \phi)\}$
$\mathcal{FD}(Course) = \{(\{courseId\}, \phi)\}$

C. Impact of Functional Dependencies on Data Reconciliation

We claim that $\mathcal{FD}$ can facilitate the data reconciliation problem, especially when no common identifier is used by various sources. To illustrate this point, let's consider the following example.

Example. Let S_1, S_2 and S_3 be three sources containing the same relation $Person$, but with different properties:
$S_1.Person(personId(PK), name, address, email)$,
$S_2.Person(personId(PK), name, email)$ and
$S_3.Person(email(PK), name, address)$.
On this table, the following $\mathcal{FD}$ are defined: $fd_1 : Person : personId \rightarrow name$, $fd_2 : Person : personId \rightarrow address$, $fd_3 : Person : personId \rightarrow email$, $fd_4 : Person : email \rightarrow name$, $fd_5 : Person : email \rightarrow address$.
The mediator schema contains a $Person$ relation with the properties $personId(PK)$, $name$, $address$. Let's consider the following query: *list names and addresses of all persons*. The mediator decomposes this query on the three sources. Without $\mathcal{FD}$, we cannot reconcile all source results, since the source S_3 has a different identifier for $Person$. By using $fd_4 : email \rightarrow name$ and $fd_5 : email \rightarrow address$, we notice that the attribute $email$ is a common candidate key between the three sources. Therefore, a reconciliation of the results coming from these three sources becomes possible using $email$ as a common identifier.

V. Our Integration Methodology

In this section, we first present a formal model of our integration system and the initialization of its components. Then we show how a new source is integrated. Finally, we detail the query processing aspect of our system.

A. Formalization of our data integration system

Formally, our integration system *MIRSOFT* is defined as a triple $\mathcal{M}ed :< \mathcal{G}, \mathcal{S}, \mathcal{M} >$, where:

1) $\mathcal{G} :< \mathcal{O}, \mathcal{S}ch >$ is the global schema.It is composed of the mediator ontology $\mathcal{O} :< \mathcal{C}, \mathcal{P}, \mathcal{A}pplic, \mathcal{S}ub, \mathcal{FD} >$ and the schema $\mathcal{S}ch$ of its ontology classes. $\mathcal{S}ch : \mathcal{C} \rightarrow 2^{\mathcal{P}}$ associates to each mediator ontology class c of $\mathcal{C}$ the properties describing the instances of the class c, which are valuated in at least one integrated source.
2) $\mathcal{S}$ is the set of source schemas, where each source schema is defined as a couple $S_i :< OL_i, SchL_i >$. $OL_i :< CL_i, PL_i, ApplicL_i, SubL_i >$ is the ontology of the source S_i and $SchL_i : CL_i \rightarrow 2^{PL_i}$ is the schema of the OL_i ontology classes.
3) $\mathcal{M}$ is the mapping between the classes of mediator ontology $\mathcal{O}$ and the classes of source ontologies. $\mathcal{M} : \mathcal{C} \rightarrow 2^{\{CL_1 \cup \ldots \cup CL_n\}}$ associates to each mediator ontology class c of $\mathcal{C}$ the corresponding classes of source ontologies.

B. Initialization of the data integration system components

Before starting the integration process, *MIRSOFT* components are initialized. The *mediator ontology* $\mathcal{O}$ is imported by selecting classes and properties from the shared ontology $O_s :< C_s, P_s, Applic_s, Sub_s, FD_s >$. This importation is performed as follows.

1) Starting from user requirements expressed by a set of queries, the mediator administrator selects relevant classes and properties from the shared ontology.
2) The selected classes and properties are added to the mediator ontology;
3) If an imported class has a super class, all its super classes through the class hierarchy are also imported.

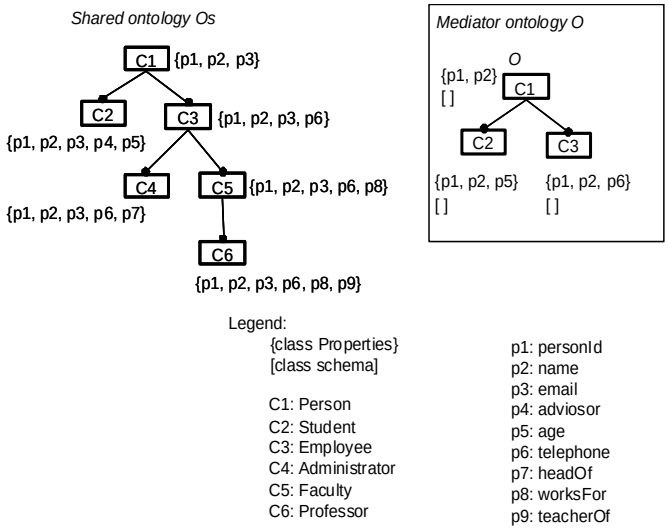

Figure 2. Example of mediator ontology importation

4) To keep the semantic of complex properties (*object properties*), the importation of a complex property involves the importation of its range classes.
5) Likewise, to keep $\mathcal{FD}$, the importation of a property appearing in a right part of a $\mathcal{FD}$ implies the importation of all the properties of the left part of this $\mathcal{FD}$.
6) We check the consistency and we classify the taxonomy of the mediator ontology using a reasoner such as *Racer*, *Pellet*, *Fact++*, etc.
7) New $\mathcal{FD}$ may be added by the administrator in the shared ontology.

Initially, the schema $\mathcal{S}ch$, the source schemas $\mathcal{S}$ and the mapping $\mathcal{M}$ are empty ($\forall\ c \in \mathcal{C}\ \mathcal{S}ch(c) = \phi$, $\mathcal{S} = \phi$ and $\forall\ c \in \mathcal{C}\ \mathcal{M}(c) = \phi$). They are incrementally updated when a new source is integrated.

Example. Figure 2 presents an example of a shared ontology that contains six classes (*Person*, *Student*, *Employee*, *Administrator*, *Faculty* and *Professor*) and nine properties (*personId*, *name*, *email*, *advisor*, *age*, *telephone*, *headOf*, *worksFor* and *teacherOf*).

Suppose that the mediator administrator select only three classes (*Person*, *Student* and *Employee*) and four properties (*personId*, *name*, *age* and *telephone*). After the importation the components of the mediator $\mathcal{M}ed : < \mathcal{G}, \mathcal{S}, \mathcal{M} >$ contain: (1) the mediator ontology $\mathcal{O}$ presented in Figure 2 and (2) the schema $\mathcal{S}ch$, the source schemas $\mathcal{S}$ and the mapping $\mathcal{M}$ which are empty.

