MIRSOFT: mediator for integrating and reconciling sources using ontological functional dependencies

Abdelghani Bakhtouchi

Ecole nationale Supérieure d'Informatique (ESI),
Algiers, Algeria
E-mail: a_bakhtouchi@esi.dz

Ladjel Bellatreche* and Stéphane Jean

Laboratoire d'Informatique et d'Automatique pour les Systèmes (LIAS)/Ecole Nationale Supérieure de Mécanique et d'Aérotechnique (ISAE-ENSMA),
Futuroscope, 86960 France
E-mail: bellatreche@ensma.fr
E-mail: jean@ensma.fr
*Corresponding author

Yamine Ait-Ameur

Institut de Recherche en Informatique de Toulouse (IRIT)/
Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et des Télécommunications (ENSEEIHT), Toulouse, 31000 France
E-mail: yamine@enseeiht.fr

Abstract: Providing automatic integration solutions is the key to the success of applications managing massive amounts of data. Two main problems stand out in the major studies:

i the management of the source *heterogeneity*

ii the *reconciliation* of query results.

To tackle the first problem, formal ontologies are used to explicit the semantic of data. The reconciliation problem consists in deciding whether different identifiers refer to the same instance. Two main trends emerge in the reconciliation process:

i the assumption that different source entities representing the same concept have the same key – a strong hypothesis that violates the autonomy of sources.

ii The use of *statistical* methods that identify affinities between concepts – not suitable for *sensitive-applications*.

In this paper, we propose a methodology integrating sources referencing *shared domain ontology* enriched with *functional dependencies* ($\mathcal{FD}$).

Copyright © 2012 Inderscience Enterprises Ltd.

The presence of $\mathcal{FD}$ gives more autonomy to sources when choosing their primary keys and allows deriving a reconciliation key for a given query. The methodology is then validated using LUBM.

Keywords: data integration; mediation architecture; ontologies; functional dependencies; reconciliation.

Reference to this paper should be made as follows: Bakhtouchi, A., Bellatreche, L., Jean, S. and Yamine, A-A. (2012) 'MIRSOFT: mediator for integrating and reconciling sources using ontological functional dependencies', *Int. J. Web and Grid Services*, Vol. 8, No. 1, pp.72–110.

Biographical notes: Abdelghani Bakhtouchi is PhD candidate in Computer Science at the National High School for Computer Science (ESI) in Algiers, Algeria. He received his Diploma-Master in Computer Science from ESI in 2007 and his Engineer Degree in Computer Science from the Polytechnic School of Algiers (EMP) in 2001. His research interests include Ontology-based data integration and data warehousing.

Ladjel Bellatreche is a Professor at National Engineering School for Mechanics and Aerotechnics (ENSMA), Poitiers, France where he joined as a faculty member since September 2010. He is a member of Data Engineering Team of Laboratory of Applied Computer Science (LISI). Ladjel Bellatreche has been actively involved in the research community by serving as reviewer for technical journals and Editorial Board Member, *International Journal of Reasoning-based Intelligent Systems*, Inderscience, and as an organiser/co-organiser of numerous international and National Conferences and Workshops. In addition, he served as a programme committee member for over 30 international conferences and Workshops. His research interests include data integration systems, data warehousing, ontologies, physical design of VLDB.

Stéphane Jean is an Assistant Professor at the University of Poitiers, France. He is a member of Data Engineering Team of Laboratory of Applied Computer Science (LISI) at the National School of Engineers in Mechanics and Aeronautics (ENSMA). His research interests include ontologies management especially in the context of databases, semantic query languages, data warehousing. He participated to several research and Industrial projects around semantic query languages and has published research papers in international Journals and Conferences.

Yamine Ait-Ameur is Full Professor at IMP-ENSEEIHT, Polytechnique National Intitute in Toulouse, France. His reserach interests are formal modelling with a particular focus on ontology based modelling and proof based formal techniques. He participated to several research projects around formal modelling and has published a couple of research papers in International Journals and Conference Proceedings.

1 Introduction

In past years, Enterprise and Information Integration (EII) became an established business, with academic and commercial tools integrating data and XML sources more readily available. These tools offer final users a uniform and a transparent access to data. The spectacular development of this business has been motivated by the need companies have to be able to access data allocated over the internet and within their intranets (Dong and Naumann, 2009; Halevy et al., 2005; Sarma et al., 2011; Nguyen et al., 2011). Source Integration problem has as inputs: a set of *distributed*, *heterogeneous*, *autonomous* sources, where each one has its schema and population. Its output a unified description of source schemes via an integrated schema and mapping rules allowing the access to data sources. The construction of a data integration system is a difficult task due to the following main factors:

a the large number of data sources candidate for integration

b the lack of semantic sources explicitation

c the heterogeneity of sources

d the autonomy of sources.

(a) *The explosive growth of data sources*: The number of data sources involved in the integration process is increasing. The amount of information generated in the world increases by 30% every year and this rate is bound to accelerate (Dong and Naumann, 2009), especially in domains such as E-commerce, engineering, etc. Integrating these mountains of data requires *automatic solutions*.

(b) *The lack of explicitation of the semantic of sources*: The semantics of data sources is usually not explicit. Most sources participating in the integration process were designed to satisfy *day-to-day* applications and not to be integrated in the future. Often, the small amount of semantic contained in their conceptual models is lost, since only their logical models are implemented and used by applications. The presence of a conceptual model may allow designers to express the application requirements and domain knowledge in an intelligible form for a user. Thus, its absence or any other semantic representation in final databases makes their interpretation and understanding complicated, even for designers who have good knowledge of the application domain.

(c) *The heterogeneity of data sources impacts both the structure and the semantic*: Structural heterogeneity exists because data sources may have different structures and/or different formats to store their data. The autonomy of the sources increases heterogeneity significantly. Indeed, the data sources are designed independently by various designers with different application objectives. Semantic heterogeneity presents a major issue in developing integration systems (Hakimpour and Geppert, 2002). It is due to different interpretations of real world objects, generating several categories of conflicts (naming conflicts, scaling conflicts, confounding conflicts and representation conflicts (Goh et al., 1999)).

(d) *Autonomy of sources*: Most sources involved in the data integration are fully autonomous in choosing their schemes.

To deal with semantic problems and ensure an automatic data integration, a large number of research studies propose the use of ontologies. Several integration systems were proposed under this hypothesis. We can cite for instance: *COIN* (Goh et al., 1999), *Observer* (Mena et al., 1996a, 1996b) *OntoDaWa* (Xuan et al., 2006; Bellatreche et al., 2006), etc. In Bellatreche et al. (2006), we claim that if the semantic of each source participating in the integration process is explicit in *a priori* way, the integration process becomes automatic. This means that the used ontology should exists before the creation of each source participating in the integration process. This assumption is reasonable in several application domains, where ontologies are largely developed: medicine (Unified Medical Language System), engineering (e.g., IEC [ISO13584-42, IEC61360-2]), biology (http://www.iplantcollaborative.org/), businness inteligence applications (Romero and Abelló, 2010), etc. The storage of ontologies in a database leads to the concept of *Ontology-Based DataBases* ($\mathcal{OBDB}$). Several academic and industrial systems offer efficient solutions to store and manage ontologies and their associated data (e.g., *Jena* (Carroll et al., 2004), *Sesame* (Broekstra et al., 2002), *Oracle* (Das et al., 2004), *IBM Sor* (Lu et al., 2007)). If we follow this trend, $\mathcal{OBDB}$ sources become then candidate for the integration process. Therefore, integration services should be developed for this type of sources.

Once source heterogeneity is solved by the use of ontologies, data integration designers have to propose solutions for data reconciliation when queries are answered over the integration system (following mediator architecture). Throughout the literature, two main trends arise:

1 works in the first one assume the existence of a common *single identifier* for each common concept of sources participating in the integration process.

The assumption relaxes the data reconciliation problem, but it violates the source autonomy, since this common identifier is imposed for various sources. Other research efforts characterising the second trend propose the use of entity reconciliation methods performed either by *entity matching* or *data fusion* (Bleiholder and Naumann, 2008; Saïs et al., 2009). These approaches have shown their efficiency in linguistic and information retrieval applications (Dong and Naumann, 2009). 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.

Note that these two issues: *heterogeneity management* and *data reconciliation* are treated separately. In this paper, we propose a complete integration methodology, called, *MIRSOFT* that incorporates these issues in a mediator architecture. It is mainly motivated by a conjunction of three main factors:

- the conceptual continuity offered by ontologies to generate conceptual models (Bellatreche et al., 2011) and to ease the resolution of data heterogeneity
- the recent research efforts that define functional dependencies $\mathcal{FD}$ on ontological concepts (Romero et al., 2009). The presence of $\mathcal{FD}$ allows designers to generate all candidate keys for each class of an ontology. As a consequence, a source designer may pick her or his primary keys from

candidate keys. So $\mathcal{FD}$ give more autonomy to sources and may contribute in reconciling query results.

- The spectacular development of $\mathcal{OBDB}$ sources that may need to be integrated. Note that *MIRSOFT* integrates $\mathcal{OBDB}$ by relaxing the constraint of existing of a common identifier. To our best of knowledge, this work is the first that considers simultaneously heterogeneity and data reconciliation problems.

The paper is structured as follows. Section 2 presents a state of the art related to data integration problem, where a classification of existing systems is proposed. Section 3 presents the concepts and the formal definitions of ontology and $\mathcal{OBDB}$. Section 4 describes an extension of ontology model by $\mathcal{FD}$ and shows their impact on data reconciliation. Section 5 describes in details our integration system with its main components. Section 6 presents experimental studies. Section 7 concludes the paper.

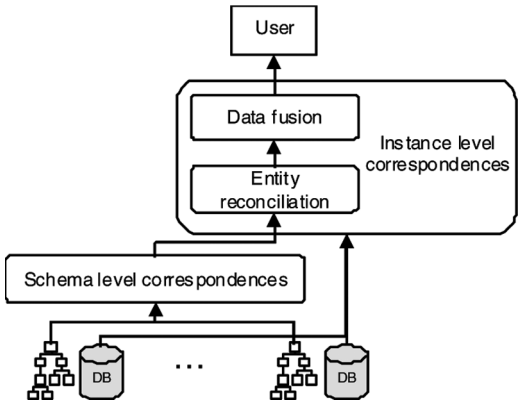
2 State of the art

In this section, we first describe in details the most important studies conducted on data reconciliation and then a classification, based on five orthogonal criteria, of existing integration systems is presented. This classification will help designers and readers to understand each studied system and to analyse it.

2.1 Data reconciliation

Data reconciliation occurs on two levels: *schema* and *instance* correspondences (Figure 1).