C. Integration of a new source

The integration of a new source in our system is based on two assumptions on the mapping between global and local ontologies: (1) each class of a local ontology references explicitly (or implicitly through its parent class) its lowest subsumption class in the shared ontology and (2) only properties that do not exist in the shared ontology may be defined on a local ontology. These assumptions enable two different integration

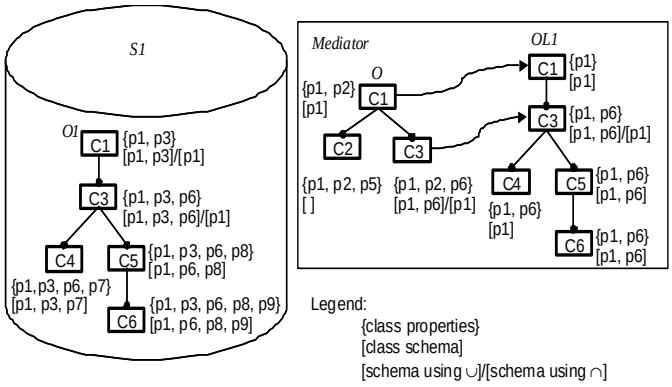

Figure 3. Integrating of a new source in the fragmentation scenario

scenarios associated with automatic integration algorithms: (1) *Fragmentation scenario* where each local ontology of each source is a fragment of the shared ontology and (2) *Integrated ontology scenario*, where each local ontology may be an extension of the shared ontology (to satisfy the autonomy of a local source). This extension is done by adding new classes and properties that do not exist in the shared ontology.

1) Fragmentation scenario: This scenario assumes that the shared ontology is rich enough to cover the needs of all local sources. Such an assumption has been used in the Picsel2 project [2] and in COIN [1]. The source keeps the autonomy to choose (1) the relevant subset (fragment) of the shared ontology (classes and properties) for this source and (2) its local database schema.

In this scenario, the integration of a new source $S_i :< O_i, I_i, Sch_i, Pop_i >$ with $O_i :< C_i, P_i, Applic_i, Sub_i, FD_i >$ is performed using the following steps:

1) the source schemas $\mathcal{S}$ is updated by adding the schema of the new source ($\mathcal{S} = \mathcal{S} \cup \{S_i :< OL_i, SchL_i >\}$);
2) in the ontology $OL_i :< CL_i, PL_i, ApplicL_i, SubL_i >$ only classes and properties existing in the mediator ontology ($OL_i = \mathcal{O} \cap O_i$) are kept;
3) the schemas of the OL_i classes from Sch_i ($\forall\ c \in CL_i\ SchL_i(c) = Sch_i(c)$) are imported;
4) the schema of the mediator ontology classes $\mathcal{S}ch$ is updated by adding the properties valued in S_i to the schema of their classes ($\forall\ c \in CL_i\ \mathcal{S}ch(c) = \mathcal{S}ch(c) \cup SchLi(c)$). This definition means that instances of a query result are expanded with null values to fit with the more precisely defined instances. An alternative definition may also be used to define the schema of a class where instances contain no null values by considering only properties valued in all integrated sources (initially $\mathcal{S}ch(c) = SchL_1(c)$ then $\forall\ i > 1\ \mathcal{S}ch(c) = \mathcal{S}ch(c) \cap SchL_i(c)$);
5) the mapping of the mediator classes is updated by adding the mapping between the classes of the mediator ontology $\mathcal{O}$ and the classes of the new source ontology OL_i ($\forall\ c \in CL_i\ \mathcal{M}(c) = \mathcal{M}(c) \cup S_i.c$).

Example. The integration of a new source in the fragmentation scenario is illustrated Figure 3. At the end of the integration process, the components of the mediator $\mathcal{M}ed : < \mathcal{G}, \mathcal{S}, \mathcal{M} >$ contain:
- No change on the mediator ontology.
- $\mathcal{S}ch(Person) = \{personId\}$, $\mathcal{S}ch(Student) = \phi$ and $\mathcal{S}ch(Employee) = \{personId, telephone\}$ using union operator, $\mathcal{S}ch(Employee) = \{personId\}$ using intersection operator.
- $\mathcal{S} = \{S_1\}$ with OL_i and $SchL_i$ presented in Figure 3.
- $\mathcal{M}(Person) = \{S_1.Person\}$, $\mathcal{M}(Student) = \phi$ and $\mathcal{M}(Employee) = \{S_1.Employee\}$.

2) Integrated ontology scenario: In a number of cases, including e-business applications for instance, sources need more autonomy. Indeed, each local source may have its local ontology composed of specific classes and properties. We support this scenario by requiring that the local ontologies reference the shared ontology O_s while satisfying the two assumptions presented in section V-C. Thus, each source S_i references the shared ontology O_s as follows: $OntoSub_i : C_s \rightarrow 2^{C_i}$ which associates to each class $c \in C_s$ the set of classes $c_i \in C_i$ that are subsumed directly by c. Contrary to the previous case, each data source S_i is defined as a quintuple: $S_i :< O_i, I_i, Sch_i, Pop_i, OntoSub_i >$.

In this scenario, the integration of a new source $S_i :< O_i, I_i, Sch_i, Pop_i, OntoSub_i >$ with $O_i :< C_i, P_i, Applic_i, Sub_i, FD_i >$ is performed as follows.

1) The mediator ontology $\mathcal{O} :< \mathcal{C}, \mathcal{P}, \mathcal{A}pplic, \mathcal{S}ub, \mathcal{FD} >$ is updated by adding the new classes and properties corresponding to the user requirements defined in O_i that does not exist in the shared ontology. To extend the mediator ontology, we use the same algorithm defined for the mediator ontology importation. We denote C_i^+ and P_i^+ the sets of classes and properties which will be added to the mediator ontology. So the component of $\mathcal{O}$ are extended as follow:
$\mathcal{C} = \mathcal{C} \cup C_i^+$,
$\mathcal{P} = P \cup P_i^+$,
$\mathcal{S}ub(c) = \mathcal{S}ub(c) \cup OntoSub_i(c)$,
$\forall\ c \in C_i^+\ \mathcal{A}pplic(c) = \mathcal{A}pplic_i(c)$ and
$\forall\ c \in C_i^+\ \mathcal{FD}(c) = FD_i(c)$.
2) The source schemas $\mathcal{S}$ is updated by adding the schema of the new source ($\mathcal{S} = \mathcal{S} \cup \{S_i :< OL_i, SchL_i >\}$).
3) In the ontology $OL_i :< CL_i, PL_i, ApplicL_i, SubL_i >$ only classes and properties existing in the mediator ontology after extension are kept ($OL_i = \mathcal{O} \cap O_i$).
4) The schemas of the OL_i classes from Sch_i ($\forall\ c \in CL_i\ SchL_i(c) = Sch_i(c)$) are imported.
5) The schema of the mediator ontology classes $\mathcal{S}ch$ is updated by adding the schema of new classes and computing the schema of existing classes. If the class c belongs to C_i^+, $\mathcal{S}ch(c)$ is computed as $\mathcal{S}ch(c) = SchL_i(c)$. If c belongs to $\mathcal{C}$ before extension and it is a leaf class, its schema is explicitly defined as $\mathcal{S}ch(c) = \mathcal{S}ch(c) \cup SchL_i(c)$. Whereas, if c is a non-leaf class, its schema is computed recursively using a post-order tree search by: $\mathcal{S}ch(c) = \cup_{c_j \in Sub(c)} \mathcal{S}ch(c_j)$.
6) The mapping of the mediator classes is updated by adding the mapping of the new classes and updating the mapping of the existing classes ($\forall\ c \in CL_i\ \mathcal{M}(c) = \mathcal{M}(c) \cup S_i.c$).

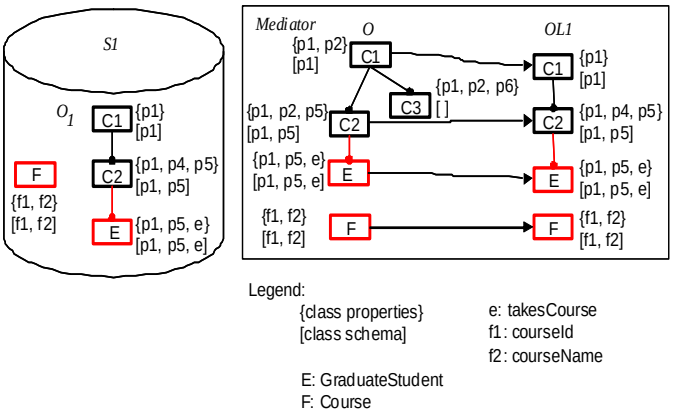

Figure 4. Integrating a new source in the integrated ontology scenario

This scenario leaves a large autonomy to each source and compute in a fully automatic and deterministic way the corresponding integrated system.

Example. Figure 4 illustrates the integration of a new source in the integrated ontology scenario. After this integration the components of the mediator $\mathcal{M}ed : < \mathcal{G}, \mathcal{S}, \mathcal{M} >$ contain:
- The mediator ontology extended with the $GraduateStudent$ and $Course$ classes and their properties.
$\mathcal{C} = \{Person, Student, Employee, GraduateStudent, Course\}$,
$\mathcal{P} = \{personId, name, age, telephone, takesCourse, courseId, courseName\}$,
$\mathcal{S}ub(Student) = \{GraduateStudent\}$
- $\mathcal{S}ch(Person) = \{personId\}$,
$\mathcal{S}ch(Student) = \{personId, age\}$,
$\mathcal{S}ch(GraduateStudent) = \{personId, age, takesCourse\}$,
$\mathcal{S}ch(Course) = \{courseId, courseName\}$ and
$\mathcal{S}ch(Employee) = \phi$.
- $\mathcal{S} = \{S_1\}$ with OL_i and $SchL_i$ presented in Figure 3.
- $\mathcal{M}(Person) = \{S_1.Person\}$,
$\mathcal{M}(Student) = \{S_1.Student\}$,
$\mathcal{M}(GraduateStudent) = \{S_1.GraduateStudent\}$,
$\mathcal{M}(Course) = \{S_1.Course\}$ and
$\mathcal{M}(Employee) = \phi$.

D. Deleting a source from the mediator

A source may be removed for different reasons. For example, the source is physically deleted or unavailable for a long time, the content is judged uninteresting, etc. A source $S_i :< OL_i, SchL_i >$ is deleted with the the following steps:

1) the mapping of the mediator ontology classes is updated by deleting the mapping to the classes of the ontology

OL_i ($\forall\ c \in CL_i\ \mathcal{M}(c) = \mathcal{M}(c) - S_i.c$);

2) the schema of the mediator ontology classes is eventually updated by removing the properties valued only in the source S_i from the schema of the corresponding classes ($\mathcal{S}ch(c) = \mathcal{S}ch(c) - \{p \in \mathcal{S}ch(c) | \forall S_j \in S - \{S_i\} p \notin SchL_j(c)\}$);

3) finally the source from $\mathcal{S}$ is deleted by removing its schema $SchL_i$ and its ontology OL_i.

E. Query answering

In the previous sections we have presented the integration process in our system. In this section we detail the querying part of the system. We consider that queries are expressed as a Union of Conjunctive Queries (UCQ). Thus, each query is given using *datalog* notation as: $Q_i(X)$:- $pr_1(X_1), ..., pr_n(X_n)$, where the predicate pr_i is defined on one of the following ontological concepts:

1) a class $c(x)$ where x is a variable and $c \in \mathcal{C}$;
2) a property $p(x_1, x_2)$ where x_1 and x_2 are variables and $p \in \mathcal{P}$;
3) an ordinary atom $a(x_1, ..., x_m)$ with $(x_1, ..., x_m)$ is a variables vector and a is a predicate.

$X = (x_n, ..., x_n)$ are distinguished variables whereas $x \notin X$ are existential variables serving to express constraints on distinguished variables.

We denote $PP_i = \{p \in \mathcal{P} \mid p(x_1, x_2) \in Q_i \wedge\ x_2 \in X\}$ the projected properties asked by the query Q_i, $Cls_i = \{c \in \mathcal{C} \mid c(x) \in Q_i\}$ the classes appearing in the query Q_i, $JCP_i = \{p \in \mathcal{P} \mid p(x_1, x_2) \in Q \wedge p\ is\ an\ object\ property\}$ the join clause properties of the query Q_i, and $CCP_i = \{p \in \mathcal{P} \mid p(x_1, x_2) \in Q \wedge\ x_2 \notin X\}$ the condition clause properties of the query Q_i.

Example. Let $Q(x, y, z)$ be a query asking for the names of graduate students, the names of courses that they take and the names of their organisations.

$Q(x, y, z)$:- *GraduateStudent* (x_1), *takesCourse*(x_1, x_2), *Course*(x_2), *memberOf*(x_1, x_3), *Organisation*(x_3), *name*(x_1, x), *courseName*(x_2, y), *OrganisationName*(x_3, z).

To answer the query Q_i, the mediator performs the four steps described below.

1) Finding the $\mathcal{FD}$ that hold in the query: Two types of $\mathcal{FD}$ that hold in a query, may be distinguished: (1) *direct* $\mathcal{FD}$ (F^d) already exist in the mediator ontology and (2) *generated* $\mathcal{FD}$ obtained from key $\mathcal{FD}$ (F^k) and from basic $\mathcal{FD}$ (F^b). The $\mathcal{FD}$ that hold in the set of query classes Cls_i, denoted by F_{Cls_i}. They are computed as follow.