Figure 1 Data and schema reconciliation



Schema level correspondences task aims at establishing semantic mappings between contents of disparate data sources. More precisely, this process identifies tables (in the case of relational sources) or XML entities (in the case of XML databases)

(Nguyen et al., 2011) that represent the same real-world entity type and identifies attributes describing the same property for the same entity type. This task is usually performed when creating the global schema of the integration system. This problem of determining schema-level correspondences is referred to as schema matching and inter-schema relationship identification (Zhao and Ram, 2008). Some studies propose the use of formal ontologies to deal with semantic conflicts between sources (Hakimpour and Geppert, 2002; Xuan et al., 2006). The main idea behind these works is to formally specify the meaning of the concepts of each source and to define a translation between each source concepts using ontologies. In other words, each source will have its own local ontology describing its meaning. Consequently, correspondences between sources can be viewed as ontology mappings (Wache et al., 2001). Our work follows this tendency.

The goal of instance level correspondences is to establish reconciliation between instances that represent the same entity in the real world and to produce the correct value of such entity. Two types of instance level correspondences exist: *entity reconciliation* methods (Rahm and Bernstein, 2001; Zhao and Ram, 2008; Köpcke and Rahm, 2010; Batini and Scannapieco, 2010) and *data fusion* methods (Naumann et al., 2006; Liu et al., 2011; Yin et al., 2008; Batini and Scannapieco, 2010).

2.1.1 Entity reconciliation

Entity reconciliation (also referred to as *entity matching*, *duplicate identification*, *record linkage* or *entity resolution*) is a crucial task for data integration and data cleaning. This step aims at resolving heterogeneity at the instance level by identifying entities (objects, data instances) referring to the same real-world entity.

Given two sets of entities $A \in S_A$ and $B \in S_B$ of a particular semantic entity type from data sources S_A and S_B, the data reconciliation problem consists in identifying all correspondences between entities in $A \times B$ representing the same real-world object. This definition includes the special case of finding pairs of equivalent entities within a single source ($A = B, S_A = S_B$). The match result is typically represented either by a set of correspondences, sometimes called a mapping, or by a set of clusters. A correspondence $c = (e_i, e_j, s)$ interrelates two entities e_i and e_j from sources S_A and S_B. An optional similarity value $s \in [0, 1]$ indicates the similarity or strength of the correspondence between the two objects. In the alternate result representation, a cluster contains entities that are deemed to represent the same real-world object. Ideally all entities in a cluster refer to the same object, and two entities from two different clusters do not refer to the same object (Köpcke and Rahm, 2010).

Principally, data reconciliation seems to be a simple process. If a pair is more similar than a given threshold it is declared a duplicate. But it has two main drawbacks: *effectiveness* and *efficiency* (Bleiholder and Naumann, 2008). Effectiveness is mostly affected by the quality of the similarity measure and the choice of a similarity threshold. Usually the similarity measure is domain-specific. The similarity threshold determines when two objects are duplicates. To ensure efficiency, entity reconciliation should be fast even for voluminous datasets. For very large datasets this requirement usually prescribes the use of blocking methods to reduce the search space for entity reconciliation. More precisely, blocking reduces

the search space from the Cartesian product to a small subset of the most likely reconciling entity pairs. Numerous blocking algorithms have been proposed in the past years. These techniques commonly use a key to partition the entities to be matched into groups (blocks). The reconciliation of an entity can then be restricted to the entities in the same block (Köpcke and Rahm, 2010). The key is typically composed from parts of entity attribute values.

The various approaches to entity reconciliation proposed in the literature can be classified into two categories: *rule-based* approaches and *learning-based* approaches (Zhao and Ram, 2008). In rule-based approaches, domain experts are required to directly provide decision rules for matching semantically corresponding records. In learning-based approaches, domain experts are required to provide sample matching (and non-matching) records, based on which classification techniques are used to learn the entity reconciliation rules.

2.1.2 Data fusion

The objective of data fusion step is to combine records that refer to the same real-world entity by merging them into a single representation and resolving possible conflicts from different data sources. Data fusion aims at resolving conflicts from data and increasing correctness of data. There are many different strategies to resolve conflicts adopted by different integration systems.

Techniques that deal with data fusion can be applied in two different phases of the life cycle of a data integration system, namely, *at design time* and at *query time* (Batini and Scannapieco, 2010). In both cases, the actual conflicts occur at query time; however, the design time approaches decide on the strategy to follow for fixing conflicts before queries are processed, i.e., at the design stage of the data integration system. The techniques operating at query time incorporate the specification of the strategy to follow within the query formulation. Design time techniques have a major optimisation problem. Therefore, data fusion at design time may be very inefficient. Query time data fusion techniques have been proposed to overcome such performance inefficiencies. Furthermore, query time techniques are characterised by greater flexibility, they allow those who formulate the query to indicate a specific strategy to adopt for conflict resolution. Given a query, these techniques deal with data fusion by resolving conflicts that may occur on query results.

Another classification is proposed in Bleiholder and Naumann (2008), where existing techniques of resolving data conflicts are classified into three categories. In particular, *Conflict ignoring* strategies are not aware of conflicts, perform no resolution, and thus may produce inconsistent results. *Conflict avoiding* strategies are aware of conflicts but do not perform individual resolution for each conflict. Rather, a single decision is made, e.g., preference of a source, and applied to all conflicts. Finally, *conflict resolving* strategies provide the means for individual fusion decisions for each conflict. Such decisions can be *instance-based*, i.e., they deal with the actual conflicting data values, or they can be *metadata-based*, i.e., they choose values based on metadata, such as freshness of data or the reliability of a source. Finally, strategies can be classified by the result they are able to produce: *deciding* strategies choose a preferred value among the existing values, while *mediating* strategies can produce an entirely new value, such as the average of a set of conflicting numbers.

2.2 Classification of data integration systems

In this section, we present a classification of integration systems. Most of existing classifications ignore the data reconciliation. Initial classifications used only one criterion: the nature of global schema generation (Global as View (GaV) approach (Chawathe et al., 1994) and Local as View (LaV) approach (François Goasdoué et al., 2000; Levy et al., 1996; Reynaud and Giraldo, 2003)). This criterion has been enriched by the type of the architecture materialising the integrated systems (Ullman, 1997). Other contributions consider the number of used ontologies (single ontology, multiple ontologies, and a shared ontology) (Wache et al., 2001). Our classification takes into account data reconciliation, where we distinguish between systems assuming the existence of a global common identifier and systems using statistical methods. We also take into account the data fusion capabilities where we distinguish systems handling conflicts by resolution, systems handling conflicts by avoidance and systems ignoring conflicts. Table 1 summarises most important data integration systems using our five criteria:

Table 1 Classification of existing data integration systems (inspired from (Bleiholder and Naumann, 2008))

System	*C1*	*C2*	*C3*	*C4*	*C5*
Multibase (Dayal, 1983)	V	GaV	M	CI	R
Hermes (Adali and Emery, 1995)	V	GaV	M	CI	R
Fusionplex (Motro et al., 2004)	V	GLaV	M	CI	R
HumMer (Bilke et al., 2005)	V	GaV	SA	ER	R
Ajax (Bilke et al., 2005)	M	n/a	M	ER	R
TSIMMIS (Chawathe et al., 1994)	V	GaV	M	CI	A
SIMS/Ariadne (Arens and Knoblock, 1993)	V	LaV	M	ER	A
Infomix (Leone et al., 2005)	V	GaV	M	CI	A
Hippo (Chomicki et al., 2004)	V	n/a	M	CI	A
ConQuer (Fuxman et al., 2005)	V	n/a	M	CI	A
Rainbow (Caroprese and Zumpano, 2006)	V	n/a	M	CI	A
Pegasus (Ahmed et al., 1991)	V	GaV	M	ER	I
Nimble (Draper et al., 2001)	V	unknown	unknown	ER	I
Carnot (Singh et al., 1997)	V	Lav	manual	CI	I
InfoSleuth (Nodine et al., 2000)	V	Lav	M	CI	U
Picsel2 (Reynaud and Giraldo, 2003)	V	Lav	A	CI	U
Potter's wheel (Raman and Hellerstein, 2001)	M	n/a	n/a	n/a	I
Our approach	V	GaV	A	$\mathcal{FD}$	R

- Data representation
- Sense of the mapping between the global and local schemas
- Mapping automation
- the used data reconciliation method: CI for common identifier, ER for entity reconciliation
- Data fusion capabilities: R for resolution, A for avoidance, I for ignorance, U for unknown.

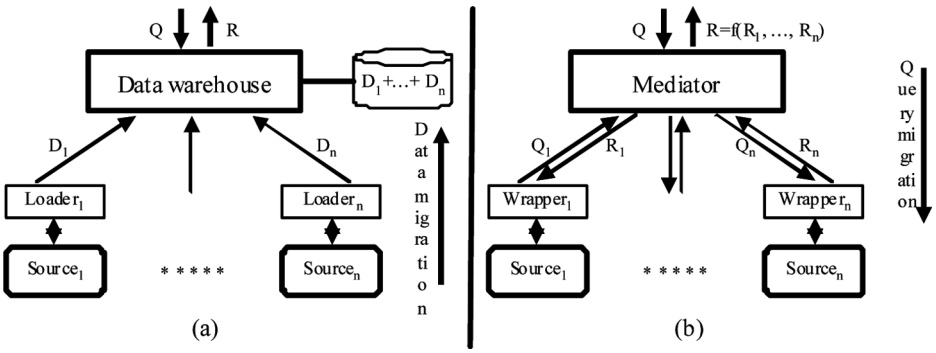
2.2.1 Data representation

This criterion specifies whether data of local sources are duplicated in a warehouse or remained in local sources and then accessed through a mediator. Two major data integration architectures are proposed in the literature: materialised (warehouse) and virtual (mediator). In the materialised architecture (Inmon and Hackathorn, 1999; Kimball and Caserta, 2004) (Figure 2(a)), a copy of data recovered from the sources is stored in a single database (called a data warehouse). The data stored in the warehouse are pre-processed in a complex way before storage; it is usually referred to as Extract, Transform and Load (ETL). In this approach, the difficulty is in the data transformation, contrary to the mediation approach, which is oriented especially towards queries rewriting. The data warehouse eliminates several problems of integration, mainly the excessively long server response times, the network clogging, or the sources unavailability. Queries can be also more easily optimised, and the data transformed at the user's discretion. These modifications, even if their usefulness can be proven, will obviously not be reflected on the local sources. A main disadvantage of this approach is that the answers to the queries can frequently be built from outdated data. Data warehouse updating can be expensive, and furthermore copyrights may exist on what certain sources provide.