- $F_{Cls_i} = F^d \cup F^k \cup F^b$;
- $F^d = fd_1 \cup fd_2 \cup ... \cup fd_n$ are the existing classic $\mathcal{FD}$ on the classes of the query Cls_i where the right part is a projected property from PP_i or a property from the join clauses JCP_i;
- F^k is the set of $\mathcal{FD}$ generated from key $\mathcal{FD}$. F^k indicates that the left part of a key $\mathcal{FD}$ ($R : LP \rightarrow$) determines all the functional properties of its root class $FP(R)$. $F^k = \{LP \rightarrow p \mid p \in PP_i \cup JCP_i \wedge (\delta(p) : LP \rightarrow) \in \mathcal{FD}(\delta(p)) \wedge p \in FP(\delta(p))\}$ where $\delta(p)$ is the domain class of the property p.
- F^b is the set of $\mathcal{FD}$ generated from basic $\mathcal{FD}$ having a property from JCP_i as right part. F^b indicates that the functional property determines all the candidate keys of its range class and the inverse of an inverse functional property determine all the candidate keys of its domain class. $F^b = \{p \rightarrow CK(\rho(p)) \mid p \in JCP_i \wedge (\delta(p) : \rightarrow p) \in \mathcal{FD}(\delta(p))\}$.

Finally, the $\mathcal{FD}$ that hold in Q_i, denoted by F_{Q_i}, are computed by: $F_{Q_i} = \{X \rightarrow Y \mid X \rightarrow Y \in F^+_{Cls_i}\}$

2) Deriving the reconciliation key: Before deriving the reconciliation key, we determine the relevant sources (S_{Q_i}) among sources of S. We keep in S_{Q_i} only the sources in which one of the projected properties PP_i at least is valuated ($S_{Q_i} = \{S_j \in S \mid \exists\ p \in PP_i\ \exists\ c \in CL_j\ p \in SchL_j(c)\}$).

The reconciliation key is derived using the algorithm 1. Its inputs are the query Q_i, the concerned sources S_{Q_i}, the mediator components ($\mathcal{M}ed :< \mathcal{G}, \mathcal{S}, \mathcal{M} >$) and the left parts ($K$) of the functional dependencies that hold in the query F_{Q_i}. The algorithm generates as output a reconciliation key (K_R). We start by a reconciliation key K_R containing all the elements of K. If two left parts in K_R determine the same set of properties of $PP_i \cup JCP_i$, we keep the left part which is valuated in all sources. Then we remove the properties that can be functionally determined by an other left part in K_R.

Example. Consider the query $Q(x, y, z)$ of the example V-E. We have the following fds on the classes $GraduateStudent$, $Course$ and $Organisation$:
- The key $\mathcal{FD}$ $fd_1 : GraduateStudent : personId \rightarrow$,
- The key $\mathcal{FD}$ $fd_2 : GraduateStudent : email \rightarrow$,
- The basic $\mathcal{FD}$ $fd_3 : GraduateStudent : \rightarrow memberOf$,
- The key $\mathcal{FD}$ $fd_4 : Course : courseId \rightarrow$,
- The key $\mathcal{FD}$ $fd_5 : Organisation : organisationId \rightarrow$.

The $\mathcal{FD}$ that hold in $Q(x, y, z)$ are $F_{Q_{(x,y,z)}} = \{X \rightarrow Y \mid X \rightarrow Y \in F^+_{Cls_i}\}$ with: $F_{Cls_i} = F^d \cup F^k \cup F^b$
$F^d = \phi$,
$F^k = \{personId \rightarrow email,\ personId \rightarrow name,\ personId \rightarrow memberOf,\ email \rightarrow personId,\ email \rightarrow name,\ email \rightarrow memberOf,\ courseId \rightarrow courseName,\ organisationId \rightarrow organisationName\}$,
$F^b = \{memberOf \rightarrow organisationId\}$,
$K = \{personId, email, courseId, organisationId, memberOf\}$.

The algorithm 1 removes $organisationId$ and $memberOf$ and derives one of the two reconciliation keys $\{personId, courseId\}$ or $\{email, courseId\}$.

3) Query evaluation: Each query Q_i will be rewritten into the union of sub queries over the concerned sources $Q_i^{S_j}$.

$Q_i = Q_i^{S_1} \cup ... \cup Q_i^{S_r}$ with $S_j \in S_Q$ where $Q_i^{S_j}$ being a query over the ontology $OL_j :< CL_j, PL_j, ApplicL_j, SubL_j >$ of the source S_j having the following form:

$Q_i^{S_j}(X)$:- $pr_1(X_1), ..., pr_n(X_n)$ where the predicate pr_i is:
- $S_j.c(x)$, x is a variable, $S_j.c(x) \in \mathcal{M}(c)$ and $c \in Cls_i$,

Algorithm 1. Deriving the reconciliation key

Input: Q_i: The query;
S_{Q_i}: Concerned sources;
$\mathcal{M}ed :< \mathcal{O}, \mathcal{S}, \mathcal{M} >$;
K: Left parts of $\mathcal{FD}$ in F_{Q_i};

Output: K_R: Reconciliation key;

begin
  $K_R = K$;
  foreach $LP_i \in K_R$ **do**
    // $P_{LP_k} = \{p \in PP_i \cup JCP_i \mid LP_k \rightarrow p\}$: properties determined by LP_k;
    if $\exists\, LP_j \in K_R\ P_{LP_i} = P_{LP_j}$ *and* $\forall\, S_k \in S_{Q_i}\ \forall\, p \in LP_i\ p \in SchL_k(\delta(p))$ **then**
      $K_R = K_R - \{LP_j\}$;
  foreach $LP_i \in K_R$ **do**
    if $\exists\, p \in K_R\ LP_i \rightarrow p$ **then**
      $K_R = K_R - \{p\}$;
end

- $p(x_1, x_2)$, x_1 and x_2 are variables and $p \in (PP_i \cup JCP_i \cup CCP_i) \cap PL_j$,

- $a(x_1, ..., x_m)$ with $\forall\ x_i \in \{x_1, ..., x_m\}\ p(x_j, x_i) \in Q_i^{S_j}$.

All the properties of JCP_i may not be valuated by the source S_j. Therefore the query $Q_i^{S_j}$ may not be valid on this source. For example if the query $Q_i(x)$:-$Person(x, y), memberOf(x, y), Department(y)$ is executed on a source S_j where the property $memberOf$ is not valued, the rewritten query is $Q_i^{S_j}(x, y)$:-$Person(x), Department(y)$ which is not valid because it returns a Cartesian product. As a consequence, the validity of the query is tested and, if it is not valid, only a part of the results are returned (e.g., $Q_i^{S_j}(x)$:- $Person(x)$ for the previous query). This process is achieved by the algorithm 2. It is used to set the projected properties $PP_i^{S_j}$, the classes $Cls_i^{S_j}$, the join clause properties $JCP_i^{S_j}$ and condition clause properties $CCP_i^{S_j}$ of the query $Q_i^{S_j}$. We start by $Cls_i^{S_j}$ containing the class considered by the user as the most important in the query (this class must be a domain class of a property from the reconciliation key). Using the join clause properties valuated in the concerned source, we add to $Cls_i^{S_j}$ all the classes accessible from the initial class. Condition clause properties $CCP_i^{S_j}$ (respectively projected properties $PP_i^{S_j}$) are the properties of CCP_i (respectively PP_i) valuated in S_j and having the domain classes in $Cls_i^{S_j}$.

Finally, the resulting query $Q_i^{S_j}$ is sent to the source S_j to be evaluated.

4) Results reconciliation and fusion: To avoid redundancy and conflicting information, data integration systems implement data reconciliation and fusion techniques. Most of these methods need to query all sources and are designed for offline data aggregation that can take a long processing time. Instead of this approach, we propose an *incremental* and *online* reconciliation and fusion method allowing to return the first answer as soon as possible. The system returns answers from the first concerned source and refreshes the answers as it queries more sources by applying reconciliation and fusion techniques on the retrieved data. For each returned answer, it shows the asked sources. Finally the process can be stopped if the user satisfied by the current answer.

Algorithm 2. Query unfolding

Input: $\mathcal{M}ed :< \mathcal{O}, \mathcal{S}, \mathcal{M} >$;
K_R: Reconciliation key;
Q_i: The query;
S_j: The concerned source;

Output: $Cls_i^{S_j}$: Query classes;
$JCP_i^{S_j}$: Join clause properties;
$CCP_i^{S_j}$: Condition clause properties;
$PP_i^{S_j}$: Projected properties;

begin
  $Cls_i^{S_j} = \{\delta(p) \mid p \in K_R\}$ /* $\delta(p)$ is chosen by the user */;
  $JCP_i^{S_j} = \phi$;
  repeat
    noChange = true;
    if $p \in JCP_i$ *and* $p \in SchL_j(\delta(p))$ *and* $S_j.(\delta(p)) \in \mathcal{M}(\delta(p))$ *and* $\delta(p) \in JCP_i^{S_j}$ *and* $\rho(p) \notin JCP_i^{S_j}$ **then**
      $Cls_i^{S_j} = Cls_i^{S_j} \cup \{\rho(p)\}$;
      $JCP_i^{S_j} = JCP_i^{S_j} \cup \{p\}$;
      noChange = false;
  until *noChange or* $Cls_i^{S_j} = Cls_i$;
  $CCP_i^{S_j} = CCP_i \cap \{p \in PL_j \mid p \in SchL_j(\delta(p)) \wedge \delta(p) \in Cls_i^{S_j}\}$;
  $PP_i^{S_j} = PP_i \cap \{p \in PL_j \mid p \in SchL_j(\delta(p)) \wedge \delta(p) \in Cls_i^{S_j}\}$;
end

The answer of the initial query Q, denoted by $ans(Q)$, are $ans(Q) = ans(Q_1) \cup ... \cup ans(Q_n)$.

The set of instances satisfying the query Q_i from all the concerned sources S_{Q_i} is $ans(Q_i) = ans(Q_i^{S_1}) \cup_{K_R} ... \cup_{K_R} ans(Q_i^{S_r})$ where $\cup_{K_R}$ is the union with reconciling the instances using the reconciliation key K_R.

The set of instances satisfying the query $Q_i^{S_j}$ in the source S_j are tuples from the Cartesian product of the $Cls_i^{S_j}$ class populations that satisfy join clauses and condition clauses of the query $Q_i^{S_j}$. $ans(Q_i^{S_j}) = \{t \in Pop_j(c_1) \times ... \times Pop_j(c_m) \mid t \models Q_i^{S_j}\}$ with $c_1, ..., c_m \in CL_i$.

K_R is the reconciliation key of the query Q_i means that K_R functionally determines all the projected properties of the query ($\forall\ p \in PP_i\ K_R \rightarrow p$) in all sources. In other word $\forall\ p \in PP_i$, $\forall\ i_1 \in ans(Q_i^{S_v})$, $\forall\ i_2 \in ans(Q_i^{S_w})\ i_1[K_R] = i_2[K_R] \Rightarrow i_1[p] = i_2[p]$ where $i_1[K_R] = i_2[K_R] \Leftrightarrow \forall\ p \in K_R\ i_1[p] = i_2[p]$. $i[K_R]$ is the vector of values that take the

properties of K_R in the instance i.

Let $Reconcile$ be a binary predicate. $Reconcile(i_1, i_2)$ means that the two instances, denoted by i_1 and i_2, refer to the same world entity.

For two instances i_1 and i_2, a decision of reconciliation is taken ($Reconcile(i_1, i_2)$) if both instances have the same values for all properties composing the reconciliation key.

$i_1[K_R] = i_2[K_R] \Rightarrow Reconcile(i_1, i_2)$

Similarly, a decision of non-reconciliation is taken ($\neg Reconcile(i_1, i_2)$) if there is a property of the reconciliation key K_R for which the values of the two instances are different.

$i_1[K_R] \neq i_2[K_R] \Rightarrow \neg Reconcile(i_1, i_2)$

where $i_1[K_R] \neq i_2[K_R] \Leftrightarrow \exists\ p \in K_R\ i_1[p] \neq i_2[p]$

So, the reconciliation of the result coming from a source and the global result can be performed by the algorithm 3. This algorithm takes each instances of source result and check if there is an instance in the global result that can be reconciled with this source instance. If such an instance exists, the algorithm fuses the property values of the two instances as a single instance in the global result, otherwise the source instance is added to the global result as a new instance.

Algorithm 3. Reconciliation of a source result

Input: K_R: Reconciliation key;
$ans(Q_i^{S_j})$: The source result;
R: Global result;
Output: R: Global result;
begin
  foreach $i_2 \in ans(Q_i^{S_j})$ **do**
    if $\exists\ i_1 \in R\ Reconcile(i_1, i_2)$ **then**
      $i_1 = FusionOf(i_1, i_2)$;
    else
      Add i_2 to R;
end

Now that we have presented the different part of our system, let's see how it has been implemented.

F. Implementing our methodology

Since our integration system uses ontology and must be scalable, we have implemented it on an $\mathcal{OBDB}$. Most $\mathcal{OBDB}$ do not support the definition of $\mathcal{FD}$ as it is not supported by most ontology model. However, the OntoDB $\mathcal{OBDB}$ [28] supports the extension of its ontology model through its ontology exploitation language OntoQL. This section shows how we have used and extended this $\mathcal{OBDB}$ to implement our approach.

1) Overview of the implemented Architecture: Different modules composing our integration system are described in Figure 5: (1) an $\mathcal{OBDB}$ repository, (2) a user interface, (3) a query engine and (4) a result reconciliator.