In the virtual architecture (Wiederhold, 1992) (Figure 2(b)), a software called a mediator supports a virtual database (without storing data into a database), translates queries into source queries, synthesises results and returns answers to a user query. A mediator is based on a unified global schema as a synthesis of the source schemas; on this global schema queries are expressed. The most important step in using a mediator is the global schema creation. Contrary to the data warehouse approach, here the mapping deals with the relationships between the global schema and the local sources. The specification of these correspondences – according to the used method – determines the query reformulation difficulty, as well as the facility of adding or removing sources to the system. Two main concepts make up this architecture: wrappers and mediators. A wrapper wraps an information source and models the source using a source schema. A mediator maintains a global schema and mappings between the global and source schemas.

A third integration approach mixes fully materialised and fully virtual approaches and combines their advantages (Hull and Zhou, 1996). In this Figure 2(a) Materialised architecture, Figure 2(b) Virtual architecture. approach,

Figure 2 (a) Materialised architecture and (b) virtual architecture



part of the data is materialised in a warehouse whereas the rest is not. To reduce the query response time, some frequently sought data are cached. An example of a system adopting this approach is Picsel 3 (Reynaud and Safar, 2009).

2.2.2 Sense of the mapping between the global and local schemas

In *Global-as-View* (GaV) systems, the global schema is expressed as a view (a function) 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. Thus, GaV is not scalable for large applications. In the source-centric approach, each data source is expressed with one or more views over the global schema.

Therefore, *Local-as-View* (LaV) scales better, and is easier to maintain than GaV because the designer creates a global schema independently of source schemas. Then, for a new source schema, the designer has only to give (adjust) a source description that describes source relations as views of the global schema. In order to evaluate a query, a rewriting in terms of the data sources is needed. The rewriting queries using views is a difficult problem in databases (Hong et al., 2005). Thus, LaV has low performance when queries are complex.

There is a more general approach in mapping design that generalises both the GaV and the LaV paradigms. Such an approach, called *Generalised Local-as-View* (GLaV), requires the designer to associate 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 data integration environments. An example of a system adopting this approach is Fusionplex (Motro et al., 2004).

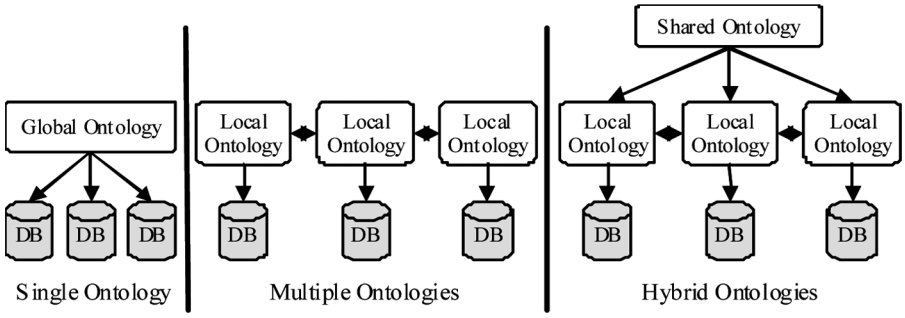
2.2.3 Mapping automation

This criterion specifies whether the mapping between the global schema and local schemas is done in a manual, a semi-automatic, or a fully automatic way.

Manual mappings are found in the first generation of integration systems that integrate sources represented by a schema and a population (i.e., each source S_i is defined as: $< Sch_i, Pop_i >$) as in classical databases and without explicit meaning representations. The manual systems focus mainly on query support and processing at the global level, by providing algorithms for identifying relevant sources and decomposing (and optimising) a global query into sub queries for the involved sources. The construction of the mediators and the wrappers used by these systems is done manually because their focus is mainly on global query processing (Castano et al., 2001).

To make the data integration process (partially) automatic, explicit representation of data meaning is necessary. Thus most of the recent integration approaches use ontologies (Castano et al., 2001; Hakimpour and Geppert, 2002; Reynaud and Giraldo, 2003). Based on the way where ontologies are utilised, we may discern three different architectures (Wache et al., 2001): single ontology methods, multiple ontologies methods, and hybrid methods (Figure 3). In the single ontology approach, each source is related to the same global domain ontology

Figure 3 Different ontology architectures



(e.g., (Lawrence and Barker, 2001; Reynaud and Giraldo, 2003; Hakimpour and Geppert, 2002) work). As a result, a new source cannot bring new or specific concepts without requiring change in the global ontology. This violates the source schematic autonomy requirement (each source can extend its schema independently). In the multiple ontologies approach (e.g., Observer Mena et al., 1996a, 1996b), each source has its own ontology developed regardless of the other sources. Then, inter-ontology relationship need to be defined. In this case, the definition of the inter-ontology mapping is very difficult as different ontologies may use different aggregation and granularity of the ontology concept (Wache et al., 2001). The hybrid approach has been proposed to overcome the drawbacks of single and multiple ontologies approaches. In this approach, each source has its own ontology, but all ontologies are connected by some means to a common shared vocabulary (e.g., KRAFT project (Visser et al., 1999)).

In all these approaches, ontologies and ontology mappings are defined at integration time. Therefore, they always request a human supervision, and they 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 ((Bhatt et al., 2006) proposes a framework to perform the extraction process). Some systems have already been proposed on that direction such as Picsel2 (Reynaud and Giraldo, 2003), and COIN (Goh et al., 1999). But to remain automatic, these systems do not allow individual data source to add new concepts and properties.

2.2.4 Data reconciliation method

Once an integration system is built it shall support user queries, by first identifying the relevant sources for a given query and then reconciling the result. To accomplish this task, two trends emerge in the current works:

Systems assuming the existence of global identifier (e.g., TSIMMIS (Chawathe et al., 1994) and Infomix (Leone et al., 2005)) suppose that different entities of sources representing the same concept have a global common identifier. This

identifier allows the reconciliation of the results of a query using relational operations (join and union and their relatives).

Sytems using statistical methods to identify similar instances (e.g., HumMer (Bilke et al., 2005) and Ajax (Bilke et al., 2005)). These systems relax the global identifier assumption and instances of a query result are compared pair wisely using athresholded similarity approach. More specifically, using a similarity measure that computes the similarity between two tuples, we classify a tuple pair as sure duplicate or non-duplicate

2.2.5 Data fusion capabilities

We can distinguish different fusion strategies (Section 2.1.2). As systems can implement one or more of these strategies, we present the systems classified by the most powerful strategy they are able to use. Existing systems can be categorised into the following four groups (Bleiholder and Naumann, 2008).

Conflict-Resolution Systems (e.g., Hermes (Adali and Emery, 1995) and Fusionplex (Motro et al., 2004)) are the most advanced systems concerning data fusion. They are able to perform conflict resolution by implementing deciding and mediating strategies based on metadata or instances.

Conflict-Avoiding Systems (e.g., SIMS (Arens and Knoblock, 1993) and ConQuer (Fuxman et al., 2005)) are the next class of systems acknowledges data conflicts and handles them by conflict avoidance. This avoidance is performed by implementing metadata-based and instance-based strategies.

Conflict-Ignoring Systems (e.g., Pegasus (Ahmed et al., 1991) and Nimble (Draper et al., 2001)) do not perform neither resolution nor avoiding methods. They simply handle conflicts by ignorance.

Finally, we consider all remaining systems that do not consider entity reconciliation or data fusion techniques in detail (e.g., Information Manifold (Levy et al., 1996) and Garlic project (Roth et al., 1996)). These systems do not handle conflicts when integrating data from different sources.

3 Background

In this section, we present concepts and definitions related to the ontologies, ontology-based databases and data reconciliation to facilitate the understanding of our proposal.

3.1 Ontologies

A crucial question that should be addressed about the design of the integration system is in which way it is in fact ontology-based. Typically, an ontology is defined as *an explicit specification of a conceptualisation* (Gruber, 1993). More precisely we consider in this paper that *an ontology is a formal and consensual dictionary of categories and properties of entities of a domain and the relationships that hold among them* (Jean et al., 2006b).

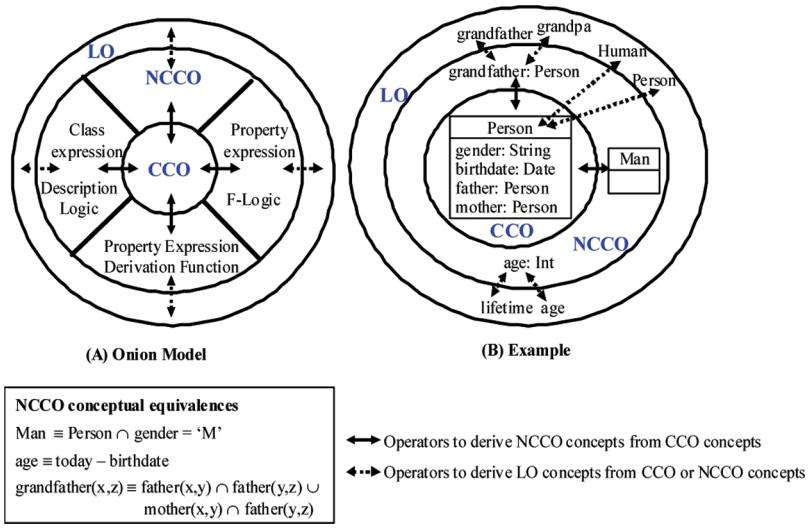
In Jean et al. (2006b), a taxonomy of ontologies, namely the onion model (Figure 4), has been proposed. It considers classes with a set of properties as the basic notion for ontology design that can be extended by different layers. The Onion Model is composed of three layers: *Conceptual Canonical* Ontologies, *Non Conceptual Canonical* Ontologies and *Linguistic* Ontologies.

1 *Conceptual Canonical Ontologies* (CCOs) contain ontologies which describe concepts of a domain without any redundancy. CCOs adopt an approach of structuring of information in term of classes and properties and associate to these classes and properties a single identifiers reusable in various languages. CCOs can be considered as shared conceptual models. They contain the core classes and play the role of a global schema in DB integration architecture.

2 *Non Conceptual Canonical Ontologies* (NCCOs) contain ontologies which also describe concepts but represent not only primitive concepts (canonical), but also definite concepts (non canonical). i.e., which can be defined from primitive concepts and/or other definite concepts. NCCOs provide mechanisms similar to views in DBs; non canonical concepts can be seen as virtual concepts defined from canonical concepts. These mechanisms may be used to represent mappings between different DBs.

3 *Linguistic Ontologies* (LOs) are those ontologies whose scope is the representation of the meaning of the words used in a particular universe of discourse, in a particular language. Beyond the textual definitions, a number of linguistics relationships (synonymous, hyponym, etc) are used to capture, in an approximate and semi-formal way, the relation between the words (Figure 4). LOs may be used to localise similarities between DB schemas, to document existing DBs or to improve the DB-user dialog.