- The *$\mathcal{OBDB}$ Repository* is based on the OntoDB architecture. It is composed of four parts illustrated in Figure 6. *Part (1)* and *Part (2)* are traditional parts available in all DBMSs, namely the *data part* that contains instance data and *meta-base* part that contains the system catalog. (3) The *ontology* part allows the representation of ontologies in the database. (4) The *meta-schema* part records the ontology model into a reflexive *meta-model*. For the *ontology part*, the *meta-schema part* plays the same role as the one played by the meta-base in traditional databases. By means of naming convention, the meta-base part also represents the logical model of the content, and its link with the ontology, thus representing implicitly the conceptual model of data in database relations.

  For supporting our approach the meta-schema part is extended by (i) a mediator and source schema model, (ii) a model of mapping between the mediator ontology and source ontologies and (iii) a $\mathcal{FD}$ model. The OntoQL commands used to do this extension are presented in section V-F2. In the ontology part, we store the mediator ontology, source ontologies and schemas, the mapping between the mediator ontology and source ontologies

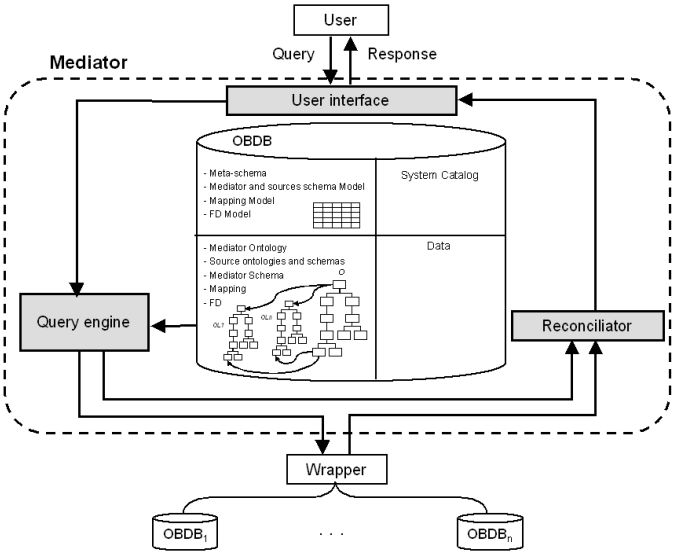

Figure 5. Different Modules Composing our Integration System

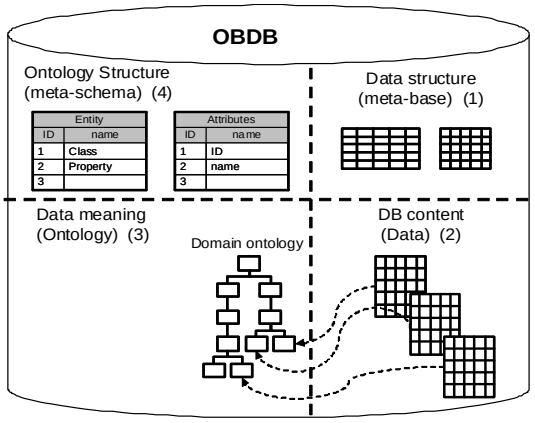

Figure 6. OntoDB architecture

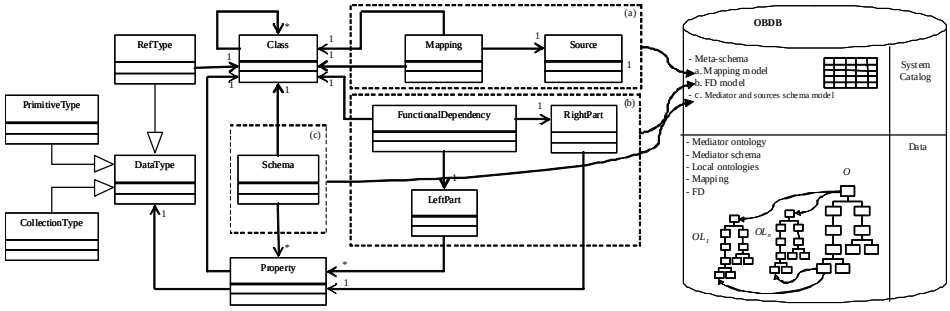

Figure 7. Extension of OntoDB meta-schema

and the $\mathcal{FD}$ between the classes and properties of the mediator ontology. The data part can be used as a caching to optimize frequently queries (in this work this issue is not addressed).

- The *user interface*: it is used by users to express queries and display its results. After parsing the input query, the user interface send to the query engine a conjunctive query defined on a set of classes and properties of the mediator ontology. The user interface is responsible also of displaying answers from the first visited sources and refreshing the answers when the answers of more sources coming from the reconciliator are available.
- The *query engine*: for a given user query Q_i, the query engine performs the following tasks: (1) finding the $\mathcal{FD}$ that hold in the query, (2) identifying then the concerned sources and deriving the reconciliation key and (3) rewrites the query defined on mediator ontology in local queries defined in sources ontologies, where each one is sent to relevant sources. It sends then the reconciliation key to the reconciliator.
- The *result reconciliator*: it reconciles the results using the reconciliation key, to merge instances referring to the same real world entity and to send progressively obtained results to the user interface in an incremental way.

2) Representation of $\mathcal{FD}$ in OntoDB: The meta-schema of OntoDB contains two main tables *Entity* and *Attribute* encoding the meta-model level. *Entity* describes ontological constructors such as class, property or data type. *Attribute* describes attributes related to each ontological constructors (*name*, *description*, *comment* ...). An extension of the meta-schema of OntoDB consists in adding new entities and attributes. Thus we have defined three meta-models describing the $\mathcal{FD}$ concepts, mapping concepts and class schemas concepts (see Figure 7). These three meta-models are instanciated in the meta-schema of OntoDB using OntoQL.

The following statements encode these instantiations.

```
CREATE ENTITY #FD(#ItsClass REF(#Class),
       #ItsLeftProperties REF(#Property)ARRAY,
       #ItsRightProperty REF(#Property));
CREATE ENTITY #Source(#URL STRING, #UserName STRING,
                      #Password STRING);
ALTER ENTITY #Ontology ADD ATTRIBUTE #ItsSource REF(#Source)
CREATE ENTITY #Mapping(#MediatorClass REF(#Class),
       #SourceClass REF(#Class), #ItsSource REF(#Source));
CREATE ENTITY #Schema(#ItsClass REF(#Class),
              #SchemaProperty REF(#Property));
```

VI. Methodology Validation

To validate the feasibility and efficiency of our system, we conduct experiments using dataset of Lehigh University Benchmark (LUBM) and its 14 queries [1]. The used ontology of LUBM has 45 classes and 32 properties (including 25 object properties, and 7 data type properties). Based on this ontology a set of ontology-based databases is generated. All experiments have been carried out on an Intel Pentium IV machine, with 3,2 GHz processor clock frequency, equipped with 1 Gb of RAM, under the operating system Windows XP professional. The OntoDB $\mathcal{OBDB}$ is implemented on PostgreSQL.