Figure 4 The onion model (see online version for colours)



In this work, we concentrate on conceptual ontologies that may be defined formally as the quadruplet $\mathcal{O} :< \mathcal{C}, \mathcal{P}, \mathcal{S}ub, \mathcal{A}pplic >$ (Pierra, 2003), where:

- $\mathcal{C}$ is the set of classes used to describe the concepts of a given domain.
- $\mathcal{P}$ is the set of properties used to describe the instances of the $\mathcal{C}$ classes.
- $\mathcal{S}ub$ is the subsumption function defined as Sub: $\mathcal{C} \rightarrow 2^{\mathcal{C}}$. For a class of the ontology, it associates its direct subsumed classes. $\mathcal{S}ub$ defines a partial order over $\mathcal{C}$.
- $\mathcal{A}pplic$ is a function defined as $\mathcal{A}pplic : \mathcal{C} \rightarrow 2^{\mathcal{P}}$. It associates to each class of the ontology, the properties that are applicable for each instance of this class and that may be used, in the database, for describing its instances. Note that for each $c \in \mathcal{C}$, only a subset of $\mathcal{A}pplic(c)$ may be used in any particular database, for describing c instances.

Example 1: Formally, the ontology presented in Figure 6 can be presented as $O < C, P, Sub, Applic >$ with:

$\mathcal{C} = \{Organisation, Person, Student, Employee, Administrator, Faculty, Professor, GraduateStudent, Course\}$

$\mathcal{P} = \{organisationId, organisationName, WebSite, subOrganisationOf, personId, name, email, address, memberOf, adviosor, age, telephone, headOf, worksFor, teacherOf, takesCourse, courseId, courseName\}$

$\mathcal{S}ub(Organisation) = \phi$

$\mathcal{S}ub(Person) = \{Student, Employee\}$

$\mathcal{S}ub(Student) = \{GraduateStudent\}$

$\mathcal{S}ub(Employee) = \{Administrator, Faculty\}$

$\mathcal{S}ub(Administrator) = \phi$

$\mathcal{S}ub(Faculty) = \{Professor\}$

$\mathcal{S}ub(Professor) = \phi$

$\mathcal{S}ub(GraduateStudent) = \phi$

$\mathcal{S}ub(Course) = \phi$

$\mathcal{A}pplic(Organisation) = \{OrganisationId, OrganisationName, WebSite, subOrganisationOf\}$

$\mathcal{A}pplic(Person) = \{personId, name, email, address, memberOf\}$

$\mathcal{A}pplic(Student) = \{advisor, age\}$

$\mathcal{A}pplic(Employee) = \{telephone\}$

$\mathcal{A}pplic(Administrator) = \{headOf\}$

$\mathcal{A}pplic(Faculty) = \{worksFor\}$

$\mathcal{A}pplic(Professor) = \{teacherOf\}$

$\mathcal{A}pplic(GraduateStudent) = \{takesCourse\}$

$\mathcal{A}pplic(Course) = \{courseId, courseName\}.$

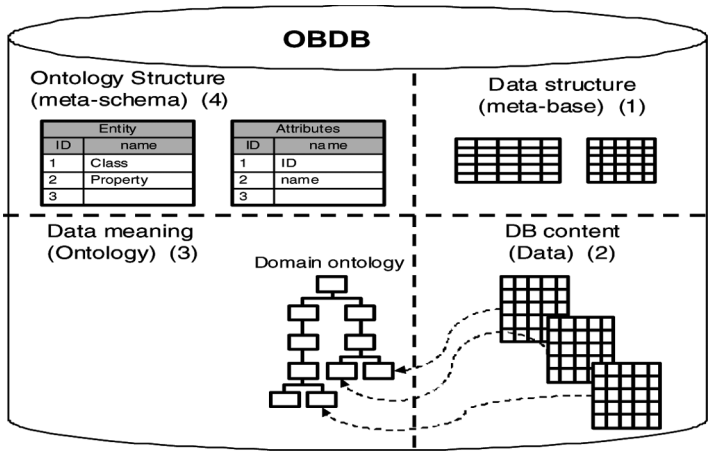
3.2 Ontology-based dataBases ($\mathcal{OBDB}$)

$\mathcal{OBDB}$ is a database that store ontologies and its instances. It can be composed of four parts (Dehainsala et al., 2007) (Figure 5). *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 catalogue. (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.

Formally, 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. The semantics of these instances is described in $\mathcal{O}$ by characterising them by classes and properties values.
- $Pop : \mathcal{C} \rightarrow 2^{\mathcal{I}}$ associates to each class its own instances. $\mathcal{P}op(c)$ constitutes the population of c.
- $\mathcal{S}ch$: $C \rightarrow 2^{P}$ associates to each ontology class c of $\mathcal{C}$ the properties, which are effectively used to describe the instances of the class c. For each class c, $\mathcal{S}ch(c)$ must satisfy: $\mathcal{S}ch(c) \subseteq \mathcal{A}pplic(c)$.

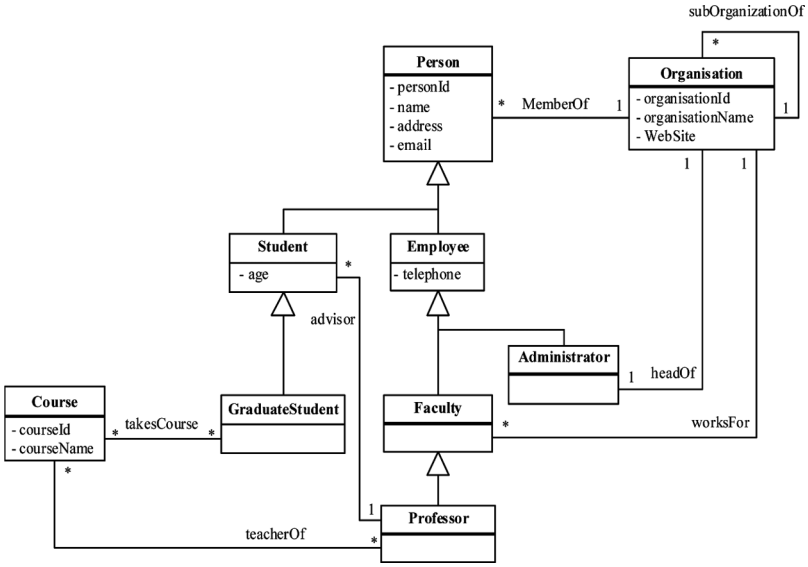
Figure 5 OntoDB architecture



4 Extension of ontology model by functional dependencies

Traditional ontology formalisms do not support $\mathcal{FD}$ in their definition. Data dependencies have been introduced as a general formalism for a large class of database constraints that augments the expressivity of database models (Abiteboul

Figure 6 UML diagram representation of a part of LUBM ontology



et al., 1995). FD compose a particularly interesting data dependency that elegantly models the relationships between attributes of a relation. $\mathcal{FD}$ are used for defining primary keys and in the normalisation theory. Other important application of $\mathcal{FD}$ in database includes *query rewriting* and *query evaluation* (Hong et al., 2005).

These benefits of $\mathcal{FD}$ could be transported to the ontology world to enrich the expressivity of the knowledge representation and data cleaning (Fan, 2008). Note that Traditional ontology formalisms do not support $\mathcal{FD}$ in their definition. Recently, couple of studies were concentrated on $\mathcal{FD}$ in the context of ontologies. Two main categories of $\mathcal{FD}$ are distinguished (Calbimonte et al., 2009; Calvanese et al., 2001; Romero et al., 2009; Toman and Weddell, 2008):

- *intra-class dependencies*
- *inter-class dependencies.*

Bellatreche et al. (2011) is an example of the first category, where $\mathcal{FD}$ are defined on properties of ontological classes as in classical databases.

The second category of $\mathcal{FD}$ involves dependency between classes. Two types are distinguished:

- $\mathcal{FD}$ with a single class in the left part
- $\mathcal{FD}$ with several classes in the left part. The work of Romero et al. (2009) is an example of the first types of this dependency.

The authors define a $\mathcal{FD}$ between classes C_1 and C_2 ($C_1 \rightarrow C_2$) when each instance of C_1 determines a single instance of C_2. In the second type, (Calbimonte et al., 2009) define a $\mathcal{FD}$ between the classes $\{C_1, \ldots, C_n\}$ linked to a root class R by properties (or a chain of properties) and a class C linked to the root class by a

property (or properties chain) ($R : \{C_1, \ldots, C_n\} \rightarrow C$), if the instances of the n classes determine a single instance of the class C.

These above research efforts are a strong incentive to extend the existing formal models of ontologies to include $\mathcal{FD}$.

4.1 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. As in traditional $\mathcal{FD}$ in databases, this definition can also be expressed as an implication: $fd\ R : LP \rightarrow RP$.

4.2 Extended formal model of ontologies

Now, we have all ingredients to extend the initial ontology model presented in Section 3 by including $\mathcal{FD}$. This extension is done 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 ($fd\ c : LP \rightarrow RP$).

Our model supports three types of $\mathcal{FD}$: *classic*, *key* and *basic*.

1 *Classic* $\mathcal{FD}$ ($fd\ R : LP \rightarrow RP$) indicates that values of the left part properties determine a unique value of the right part property.

Example 2: $fd : Person : email \rightarrow name$. Each email value of a person determines a unique name value. Formally this $\mathcal{FD}$ is written as: $(\{email\}, name) \in \mathcal{FD}\,(Person)$.

2 *Key* $\mathcal{FD}$ ($fd\ R : LP \rightarrow$) indicates that values of the left part properties determine a unique instance of the root class R. Explicitly, 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 root 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 the class R.

Example 3: $fd_k : Person : personId \rightarrow$ Each $personId$ value of a person determines a unique instance of theclass $Person$. Formally this $\mathcal{FD}$ is written as: $(\{personId\}, \phi) \in \mathcal{FD}\,(Person)$. If we suppose that all the properties of the class $Person$ are fonctional except the $address$ property, we can derive from fd_k the following FDs: $fd1 : Person$: $personId \rightarrow name$, $fd2 : Person : personId \rightarrow$ $email$ *and* $fd3 : Person : personId \rightarrow memberOf$.

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 the class 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*. This means that RP determines all candidate keys of its range class $CK(\rho(RP))$, where $\rho(p)$ is the range class of the property p.

Example 4: $fd_b : Person : \rightarrow memberOf$
Each instance of the class $Person$ determines a unique instance of the class Organisation. Formally this FD is written as: $(\phi, memberOf) \in \mathcal{FD}$ $(Person)$. $If we suppose that \{personId\}$ and $\{email\}$ are candidate keys of the class $Person$; and $\{OrganisationId\}$ *and* $\{WebSite\}$ are candidate keys of the class Organisation, we can derive from fd_b the following FDs: $fd1 : Person :\ personId \rightarrow OrganisationId$, $fd2 : Person : personId \rightarrow WebSite$, $fd3 : Person : email \rightarrow OrganisationId$ and $fd4 : Person : email \rightarrow WebSite$.