Two main experiments are conducted to evaluate (1) the scalability of our system based on the number of sources and instances and (2) the quality of the returned results.

A. Scalability of our approach

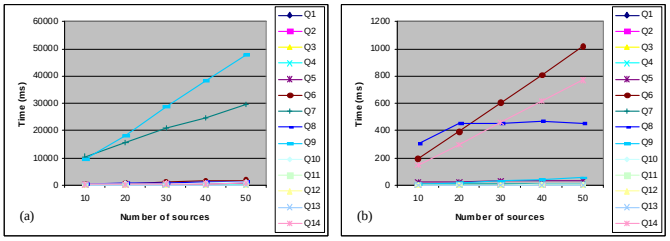

Figure 8. (a) Query Response Time - (b) Reconciliation Time vs. Number of Sources.

Figure 8 (a) shows the query response time (in millisecond) of the 14 queries when the the number of sources participating in the integration process range from 10 to 50 (following a fragmentation scenario). Figure 8 (b) presents the reconciliation time for these same queries. Based on the LUBM Benchmark, generated sources have the same schema (22 relations) and the biggest relation has 2000 tuples.

The obtained results show that our system executes efficiently (less than one second) queries involving a small set of classes (less joins) (e.g. $Q_3(x)$:- *Publication(x), publicationAuthor(x, 'http://www.Department0.University0.edu/AssistantProfessor0')*), but, for queries involving large number of classes

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

(e.g. $Q_9(x)$:- *Student(x), Faculty(y), Course(z), advisor(x, y), takesCourse(x, z), takesCourse(y, z)*), the response time is quite high, but still reasonable (50 seconds for 50 sources). Notice that the execution time comprises the time needed for mediator processing time (including finding the functional dependencies that hold in the query, deriving the reconciliation key and reconciliation of results), and local query evaluation.

Figure 8.(b) points out that, for most queries, the time needed for mediator processing is negligible w.r.t. the overall execution time. Thus, the major time consuming process is the evaluation of the query over the sources. Two queries ($Q_6(x)$:- *Student(x)* and $Q_{14}(x)$:- *UndergraduateStudent(x)*) have a reconciliation time near 1s for 50 sources. The reconciliation time are higher for these queries because they return an important number of results.

The scalability of our approach depends also on the number of instances of each source. Thus, we conduct an other experiment that consists in varying the number of instances of 10 used sources. Figure 9 shows the obtained results. In this experiment, the high costly queries are Q_7 and Q_9. In the previous experiment they were the low costly queries. Indeed these queries involves several join operations. And, when the number of instances increases, these operations become costly. Figure 9 (b) shows similar results as the first experiment. Thus, this experiment shows that the query response time depends heavily on the sources and their ability of processing queries and not on the mediator.

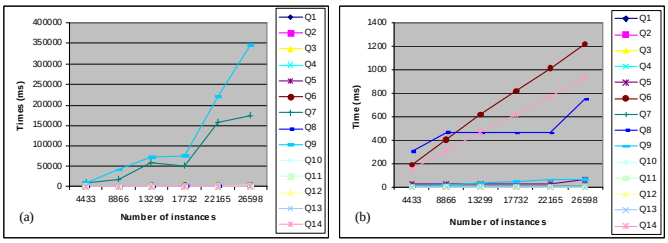

Figure 9. (a) Query Response Time - (b) Reconciliation Time vs. Number of Instances.

B. Quality of results of our approach

One strength of our approach (noted *M*) is that we reconciliate results using $\mathcal{FD}$. These $\mathcal{FD}$ are used to compute a reconciliation key that can be composed of several attributes. A lot of approaches use instead a unique reconciliation key (noted *UK*). This last approach works well if all sources use the same key for each entity. However, we think that this assumption will not hold in many situations. Thus we have chosen to compare the unique reconciliation key with our approach when this assumption does not hold. More precisely, we make the assumption that, for a given concept, a source uses the unique key with a given probability. For conciseness we present only results obtained for a probability of 50%. These results are presented in table II (number of queries results with M and UK for 10, 30 and 50 sources) and illustrated in Figure 10 (percentage of UK results compared to M results). These results do not include results for queries Q1, Q2, Q3, Q4, Q5, Q7, Q10 and Q11 because the same results are returned with the two methods. Indeed, for these queries, the result exists in the same source that uses the unique key. For other queries Q6, Q8, Q12, Q13, Q14 the UK method returns less results than our method. The percentage depends on the number of attributes in the reconciliation key. When this key is only composed of one attribute (Q6, Q8, Q12 and Q14) this percentage is around 50% because 50% of the sources use the unique key. When the number of attributes in the reconciliation key growths (e.g, 3 for Q6) this percentage can be very low because there is less chance than all sources will use the same key for the three attributes. In this experiment we also vary the number of sources. As we can see the percentage is stable for each query except for Q13 because it has very few results so the percentage is not meaningful.

TABLE II
RESULTS OF M COMPARED TO THE RESULTS OF UK

	10		30		50	
	M	UK	M	UK	M	UK
Q6	5239	2689	10407	5058	16139	7896
Q8	5238	2688	7787	4089	7787	4089
Q9	140	27	430	48	716	95
Q12	10	5	15	8	15	8
Q13	1	1	5	2	9	3
Q14	4022	2051	12357	5987	20249	9849

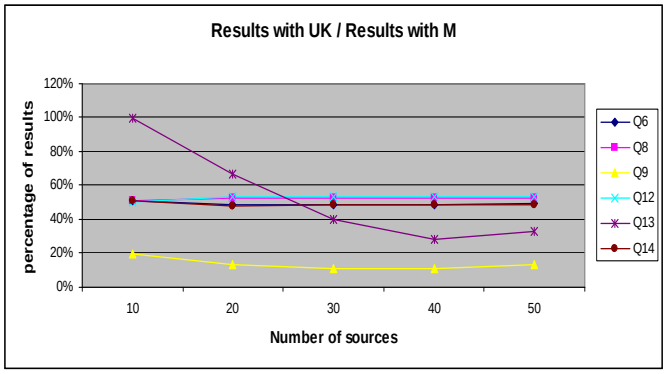

Figure 10. Percentage of results returned by UK compared to M

Finally we run a last experiment to test th data fusion technique that we use to eliminate duplicate results. For this experiment, we use five different sources. We get the number of results with these five sources (that do not have any duplicated). Then we increase the number of sources by duplicating values of one of the five sources. We run this experiment for the 14 queries and 5, 10, 20 , 30, 40 and 50 sources. Table III presents a subset of these results (the other results are similar). It shows the number of results returned by our approach (M) and without using any fusion technique (N). As we can see our method eliminates the duplicate for each number of sources tested.