Example 5: The ontology presented in Example 1 can be extended as $O < C, P, Sub, Applic, FD >$ with:

$\mathcal{FD}(Organisation) = \{(\{OrganisationId\}, \phi), (\{WebSite\}, \phi)\}$
$\mathcal{FD}(Person) = \{(\{personId\}, \phi), (\{email\}, \phi), (\phi, memberOf)\}$
$\mathcal{FD}(Student) = \{(\{personId\}, \phi), (\{email\}, \phi)\}$
$\mathcal{FD}(Employee) = \{(\{personId\}, \phi), (\{email\}, \phi)\}$
$\mathcal{FD}(Administrator) = \{(\{personId\}, \phi), (\{email\}, \phi), (\phi, headOf)\}$
$\mathcal{FD}(Faculty) = \{(\{personId\}, \phi), (\{email\}, \phi), (\phi, worksFor)\}$
$\mathcal{FD}(Professor) = \{(\{personId\}, \phi), (\{email\}, \phi)\}$
$\mathcal{FD}(GraduateStudent) = \{(\{personId\}, \phi), (\{email\}, \phi)\}$
$\mathcal{FD}(Course) = \{(\{courseId\}, \phi)\}$.

4.3 Impact of functional dependencies on data reconciliation

Note that reconciliation of query results in a mediator architecture leads to four possible cases:

- manual reconciliation based on the experience and deep knowledge of data sources of designers which is practically impossible in the real life, where a large number of sources or instances is involved.
- Only sources having common identifiers are taken into consideration to process queries. In this case, mediator may propagate the query on sources having the common identifiers. This solution compromises the quality of returned results.
- Query results are merged, where some instances overlap which may cause error.
- Overlapping instances may be discarded using probabilistic reconciliation.

The presence of $\mathcal{FD}$ may help and facilitate the data reconciliation, especially when no common identifier is used by various sources. To illustrate this point, let us consider the following example.

Example 6: Let S_1, S_2 and S_3 be three sources containing the same relation $Person$, but with different properties as follows: 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: $fd_1 : Person : personId \rightarrow name$, $fd_2 : Person : personId \rightarrow address$, $fd_3 : Person : personId \rightarrow email$, $fd_4 : Person : email \rightarrow name$, $fd_5 : Person : email \rightarrow address$.

Suppose that the mediator schema contains a $Person$ relation with the following properties: $personId(PK)$, $name$, $address$. Suppose 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.

5 Our integration methodology

Contrary to most important systems which require the presence of all sources to perform the integration process, our system *MIRSOFT* integrates the sources based on their arrival and delete them when they quit. The mediator components are incrementally enriched when considering a new source. In this section, we first present a formal model of our integration system and the initialisation of its components, secondly we show how a new source is integrated, and finally, we detail the query processing aspect.

5.1 Formalisation 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 which 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 this 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 classes of source ontologies in correspondence with the class c.

5.2 Initialisation of the data integration system components

Before starting the integration process, *MIRSOFT* components are initialised as follows:

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, 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

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 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 integrating a new source.

Example 7: Figure 7 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:*

The mediator ontology $\mathcal{O}$ *is as shown in Figure 7.*

The schema $\mathcal{S}ch$*, the source schemas* $\mathcal{S}$ *and the mapping* $\mathcal{M}$ *are empty.*

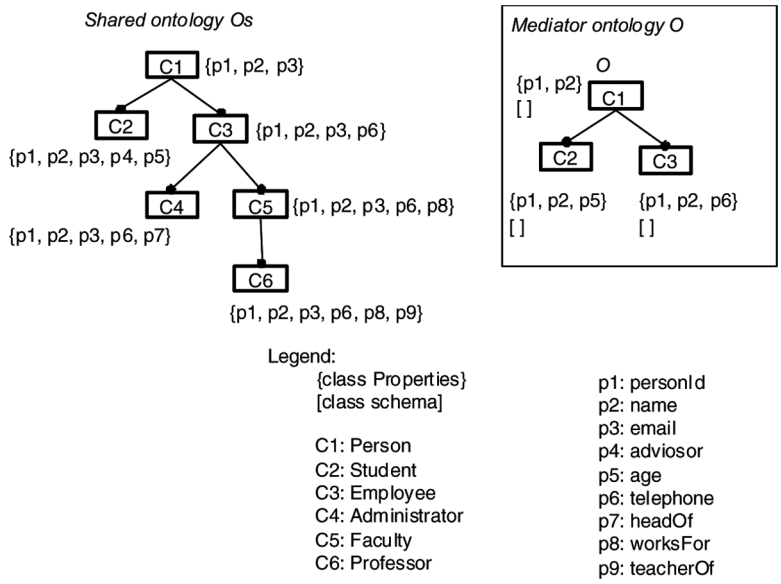
5.3 Integration of a new source

In this section, we present the scenario that consists in integrating a new source. We assume that

- each class of a local ontology references explicitly (or implicitly through its parent class) its lowest subsumption class in the shared ontology

- only properties that do not exist in the shared ontology may be defined on a local ontology.

Figure 7 Example of mediator ontology importation



We distinguish two different integration scenarios associated with integration algorithms:

- *Fragmentation scenario* where each local ontology of each source is a fragment of the shared ontology
- *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 does not exist in the shared ontology.

5.3.1 Fragmentation scenario

This integration scenario assumes that the shared ontology is rich enough to cover the needs of all local sources. Such an assumption has been used for defining the Picsel2 project (Reynaud and Giraldo, 2003) for integrating web service and in COIN (Bressan et al., 2000). Source autonomy in this scenario is materialised as follows:

- each source selects relevant subset (fragment) of the shared ontology (classes and properties)
- it designs 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 We update the source schemas $\mathcal{S}$ by adding the schema of the new source ($\mathcal{S} = \mathcal{S} \cup \{S_i :< OL_i, SchL_i >\}$)

2 In the ontology $OL_i :< CL_i, PL_i, ApplicL_i, SubL_i >$we keep only classes and properties existing in the mediator ontology ($OL_i = \mathcal{O} \cap O_i$)

3 We import the schemas of the OL_i classes from Sch_i ($\forall\ c \in CL_i\ SchL_i(c) = Sch_i(c)$)

4 We update the schema of the mediator ontology classes $\mathcal{S}ch$ 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 We update the mapping of the mediator classes 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 8: Figure 8 shows the integration of a new source following the fragmentation scenario. After this integration the components of the mediator $\mathcal{M}ed :< \mathcal{G}, \mathcal{S}, \mathcal{M} >$ contain:

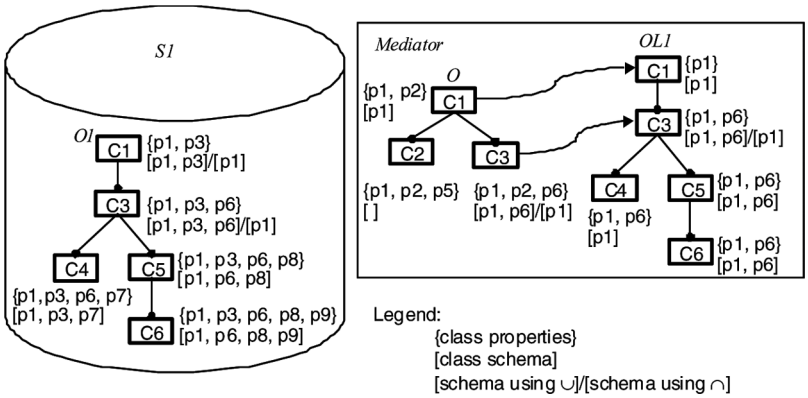
No change on the mediator ontology.

$\mathcal{S}ch(Person) = \{personId\}, \mathcal{S}ch(Student) = \phi$ 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* $SchL_i$*as shown in Figure 8.*

$\mathcal{M}(Person) = \{S_1.Person\}$, $\mathcal{M}(Student) = \phi$ and $\mathcal{M}(Employee) = \{S_1.Employee\}$.

Figure 8 Example of integrating a new source following the fragmentation scenario



5.3.2 Integrated ontology scenario

In a number of cases including e-business applications for instance, more autonomy is requested for various sources. This autonomy is characterised by the fact that each local source may have its own concepts and properties. So, each data source has its own ontology and the classes of each ontology are specific. But, all the ontologies reference the shared ontology O_s while respecting the two assumptions of Section 5.3. Therefore, 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 quintuple: $S_i :< O_i, I_i, Sch_i, Pop_i, OntoSub_i >$. Ontology is generally not static. From time to time, it needs expansion to include new concepts (Flahive et al., 2011). In this case, the shared ontology O_s does not fulfill the whole requirements of mediator users. Thus, the mediator ontology is enriched by adding new concepts defined in sources.

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 We update the mediator ontology $\mathcal{O} :< \mathcal{C}, \mathcal{P}, \mathcal{A}pplic, \mathcal{S}ub, \mathcal{FD} >$ 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 to 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:

$$\mathcal{C} = \mathcal{C} \cup C_i^+,$$
$$\mathcal{P} = P \cup P_i^+,$$
$$\mathcal{S}ub(c) = \mathcal{S}ub(c) \cup OntoSub_i(c),$$
$$\forall\ c \in C_i^+\ \mathcal{A}pplic(c) = \mathcal{A}pplic_i(c) \text{ and}$$
$$\forall\ c \in C_i^+\ \mathcal{FD}(c) = FD_i(c).$$

2 We update the source schemas $\mathcal{S}$ by adding the schema of the new source ($\mathcal{S} = \mathcal{S} \cup \{S_i :< OL_i, SchL_i >\}$).

3 In the ontology $OL_i :< CL_i, PL_i, ApplicL_i, SubL_i >$ we keep only classes and properties existing in the mediator ontology after extension ($OL_i = \mathcal{O} \cap O_i$).

4 We import the schemas of the OL_i classes from Sch_i ($\forall\ c \in CL_i\ SchL_i(c) = Sch_i(c)$).

5 We update the schema of the mediator ontology classes $\mathcal{S}ch$ 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 We update the mapping of the mediator classes by adding the mapping of the new classes and updating the mapping of the existing classes ($\forall\ c \in CL_i\ \mathcal{M}(c) = \mathcal{M}(c) \cup S_i.c$).

This scenario shows that it is possible to leave a large autonomy to each source and compute in a fully automatic, deterministic and exact way the corresponding integrated system.

We notice that when a data source uses an independent ontology without referencing a shared ontology, the task of establishing the mapping between its ontology and the mediator ontology may be done manually or semi automatically. But the rest of the integration process will be performed automatically.

Example 9: Figure 9 shows the integration of a new source following 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 is extended by adding the classes GraduateStudent and Course and their properties.

$\mathcal{C} = \{Person,\ Student,\ Employee,\ GraduateStudent,\ Course\}$,

$\mathcal{P} = \{personId,\ name,\ age,\ telephone,\ takesCourse,\ courseId,\ courseName\ \}$,

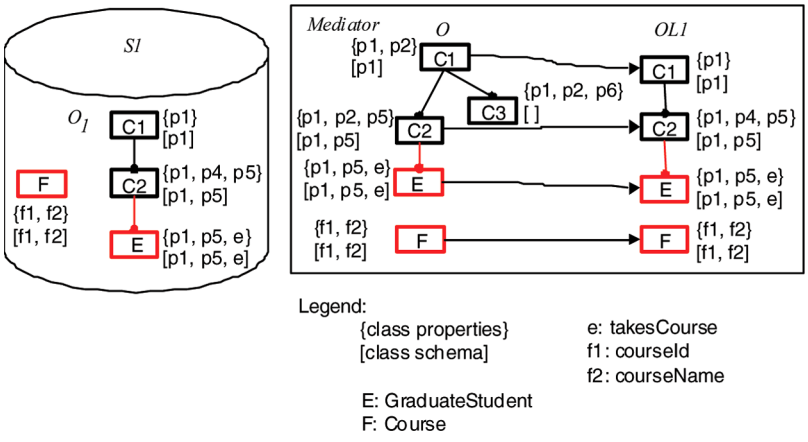
$\mathcal{S}ub(Student) = \{GraduateStudent\}$.

$\mathcal{S}ch(Person) = \{personId\}$, $\mathcal{S}ch(Student) = \{personId,\ age\}$, $\mathcal{S}ch(GraduateStudent) = \{personId, age, takesCourse\}$, $\mathcal{S}ch(Course) = \{courseId, courseName\}$ *and* $\mathcal{S}ch(Employee) = \phi$.

$\mathcal{S} = \{S_1\}$ *with* OL_i *and* $SchL_i$ *as shown in Figure 8.*

$\mathcal{M}(Person) = \{S_1.Person\}$, $\mathcal{M}(Student) = \{S_1.Student\}$, $\mathcal{M}(GraduateStudent) = \{S_1.GraduateStudent\}$, $\mathcal{M}(Course) = \{S_1.Course\}$ *and* $\mathcal{M}(Employee) = \phi$.

Figure 9 Example of integrating a new source following the integrated ontology scenario (see online version for colours)



5.4 Deleting a source from the mediator

There are many reasons to remove a source from the mediator. For example, the source is physically deleted, inaccessible, unavailable for a long time, the content is judged uninteresting, etc. To delete a source $S_i :< OL_i, SchL_i >$, we propose the following steps:

1 We update the mapping of the mediator ontology classes 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 We update eventually the schema of the mediator ontology classes 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 we delete the source from $\mathcal{S}$ by deleting its schema $SchL_i$ and its ontology OL_i.

5.5 Query answering

We consider Union of Conjunctive Queries (UCQ). 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 ordinary atom $a(x_1, \ldots, x_m)$ with $(x_1, \ldots, x_m)$ is a variables vector and a is a predicate.

$X = (x_n, \ldots, x_n)$ 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 10: 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 four steps:

- Finding the $\mathcal{FD}$ that hold in the query,
- Deriving the reconciliation key,

- Query evaluation
- Results reconciliation and fusion.

5.5.1 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

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}, are computed as: $F_{Cls_i} = F^d \cup F^k \cup F^b$.

- $F^d = fd_1 \cup fd_2 \cup \cdots \cup fd_n$ 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.

$$F^b = \{p \rightarrow CK(\rho(p)) \mid p \in JCP_i \wedge (\delta(p) : \rightarrow p) \in \mathcal{FD}(\delta(p))\}.$$

The $\mathcal{FD}$ that hold in Q_i, denoted by F_{Q_i}, are computed as:

$$F_{Q_i} = \{X \rightarrow Y \mid X \rightarrow Y \in F^+_{Cls_i}\}.$$

5.5.2 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 which has in input 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 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. We remove then the properties that can be functionally determined by an other left part in K_R.

Example 11: Consider the query $Q(x, y, z)$ of Example 10. 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: $F_{Cls_i} = F^d \cup F^k \cup F^b$

$$
\begin{aligned}
&F^d = \phi,\\
&F^k = \{personId \rightarrow email,\quad personId \rightarrow name,\quad personId \rightarrow memberOf,\\
&email \rightarrow personId,\quad email \rightarrow name,\quad email \rightarrow memberOf,\quad courseId \rightarrow\\
&courseName,\ organisationId \rightarrow organisationName\},\\
&F^b = \{memberOf \rightarrow organisationId\},\\
&K = \{personId, email, courseId, organisationId, memberOf\}.
\end{aligned}
$$

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

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 ∈ K_R do
        // P_{LP_k} = {p ∈ PP_i ∪ JCP_i | LP_k → p}: properties determined by LP_k;
        if ∃ LP_j ∈ K_R P_{LP_i} = P_{LP_j} and ∀ S_k ∈ S_{Q_i} ∀ p ∈ LP_i p ∈ SchL_k(δ(p))
        then
            K_R = K_R − {LP_j};
    foreach LP_i ∈ K_R do
        if ∃ p ∈ K_R LP_i → p then
            K_R = K_R − {p};
end
```

5.5.3 Query evaluation

Each query Q_i will be written 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:

$Q_i^{S_j}(X)$: $pr_1(X_1), \ldots, pr_n(X_n)$ where the predicate pr_i is:

- $S_j.c(x)$ with x a variable and $S_j.c(x) \in \mathcal{M}(c)$ with $c \in Cls_i$
- $p(x_1, x_2)$ with x_1 and x_2 are variables and $p \in (PP_i \cup JCP_i \cup CCP_i) \cap PL_j$
- $a(x_1, \ldots, x_m)$ with $\forall\ x_i \in \{x_1, \ldots, x_m\}\ p(x_j, x_i) \in Q_i^{S_j}$.

The source S_j does not necessarily valuate all the properties of JCP_i, therefore the query $Q_i^{S_j}$ may not be valid on this source (e.g., $Q_i(x)$: *Person*(x, y), *memberOf*(x, y), *Department*(y). In a source S_j where the property $memberOf$ is not valued, the query $Q_i^{S_j}(x, y)$: $Person(x), Department(y)$ is no valid because it returns a Cartesian product). So, we test the validity of the query, if it is not valid, we can answer only a part of this query (e.g., $Q_i^{S_j}(x)$:- $Person(x)$ for the previous query). To do so, the Algorithm 2 is used to set the projected properties $PP_i^{S_j}$,

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 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 started 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.

5.5.4 Results reconciliation and fusion

To avoid redundancy and conflicting information, data integration systems implement data reconciliation and fusion techniques to find the true values, but most of these methods need to query all sources and are designed for offline data aggregation that can take a long time. Instead of this approach, we propose an *incremental* and *online* reconciliation and fusion method allowing to return a primary answer as soon as possible. The system starts with returning answers from the first concerned source and refreshes the answers as it asks more sources by applying reconciliation and fusion techniques on the retrieved data. For each returned answer, it shows the asked sources, and stops retrieving data at the request of the user satisfied by the current answer.

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

5.6 Implementing our methodology

To implement our methodology, several choices are feasible to persist the component of the mediator (the global schema $< \mathcal{O}, \mathcal{S}ch >$, source schemas $< OL, SchL >$ and mappings $\mathcal{M}$).

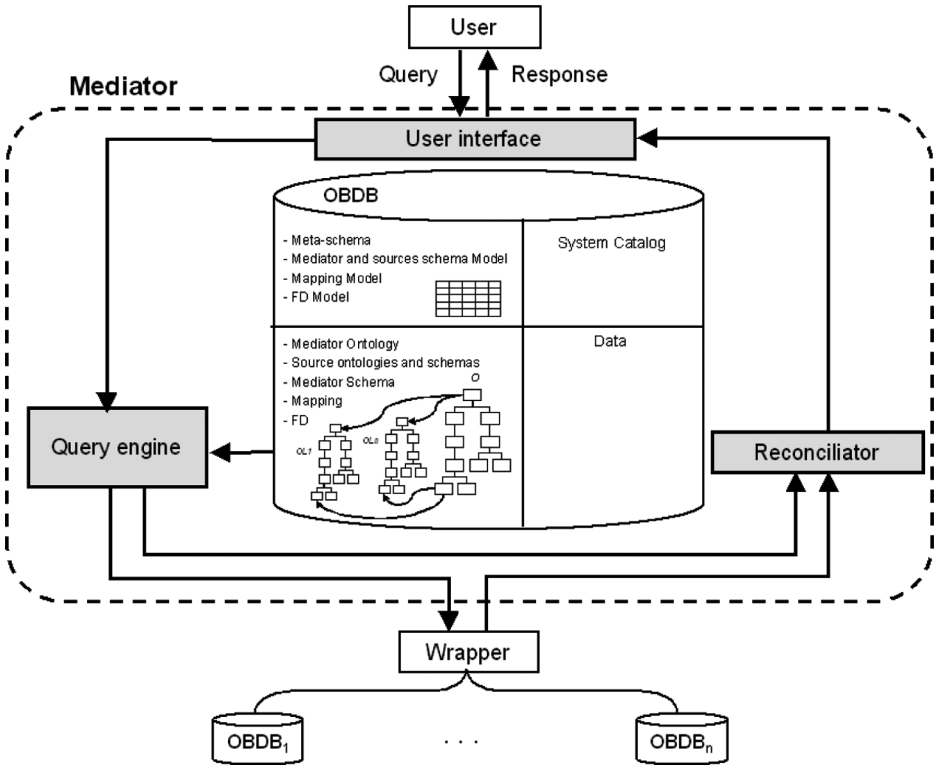
- First, using text files (XML, OWL, ...) which is a simple structure to implement but the interrogation of ontologies became costly when the number of concepts is very important.
- Second, using a database we can accelerate the interrogation but we need to develop a support for ontology and a query interface or language to query ontologies.
- Finally, using a 4-quarters ontology-bases database that offer a support of ontologies, a query language to interrogate ontologies, an extendible meta-schema allowing to add $\mathcal{FD}$ to ontologies and a data part that can be used as a caching to accelerate the response time of queries; the only drawback of this choice is that it needs a little more effort to implement it for the first time. From advantages and drawbacks of these three choices, we opt for the use of an $\mathcal{OBDB}$.

5.6.1 Overview of the implemented architecture

Different modules composing our integration system are described in Figure 10:

- an *OBDB* repository
- a user interface
- a query engine
- a result reconciliator.

Figure 10 Different modules composing our integration system



1 The *OBDB repository*: Our mediator uses the same structure as the used sources participating in the integration process. It follows *OntoDB* model (Dehainsala et al., 2007), where the meta-schema part (Section 2) is extended by

   i a mediator and source schema model,

   ii a model of mapping between the mediator ontology and source ontologies

   iii a *FD* model.

In the ontology part, we store the mediator ontology, source ontologies and schemas, the mapping between the mediator ontology and source ontologies and the $\mathcal{FD}$ between the classes and properties of the mediator ontology. The data part can be used as a caching to optimise frequently queries (in this work this issue is not addressed).

2 The *user interface*: It allows the user to express her or his query and displays its results. After parsing the input query, the user interface send to the query engine a conjunctive queries defined on a set of classes and properties of the mediator ontology. The user interface is responsible also on displaying answers from the first visited sources and refreshing the answers when the answers of more sources coming from the reconciliator are available.

3 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

   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.

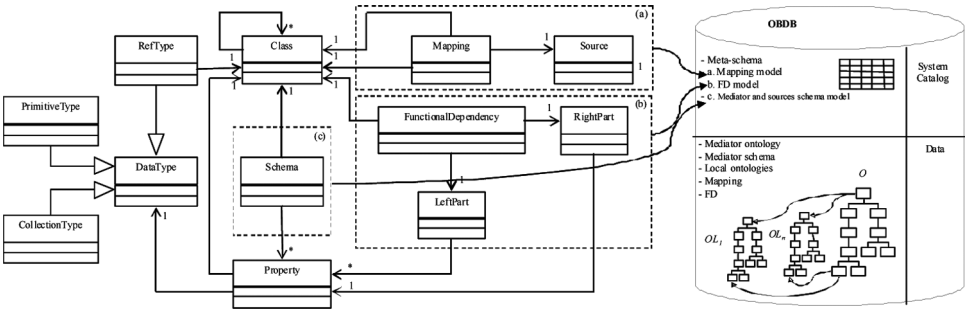
4 The *result reconciliator*: the role of this module is to reconcile the results using the reconciliation key, to merge instances refering to the same real world following the selected technic and to send progressively obtained results to the user interface in an incremantal way.

5.6.2 Make persistency of $\mathcal{FD}$ and the data integration system components

The meta-schema of OntoDB contains two main tables *Entity* and *Attribute* encoding the meta-model level. *Entity* describes ontological concepts like class, property or data type. *Attribute* describes attributes related to each ontological concept (*name*, *description*, *comment* ...). An extension of the meta-schema of OntoDB is proposed to support the $\mathcal{FD}$, the mapping and the schemas of the classes of mediator, source ontologies.

Precisely, three meta-models describing the $\mathcal{FD}$ concepts, mapping concepts and class schemas concepts are first proposed (Figure 11). Secondly the three

Figure 11 Extension of OntoDB meta-schema



meta-models are instanciated in the meta-schema of OntoDB using OntoQL–an ontology query language proposed to querying OntoDB (Jean et al., 2006a). The following statements encode these instantiations.

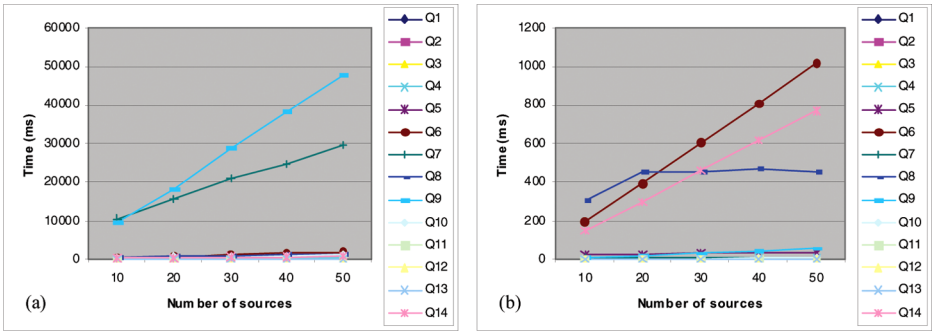
```
CREATE ENTITY #FD(#ItsClass REF (#Class), #ItsLeftProperties REF (#Property)ARRAY, #ItsRightProperty REF (#Property));