TABLE III
ELIMINATION OF DUPLICATES

	5		20		50	
	M	N	M	N	M	N
Q5	719	719	719	2876	719	7190
Q6	7790	7790	7790	31160	7790	77900
Q8	7787	7787	7787	31148	7787	77870
Q14	5916	5916	5916	23664	5916	59160

VII. CONCLUSION

The integration of different sources is an important problem for many applications in various domains. Meanwhile a lot of ontologies has been developed in these domains leading to many sources that contain both data and ontologies that describe the meaning of these data. In this paper, we have proposed an integration system called MIRSOFT for these sources. The originality of MIRSOFT is to propose a complete methodology for both integrating and/or deleting sources on the fly and reconciliating results. Note that most of existing integration systems follow two extreme scenarios for data reconciliation. (1) Some systems suppose that the manipulated sources have similar keys to ensure data integration. This usually violates the autonomy of sources. (2) Others use statistical techniques to reconcile data. In sensitive domains, where exact solutions, such techniques cannot be used. Instead, MIRSOFT uses $\mathcal{FD}$ that have been recently defined for ontologies for the reconciliation of results. We show on the Lehigh University Benchmark that if $\mathcal{FD}$ are expressed on ontologies, our approach can eliminate the result duplicates even if the different sources do not use the same key for the different entities. Moreover we show that the execution time of a query on MIRSOFT depends on the time spent by the sources to process the query and not on the mediator.

We are currently extending MIRSOFT in two ways. Firstly, we are considering other types of sources in the integration process (e.g., XML sources). Secondly, we are running more experiments on real datasets to evaluate the complexity of defining $\mathcal{FD}$ on real ontologies. We hope that this study will lead us to design a methodology for defining in a semi-automatic way such $\mathcal{FD}$.

REFERENCES

[1] C. Goh, S. Bressan, E. Madnick, and M. D. Siegel, "Context interchange: New features and formalisms for the intelligent integration of information," *ACM Transactions on Information Systems*, vol. 17, no. 3, pp. 270–293, 1999.

[2] C. Reynaud and G. Giraldo, "An application of the mediator approach to services over the web," in *Special track "Data Integration in Engineering, Concurrent Engineering (CE'2003)*, July 2003, pp. 209–216.

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

[4] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international World Wide Web conference (Alternate Track Papers & Posters)*, 2004, pp. 74–83.

[5] J. Bleiholder and F. Naumann, "Data fusion," *ACM Computing Surveys*, vol. 411, no. 1, pp. 1–41, 2008.

[6] F. Saïs, N. Pernelle, and M. C. Rousset, "Combining a logical and a numerical method for data reconciliation," *Journal of Data Semantics (JoDS)*, vol. 12, pp. 66–94, 2009.

[7] X. L. Dong and F. Naumann, "Data fusion - resolving data conflicts for integration," *PVLDB*, vol. 2, no. 2, pp. 1654–1655, 2009.

[8] A. Bakhtouchi, L. Bellatreche, and Y. Ait-Ameur, "Ontologies and functional dependencies for data integration and reconciliation," in *Proceedings of the 1st International Workshop on Modeling and Reasoning for Business Intelligence (MoreBi), held in conjunction with ER 2011*, 2011, pp. 98–107.

[9] O. Romero, D. Calvanese, A. Abello, and M. Rodriguez-Muro, "Discovering functional dependencies for multidimensional design," in *ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP)*, 2009, pp. 1–8.

[10] A. Bilke, J. Bleiholder, F. Naumann, C. Böhm, K. Draba, and M. Weis, "Automatic data fusion with hummer," in *Proceedings of the International Conference on Very Large Databases*, 2005, pp. 1251–1254.

[11] V. Raman and J. M. Hellerstein, "Potter's wheel: An interactive data cleaning system," in *Proceedings of the International Conference on Very Large Databases*, 2001, pp. 381–390.

[12] U. Dayal, "Processing queries over generalization hierarchies in a multidatabase system," in *Proceedings of the International Conference on Very Large Databases*, 1983, pp. 342–353.

[13] S. Adali and R. Emery, "A uniform framework for integrating knowledge in heterogeneous knowledge systems," in *Proceedings of the International Conference on Data Engineering (ICDE)*, 1995, pp. 513–520.

[14] M. P. Singh, P. E. Cannata, M. N. Huhns, N. Jacobs, T. Ksiezyk, K. Ong, A. P. Sheth, C. Tomlinson, and D. Woelk, "The carnot heterogeneous database project: Implemented applications," *Distributed and Parallel Databases Journal*, vol. 5, pp. 207–225, April 1997.

[15] A. Motro, J. Berlin, and P. Anokhin, "Multiplex, fusionplex and autoplex: three generations of information integration," *Sigmod Record*, vol. 33, pp. 51–57, 2004.

[16] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom, "The tsimmis project: Integration of heterogeneous information sources," in *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, 1994, pp. 7–18.

[17] D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Staniszkis, "The infomix system for advanced integration of incomplete and inconsistent data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005, pp. 915–917.

[18] H. Galhardas, D. Florescu, D. Shasha, and E. Simon, "Ajax: an extensible data cleaning tool," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, 2000, p. 590.

[19] Y. Arens and C. A. Knoblock, "Sims: Retrieving and integrating information from multiple sources," in *SIGMOD*, May 1993, pp. 562–563.

[20] A. Fuxman, D. Fuxman, and R. J. Miller, "Conquer: A system for efficient querying over inconsistent databases," in *Proceedings of the International Conference on Very Large Databases*, 2005, pp. 1354–1357.

[21] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. A. Ketabchi, W. A. Litwin, A. Rafii, and M.-C. Shan, "The pegasus heterogeneous multi-database system," *IEEE Computer*, vol. 24, no. 12, pp. 19–27, 1991.

[22] D. Draper, A. Y. Halevy, and D. S. Weld, "The nimble integration engine," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001, pp. 567–568.

[23] T. Gruber, "A translation approach to portable ontology specification," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[24] G. Pierra, "Context-explication in conceptual ontologies: The plib approach," in *Proceedings of 10th ISPE International Conference on Concurrent Engineering: Research and Applications (ce'03) : Special Track on Data Integration in Engineering*, July 2003, pp. 243–254.

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

[26] D. Calvanese, G. Giacomo, and M. Lenzerini, "Identification constraints and functional dependencies in description logics," in *Proceedings of Joint Conference on Artificial Intelligence (IJCAI)*, 2001, pp. 155–160.

[27] D. Toman and G. E. Weddell, "On keys and functional dependencies as first-class citizens in description logics," *Journal of Automated Reasoning*, vol. 40(2-3), pp. 117–132, 2008.

[28] H. Dehainsala, G. Pierra, and L. Bellatreche, "Ontodb: An ontology-based database for data intensive applications," in *in 12th International Conference on Database Systems for Advanced Applications (DASFAA)*, 2007, pp. 497–508.