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));
```

6 Validation of our architecture

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.

Two main experiments are conducted to evaluate the scalability of our system based on the number of sources and instances. Figure 12 shows the results of executing 14 queries in (millisecond) by varying the number of sources participating in the integration process from 10 to 50 (following a fragmentation scenario). Generated sources have the same schema (22 relations). The biggest relation has 2000 tuples. The obtained results show that our system executes efficiently 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 (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. 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),

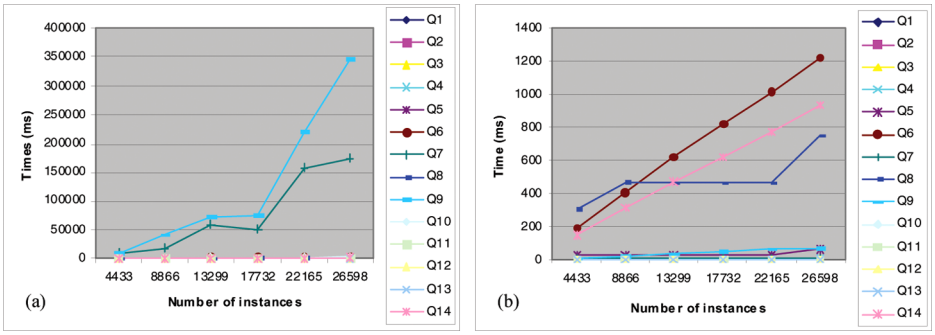
Figure 12 (a) Query response time and (b) reconciliation time vs. number of sources (see online version for colours)



and local query evaluation. Figure 12(b) points out that the time needed for mediator processing is negligible w.r.t. the overall execution time, and that the major time consuming process is the evaluation of the query over the sources. However, the reconciliation time of queries ($Q_6(x)$:- *Student*(x) and $Q_{14}(x)$: *UndergraduateStudent*(x)) increases progressively because the number of instances of these queries results is very important.

In the same direction of the previous experiment, we conduct another one by varying the number of instances of 10 used sources. Figure 13 shows the obtained results. Also, we notice that the high costly queries are Q_7 and Q_9 considered as low costly queries in the first experiments, but when the number of instances increases, join operation becomes costly where these queries (Q_7 and Q_9) become costly. 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 13 (a) Query response time and (b) reconciliation time vs. number of instances (see online version for colours)



7 Conclusion

Actually, we assist to an exposition of data and data sources over the web. As a result, integration solutions have become an important phase for many applications in various domains: engineering, medicine, traveling, biology, etc. In parallel to this trend, ontologies have been largely developed in these domains. Consequently, a large amount of ontological instances have been generated. To facilitate their management, commerical and academic database management systems offer solutions to store, query and manage these data. These efforts gave rise to a new type of databases called, *ontology-based databases*. They store in the same repository the data and the ontology describing their sense. As a consequence, these databases are potential a candidate for data integration. The presence of the ontologies within these database sources may contribute in resolving the heterogeneity. Note that most of the actual integration systems follow two extreme scenarios for data reconciliation:

1 Some suppose that the manipulated sources have similar keys to ensure data integration. This usually violates the autonomy characteristic of sources.

2 Others use statistical techniques to reconcile data.

In sensitive domains, where exact solutions are required from data integration systems such techniques cannot be used. In this paper, we proposed a complete ontology-based integration methodology, called, MIRSOFT, for ontology-based sources. It deals with heterogeneity by the means of ontologies enriched by functional dependencies defined on each ontological class. These functional dependencies allows the relaxation of the two extreme scenarios for the data reconciliation. Finally, our approach is evaluated using the dataset of the Lehigh University Benchmark. The obtained results show its efficiency and scalability.

We are currently running more experiments using a large scale dataset to evaluate the real efficiency of MIRSOFT. An important issue that should be considered is the proposition of quality metrics for integration systems taking into account the use of reconciliation methods.

References

Abiteboul, S., Hull, R. and Vianu, V. (1995) *Foundations of Databases: The Logical Level*, 1st ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

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

Ahmed, R., De Smedt, P., Du, W. Kent, W., Ketabchi, M.A., Litwin, W.A., Rafii, A. and Shan, M-C. (1991) 'The pegasus heterogeneous multi-database system', *IEEE Computer*, Vol. 24, No. 12, December, pp.19–27.

Arens, Y. and Knoblock, C.A. (1993) 'Sims: retrieving and integrating information from multiple sources', *SIGMOD*, May, pp.562,563.

Batini, C. and Scannapieco, M. (2010) *Data Quality: Concepts, Methodologies and Techniques*, 1st ed., Springer, Secaucus, NJ, USA.

Bellatreche, L., Ait-Ameur, Y. and Chakroun, C. (2011) 'A design methodology of ontology based database applications', *Logic Journal of the IGPL*, Vol. 19, No. 5, pp.648–665.

Bellatreche, L., Xuan, D.N., Pierra, G. and Dehainsala, H. (2006) 'Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases', *Computers in Industry Journal Elsevier*, Vol. 57, Nos. 8–9, pp.711–724.

Bhatt, M., Flahive, A., Wouters, C., Rahayu, W. and Taniar, D. (2006) 'Move: a distributed framework for materialized ontology view extraction', *Algorithmica*, Vol. 45, No. 3, pp.457–481.

Bilke, A., Bleiholder, J., Naumann, F., Böhm, C., Draba, K. and Weis, M. (2005) 'Automatic data fusion with hummer', *Proceedings of the International Conference on Very Large Databases*, Trondheim, Norway, pp.1251–1254.

Bleiholder, J. and Naumann, F. (2008) 'Data fusion', *ACM Computing Surveys*, Vol. 411, No. 1, pp.1–41.

Bressan, S., Goh, C., Levina, N., Madnick, S., Shah, A. and Siegel, M. (2000) 'Context knowledge representation and reasoning in the context interchange system', *Applied Intelligence*, Vol. 13, No. 2, September, pp.165–180.

Broekstra, J., Kampman, A. and van Harmelen, F. (2002) 'Sesame: a generic architecture for storing and querying rdf and rdf schema', *International Semantic Web Conference*, Springer, Sardinia, Italy, pp.54–68.

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

Calvanese, D., Giacomo, G. and Lenzerini, M. (2001) 'Identification constraints and functional dependencies in description logics', *Proceedings of Joint Conference on Artificial Intelligence (JCAI'2001)*, Seattle, Washington, USA, pp.155–160.

Caroprese, L. and Zumpano, E. (2006) 'A framework for merging, repairing and querying inconsistent databases', *Proceedings of the 10th East European Conference on Advances in Databases and Information Systems (ADBIS'2006)*, Thessaloniki, Greece, pp.383–398.

Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. and Wilkinson, K. (2004) 'Jena: implementing the semantic web recommendations', *Proceedings of the 13th international World Wide Web Conference (Alternate Track Papers & Posters)*, New York, USA, pp.74–83.

Castano, S., Antonellis, V. and Vimercati, S.D.C. (2001) 'Global viewing of heterogeneous data sources', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 13, No. 2, pp.277–297.

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

Chomicki, J., Marcinkowski, J. and Staworko, S. (2004) 'Hippo: a system for computing consistent answers to a class of sql queries', *Proceedings of International Conference on Extended Database Technology (EDBT)*, Crete, Greece, pp.841–844.

Das, S., Chong, E.I., Eadon, G. and Srinivasan, J. (2004) 'Supporting ontology-based semantic matching in rdbms', *Proceedings of the International Conference on Very Large Databases*, Toronto, Canada, pp.1054–1065.

Dayal, U. (1983) 'Processing queries over generalization hierarchies in a multidatabase system', *Proceedings of the International Conference on Very Large Databases*, Florence, Italy, pp.342–353.

Dehainsala, H., Pierra, G. and Bellatreche, L. (2007) 'Ontodb: An ontology-based database for data intensive applications', *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'2007)*, Bangkok, Thailand, pp.497–508.

Dong, X.L. and Naumann, F. (2009) 'Data fusion – resolving data conflicts for integration', *PVLDB*, Vol. 2, No. 2, pp.1654–1655.

Draper, D., Halevy, A.Y. and Weld, D.S. (2001) 'The nimble integration engine', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Santa Barbara, California, USA, pp.567–568.

Fan, W. (2008) 'Dependencies revisited for improving data quality', *PODS*, pp.159–170.

Flahive, A., Taniar, D., Rahayu, J.W. and Apduhan, B.O. (2011) 'Ontology expansion: appending with extracted sub-ontology', *Logic Journal of the IGPL*, Vol. 19, No. 5, 618–647.

François Goasdoué, F., Lattès, V. and Rousset, M.C. (2000) 'The use of carin language and algorithms for information integration: the picsel system', *International Journal of Cooperative Information Systems (IJCIS)*,Vol. 9, No. 4, December, pp.383–401.

Fuxman, A., Fuxman, D. and Miller, R.J. (2005) 'Conquer: a system for efficient querying over inconsistent databases', *Proceedings of the International Conference on Very Large Databases*, Trondheim, Norway, pp.1354–1357.

Goh, C., Bressan, S., Madnick, E. and Siegel, M.D. (1999) 'Context interchange: new features and formalisms for the intelligent integration of information', *ACM Transactions on Information Systems*, Vol. 17, No. 3, pp.270–293.

Gruber, T. (1993) 'A translation approach to portable ontology specification', *Knowledge Acquisition*, Vol. 5, No. 2, pp.199–220.

Hakimpour, F. and Geppert, A. (2002) 'Global schema generation using formal ontologies', *ER*, pp.307–321.

Halevy, A.Y., Ashish, N., Bitton, D., Carey, M.J., Draper, D., Pollock, J., Rosenthal, A. and Sikka, V. (2005) 'Entreprise information integration: successes, challenges and controversies', *SIGMOD*, pp.778–787.

Hong, J., Liu, W., Bell, D.A. and Bai, Q. (2005) 'Answering queries using views in the presence of functional dependencies', *BNCOD*, pp.70–81.

Hull, R. and Zhou, G. (1996) 'A framework for supporting data integration using the materialized and virtual approaches', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, pp.481–492.

Inmon, W.H. and Hackathorn, R.D. (1994) *Using the Data Warehouse*, Wiley-QED Publishing, Somerset, NJ, USA.

Jean, S., Ait-Ameur, Y. and Pierra, G. (2006a) 'Querying ontology based databases. the ontoql proposal', *Software Engineering and Knowledge Engineering (SEKE)*, pp.166–171.

Jean, S., Pierra, G. and Ait-Ameur, Y. (2006b) 'Domain ontologies : a database-oriented analysis', *Proceedings of International Conference on Web Information Systems and Technologies (WEBIST)*, Setúbal, Portugal, pp.341–351.

Kimball, R. and Caserta, J. (2004) *Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*, Wiley, Indianapolis, USA.

Köpcke, H. and Rahm, E. (2010) 'Frameworks for entity matching: a comparison', *Data & Knowledge Engineering*, Vol. 69, No. 2, pp.197–210.

Lawrence, R. and Barker, K. (2001) 'Integrating relational database schemas using a standardized dictionary', *Proceedings of the ACM Symposium on Applied Computing (SAC)*, March, pp.225–230.

Leone, N., Greco, G., Ianni, G., Lio, V., Terracina, G., Eiter, T., Faber, W., Fink, M., Gottlob, G., Rosati, R., Lembo, D., Lenzerini, M., Ruzzi, M., Kalka, E., Nowicki, B. and Staniszkis, W. (2005) 'The infomix system for advanced integration of incomplete and inconsistent data', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Baltimore, Maryland, USA, pp.915–917.

Levy, A.Y., Rajaraman, A. and Ordille, J.J. (1996) 'The world wide web as a collection of views: Query processing in the information manifold', *Proceedings of the International Workshop on Materialized Views: Techniques and Applications (VIEW'1996)*, June, pp.43–55.

Liu, X., Dong, X.L., Ooi, B.C. and Srivastava,'D. (2011) 'Online data fusion', *Proceedings of the International Conference on Very Large Databases*, Vol. 4, No. 11, pp.932–943.

Lu, J., Ma, L., Zhang, L., Brunner, J-S., Wang, C., Pan, Y. and Yu, Y. (2007) 'Sor: a practical system for ontology storage, reasoning and search', *Proceedings of the International Conference on Very Large Databases*, Vienna, Austria, pp.1402–1405.

Mena, E., Kashyap, V., Sheth, A.P. and Illarramendi, A. (1996a) 'Observer: an approach for query processing in global information systems based on interoperation across pre-existing ontologies', *CoopIS*, pp.14–25.

Mena, E., Vipul Kashyap, V., Illarramendi, A. and Sheth, A.P. (1996b) 'Managing multiple information sources through ontologies: relationship between vocabulary heterogeneity and loss of information', *Proceedings of Third Workshop on Knowledge Representation Meets Databases (KRDB)*, Budapest, Hungary.

Motro, A., Berlin, J. and Anokhin, P. (2004) 'Multiplex, fusionplex and autoplex: three generations of information integration', *Sigmod Record*, Vol. 33, pp.51–57.

Naumann, F., Bilke, A., Bleiholder, J. and Weis, M. (2006) 'Data fusion in three steps: resolving inconsistencies at schema-, tuple-, and value-level', *Bulletin of The Technical Committee On Data Engineering*, pp.21–31.

Nguyen, H-Q., Taniar, D., Rahayu, J.W. and Nguyen, K. (2011) 'Double-layered schema integration of heterogeneous xml sources', *Journal of Systems and Software*, Vol. 84, No. 1, pp.63–76.

Nodine, M., Fowler, J., Ksiezyk, T., Perry, B., Taylor, M. and Unruh, A. (2000) 'Active information gathering in infosleuth', *International Journal of Cooperative Information Systems*, Vol. 9, Nos.1–2, pp.3–28.

Pierra, G. (2003) 'Context-explication in conceptual ontologies: the plib approach', *Proceedings of the 10th ISPE International Conference on Concurrent Engineering (ISPE CE 2003)*, Madeira, Portugal, pp.243–253.

Rahm, E. and Bernstein, P. (2001) 'A survey of approaches to automatic schema matching', *VLDB Journal*, Vol. 10, pp.334–350.

Raman, V. and Hellerstein, J.M. (2001) 'Potter's wheel: an interactive data cleaning system', *Proceedings of the International Conference on Very Large Databases*, Roma, Italy, pp.381–390.

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

Reynaud, C. and Safar, B. (2009) 'Construction automatique d'adaptateurs guide par une ontologie pour l'integration de sources et de donnes xml', *Technique et Science Informatiques (TSI)*, Vol. 28, pp.199–228.

Romero, O. and Abelló A. (2010) 'A framework for multidimensional design of data warehouses from ontologies', *Data Knowledge Engineering (DKE)*, Vol. 69, No. 11, pp.1138–1157.

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

Roth, M.T., Arya, M., Haas, L., Carey, M., Cody, W., agin, R., Schwarz, P., Thomas, J. and Wimmers, E. (1996) 'The garlic project', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, p.557.

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

Sarma, A.D., Dong, X.L. and Halevy, A.Y. (2011) 'Data integration with dependent sources', *Proceedings of 14th International Conference on Extending Database Technology (EDBT)*, Uppsala, Sweden, pp.401–412.

Singh, M.P., Cannata, P.E., Huhns, M.N., Jacobs, N., Ksiezyk, T., Ong, K., Sheth, A.P., Tomlinson, C. and Woelk, D. (1997) 'The carnot heterogeneous database project: implemented applications', *Distributed and Parallel Databases Journal*, Vol. 5, April, pp.207–225.

Toman, D. and Weddell, G.E. (2008) 'On keys and functional dependencies as first-class citizens in description logics', *J. of Automated Reasoning*, Vol. 40, Nos. 2,3, pp.117–132.

Ullman, J.D. (1997) 'Information integration using logical views', *Proceedings of the International Conference on Database Theory (ICDT)*, Delphi, Greece, pp.19–40.

Visser, P.R.S., Beer, M., Bench-Capon, T., Diaz, B.M. and Shave, M.J.R. (1999) 'Resolving ontological heterogeneity in the kraft project', *Proceedings of 10th International Conference on Database and Expert Systems Applications (DEXA)*, Florence, Italy, pp.668–677.

Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H. and Hübner, S. (2001) 'Ontology-based integration of information – a survey of existing approaches', *Proceedings of the International Workshop on Ontologies and Information Sharing*, August, pp.108–117.

Wiederhold, G. (1992) 'Mediators in the architecture of future information systems', *IEEE Computer*, Vol. 25, No. 3, pp.38–49.

Xuan, D.N., Bellatreche, L. and Pierra, G. (2006) 'A versioning management model for ontology-based data warehouses', *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, Krakow, Poland, pp.195–206.

Yin, X., Han, J. and Yu, P.S. (2008) 'Truth discovery with multiple conflicting information providers on the web', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 20, No. 6, June, pp.796–808.

Zhao, H. and Ram, S. (2008) 'Entity matching across heterogeneous data sources: an approach based on constrained cascade generalization', *Data & Knowledge Engineering*, Vol. 66, No. 3, pp.368–381.

Note

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